universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

MEDIATORS IN A DISTRIBUTED ENVIRONMENT

Verfasserin:

Barbara Selista

angestrebter akademischer Grad:
Magistra der Sozial- und Wirtschaftswissenschaften
(Mag.rer.soc.oec.)

Wien, November 2008

Studienkennzahl: A175
Studienrichtung: Wirtschaftsinformatik
Betreuer: Univ.-Prof. 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 übereinstimmt.

Wien, im November 2008 Barbara Selista

Abstract

In the past data was stored in centralized databases, handled and manipulated by just few persons or programs. Today nearly every company structure has changed. The departments and data sources that they produce are distributed all over the world and more people and different soft- and hardware is involved.

There exist different solutions to store structured, semi-structured data and objects. The most common are flat file storage, relational DBs, XML DBs and object-oriented databases. Beside these conceptual differences, also database administrators can design different schemas for the same task and storage environment. This leads to three kinds of conceptual data partitioning. Horizontal partitioning means data of the same type divided into multiple databases. Vertical partitioning stands for related data stored in multiple databases and combined via a key join. Partitioning over heterogeneous data sources means handling data that can be different in format, type or database. To handle these differences, it is necessary to map and integrate the involved data according to one global schema. This is the task of mediation.

The wrapper/mediator approach is the most commonly used approach to perform queries against a mediated schema. The wrapper part is responsible for the low level access of a data source, to hide its data representation specifics and to translate and process queries. The mediator part provides a unified schema for the different schemata of the data sources and transforms the unified schema queries into one or more data resource queries that can be executed by the wrapper. It can be implemented as central mediator or as distributed mediator. In the latter case a name server keeps track of the participating peers. If cost information for each peer query is available query optimization can take place. The AMOS II system follows this wrapper/mediator approach.

Grid computing on the other hand focuses on collaborative usage of computing resources. Loose coupling and easy access of these resources are the main requirements. The OGSA-DAI middleware allows the uniform access to different data sources via Grid and Web services.

The combination of the advantages of the wrapper/mediator approach and grid computing has not fully been investigated. OGSA-DAI's DQP (Distributed Query Processing) is one advance in this direction. It enables queries in a declarative language over multiple OGSA-DAI data resources and other grid services. Nevertheless, a fully featured wrapper/mediator within OGSA-DAI has not yet been implemented. This Master Thesis discusses the possibilities of combining the wrapper/mediator approach with grid computing. It identifies the main features of both approaches and examines how they can work together and which limitations occur. The practical part of this Thesis attempts to extend OGSA-DAI with new activities to make AMOS II accessible from within OGSA-DAI. Therefore it uses the AMOS II Java call interface to forward AmosQL queries to the peers. To evaluate the prospects of this approach the provided functionality is discussed and performance comparisons are undertaken.

To my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Today data and computing resources are often distributed over networks. This leads to challenges when attempting to utilize these informations and computing power. Different data storage technologies like relational or XML databases are used and the data can be horizontally or vertically partitioned (see [OV99]). In the history of information technology various solutions have been developed for these challenges.
Grid computing on the one hand simplifies the usage of distributed computing resources and data. OGSA-DAI (see [AOD]) is one of the most advanced approaches with a focus on shared data. Beside its complex functionality it provides extension points so that new features can be easily and seamlessly integrated. Mediators on the other hand focus on the unified access to heterogeneous data resources. As these data resources are often stored using different data storage technologies a need to provide access to these resources arises. Therefore for example Amos II (see [RJK03]) follows the wrapper/mediator approach where the wrapper takes care of this task.
As both approaches have their strengths and weaknesses this paper discusses the combination of the two representatives OGSA-DAI and Amos II. To allow an estimation how these two software components can be used together and integrated a description of the functionality and the extension points of each component is given.

1.1.1 Use Case

To give a more concrete example why a grid and a mediator solution could be used together a simple use case is assumed. Data describing some *persons* is stored in two databases. The noteworthiness is the fact that information about each person is distributed over two heterogeneous databases. Therefore mediation is necessary to get all available information for each person.

Figure 1.1 shows the case where the client is responsible for the mediation step. Two database queries are submitted by the client to fetch all distributed data and the mediation is undertaken by the client when the results of the queries are available. The advantages of using a Grid middleware is described in detail in Section 4.5.

Figure 1.2 shows the case where a dedicated mediator is responsible for the mediation step, this figure shows also the part of a *centralized* mediator. The client sends a query against a mediated schema to the dedicated mediator and this mediator is responsible for

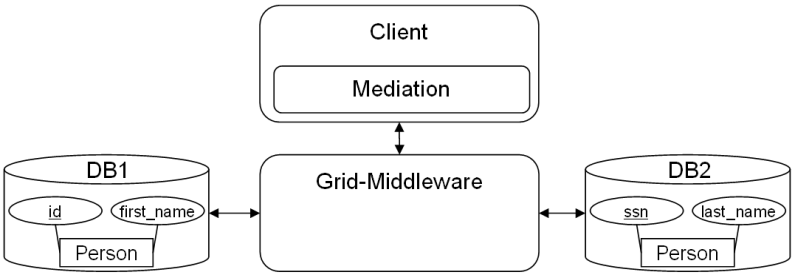

Figure 1.1: Client-side mediation

the retrieving of the mediated results. Therefore no further mediation steps are required on behalf of the client. In this case the client needs not to be aware of the heterogeneous databases. Knowledge about the often simpler mediated schema and an appropriate query language (not necessary SQL for relational databases) to formulate the query is sufficient.

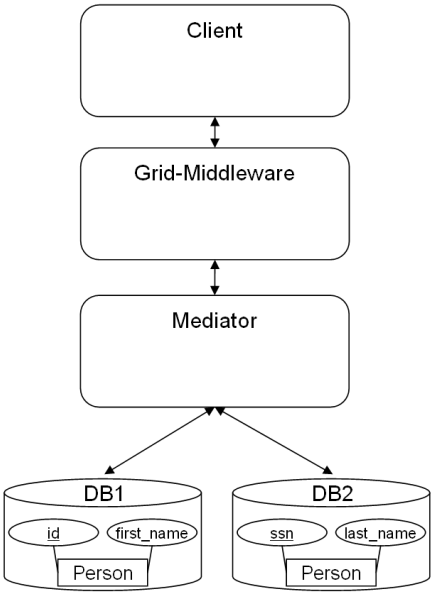

Figure 1.2: Centralized Mediation

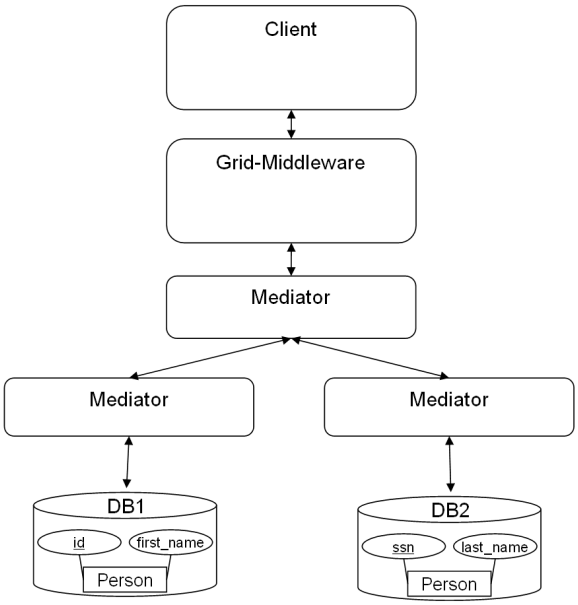

Figure 1.3: Distributed Mediation

Within distributed environments also corresponding to the *centralized* mediators, *distributed* mediators are possible, meaning that more than one mediator is responsible for the mediation within this environment. The system of Amos II uses the mediator/wrapper approach and it is possible to start Amos II as a centralized mediator and as a distributed mediator, meaning that more than one mediator is involved. Figure 1.3 shows a distributed mediator system with more than one involved mediators. Within our thesis the different distributed mediators will be named as *Amos II peers*.

1.2 Goals

The main goal of this thesis is the adding of the distributed mediators functionality provided by Amos II to the grid middleware OGSA-DAI. This middleware owes its functionality to a great deal of its activities. These activities target different tasks like data retrieval, transformation or delivery. The possibility to combine the activities allows this

grid framework to provide functionality that exceeds the sum of its parts. Therefore an effort is undertaken to develop activities that follow the guidelines and best-practices provided by the OGSA-DAI documentation. A further goal are justifiable performance losings compared for example to the direct access to the mediator. The final requirement is the capability to handle distribution. Therefore it should be possible that the OGSA-DAI middleware, the Amos II mediators and the data resources are distributed over a network.

1.3 Results

The main software artifacts developed for this thesis can be divided in three groups:

- A data resource providing access to an Amos II peer from OGSA-DAI. The grid framework and the peer can be distributed.
- Schema and data retrieval activities for a possible mediated schema in an Amos II mediator instance also known as peer. This provides the promised mediator functionality if a schema is properly configured in an Amos II peer. Queries against the mediated schema can be performed using the AmosQL queries (see [FHJ+]).
- Schema and data retrieval activities for relational databases accessible via a wrapper in an Amos II peer. This functionality allows to access these databases in an almost identical manner compared to the relational activities provided by OGSA-DAI (see [AOD]). For example the same SQL query can be used if an identical database is accessible once directly via OGSA-DAI and once via OGSA-DAI and wrapped in an Amos II peer.

Beside these core components much utility functionality is needed to adapt the various interfaces used by Amos II and OGSA-DAI. For example a XML representation of a possible mediated schema in an Amos II peer is provided.

Figure 1.4 shows the implemented new activities within OGSA-DAI to access Amos II. As in the figure described the new implementation contains five new activities named as *AmosSchema*-Activity, *AmosQLQuery*-Activity, *AmosGetAvailableTables*-Activity, *AmosExtractTableSchema*-Activity and *AmosSQLQuery*-Activity. The complete design and implementation of this thesis can be found in Chapter 5.

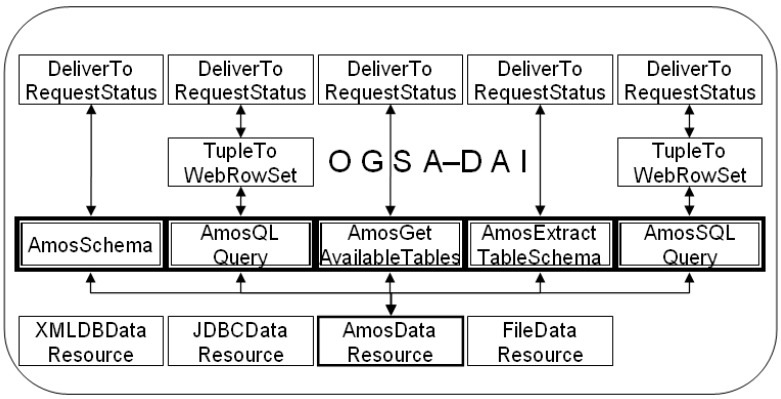

Figure 1.4: Implemented new activities

1.4 Document Organization

This thesis describes in the first chapter after the introduction the reasons why distributed systems and grid solutions have been developed within the last few years. When thinking of different databases also other topics need to be mentioned, for example the distribution of these databases and also the heterogeneities that can occur. Within the second chapter a short overview of grid computing is given. In the following chapter a detailed description of the parts of Amos II relevant for this thesis is given. For example the main architecture of Amos II, but also the functional data model of Amos II. As we tried to retrieve the schema of an Amos II peer with our activities it is necessary to know what the different parts involved within Amos II functional data model do stand for. Derived types are necessary for the performing of the mediation conception within Amos II. Also the query processing is mentioned, explaining how the query processing is done within Amos II. The grid framework OGSA-DAI is described afterwards completing the overview of the knowledge needed to understand the design and implementation of the data resource and activities. A short overview of the framework is given, explaining what activities are possible and also showing the workflows that are possible. OGSA-DAI is extensible and therefore it is quite interesting extending the functionality of OGSA-DAI with the purpose of writing new activities that can access, with the help of a new data resource, the Amos II system. The resource and the activities are discussed from the requirements to the design, the implementation and simple usage examples. More complex use cases are provided in the last chapter where a more detailed discussion of the functionality and performance aspects of the activities is provided.

Chapter 2

Background: Distributed Systems, Databases and Grid Computing

As already mentioned in the introductory chapter, the main purpose of this work is to couple two different systems, on the one hand a grid solution (OGSA-DAI) and on the other hand a wrapper-mediator system (Amos II). The main idea is to make mediators within grid environments possible and to gain the advantages of both systems within the new implemented system. This chapter gives a short overview of the different topics that have to be considered when reading this thesis. First, the term of distributed systems should be explained. Within the last few years research teams used different terms and the system parts (e.g. processing logic and data) mentioned as distributed will be explained. After this, a short overview of *distributed databases* will be given, including transparencies and also the partitioning of the databases, as we will use the partitioning within our work, to show the mediation principles within Amos II. The last part of this chapter should give an overview of grid computing, involving also the term of grid computing. As OGSA-DAI focuses on the access to heterogeneous data resources over the grid, a short overview of data focused projects within the grid evolution will be given. But first we start with the topic of distributed systems.

2.1 Introduction to Distributed Systems

Distributed computing has many definitions and different components of computing can be distributed. Asking different researchers to explain the topic *distributed computing*, a number of different answers will be given. The reason for the differing understanding can be found within the next few paragraphs, as there will be defined, which components can be distributed. Distributed computing can on the one hand involve distributed processing within computer networks for the one research group, the next research group explains distributed systems as computing within multiprocessor systems (see [OV99]), and therefore different definitions can be given. Our definition can be found in [OV99] and defines a distributed computing system as:

> *A distributed computing system is a number of autonomous processing elements that are interconnected by a computer network and that cooperate in performing tasks.*

As mentioned above, distributed computing can be defined for different parts of the computing area. Within [OV99] the most important parts of possible distribution are:

- *Processing logic*: in most distributed systems we can ensure that the processing logic and the processing elements are distributed.
- *Function*: meaning delegating functions to different hard- and software.
- *Data*: different storage locations for data.
- *Control*: meaning the distribution of the control. When distributing the control it can be quite complicated to handle the consistence of different data.

The distributed computing systems can be classified by different criteria [OV99]:

- *Degree of coupling*: this means how near are the elements connected together. [OV99] distinguishes between two forms of coupling, first *weak coupling* and second *strong coupling*. An example for weak coupling can be handling the communication over a network. The term strong coupling means computing with the help of shared components.
- *Interconnection structure*: defines how the systems are connected together. The most commonly used connection within distributed systems is the *point to point interconnection*.
- *Interdependence components*: defines how dependent the different components are. [OV99] defined two different forms. There are on the one side strongly dependent components during the execution and also components where the interdependency is quite low, if we think for example of components where the communication is just done over message passing.
- *Synchronization*: Within distributed systems, two forms of synchronization can be found, named synchronous and asynchronous. Further information about distributed systems you can find in [OV99].

Using distributed systems corresponds better to the actual enterprise structure we have to deal with. Different companies are spreaded and therefore more distributed than ever before. Using centralized systems does not satisfy the wishes of the user any more. Distributed systems are more reliable and more responsive because we are able to duplicate and fragment databases and make sure that the replicated data is accessible by different users (see [OV99]). Using the Internet has become much more common in the last few years. At the beginning the Internet was just another part of large companies but nowadays nearly everybody has already done an online transaction. The use of distributed systems makes the *divide and conquer* rule much more easier. Splitting one large problem into a number of smaller ones and delegating the smaller problems to different parts of the system makes the problem easily solvable. The need of powerful computer systems lead to high costs of buying always the fastest processors or the latest hardware, if we think of splitting the large tasks into smaller ones, it is not necessary to think about highend computing products because it is possible to solve the smaller problems within older systems too. As we think of grid computing, a main point is to share the different resources within groups, so for example a computational resource that is not used to full capacity can be shared with other groups.

2.2 Distributed Databases

In the past data was stored in quite simple ways. One database has stored a huge amount of data and this database was maintained and handled with the help of just few persons. Also the storage of data within files was common, but now the change of company structure also implied a change of data usage and data storage. Distribution all over the world has become common and nearly every company is spreaded all over the world. The data is saved within the company, but different departments and persons have to take care of these resources. The knowledge of the different data and the handling of these data has arised to a main point within database management systems. Few years ago a database team consisted, depending on the company size, of 2-5 members, nowadays large companies have own departments full of database experts. The companies have to handle the data of different departments with care and also handle different problems that have to be solved. Different departments are lead by different persons and so also different databases can be used. Handling huge amount of data can also led to spreading the data to different servers and databases which could be geographically distributed.

The term *distributed database* is defined by [OV99] as:

> *'A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network.'*

Within the next few paragraphs a short overview of databases and the management of this data will be given. Of course this will be just a really short introduction as everybody knows, that about the topic databases a quite large number of books and other publications was written.

2.2.1 Distributed Databases and Heterogeneity

The very important term discussed here is *heterogeneity*, the differences among the databases. Different forms of heterogeneity will be dealt within the next few sections.

The most common databases used in the last few years can be mentioned as *relational DBMS, XML DBMS, object-oriented DBMS and file storage*. When we think about relational databases nearly everyone thinks of SQL[1]. The following part can be found in [Tut08] and should just give a really short explanation what SQL is. SQL is used to access databases and to formulate queries for these databases. Also we can formulate SQL-statements to retrieve necessary data from the database. Updating a database can also be done, using the *Update*-statement. Also the creation of new databases or database tables can be done with the help of *Create*-statements. Of course if creation is possible, also deleting is a function that is supported also known as *Delete*-statements. The common format for a SQL-query is the typical *select-from-where*-formula. This should just give a short explanation of the main functionality of SQL within relational databases, of course not all specialities are mentioned, because it is possible to write a dedicated thesis about relational databases and the possibility to write and extend SQL-statements. As within OGSA-DAI it is possible to access relational data sources, a lot of these statements can also be performed within OGSA-DAI, as there is for example an activity called *SQLQuery* and this activity is responsible for performing a SQL query against a relational data source within OGSA-DAI. But for sure it is to mention that the chapter on *Amos II* will also show queries similar to SQL queries, extended by the possibility to access distributed data

[1] Structured query language

resources.

But have non relational databases, for example XML databases, also the large XML family, found different solutions to handle the data and to find ways to access these data sources easily. One possible solution to access data within XML files is XQuery (see [XQu08]) also in combination with XPath (see [XPa08]). If we think of changing relational databases, update and insert statements are given, while in XML an own 'language' is provided to make changes and modifications of the XML files possible. The language using this possibility is called XUpdate (see [XUp08]). As mentioned above, OGSA-DAI allows to access different resources and therefore also the access to XML data resources is possible, so special activities similar to the activities for relational data sources are formulated and can be executed against a XML data source.

The data can also be stored in different files, also known as CSV-files[2]. This can be just comfortable for data that can be handled easily, but thinking of finding a special data set, it is much easier handling it with the help of SQL-statements for example, just inserting a SQL statement to look for special requirements makes an easy access of this data row possible.

Of course it is possible that the company uses just relational databases, but therefore also different commercial database products can also deliver different results and therefore heterogeneity. Relational DBMSs can for example be *Ingres*, *DB2*, *Oracle* and so on. The table 2.1 shows an overview of relational DBMSs and the vendors of the products. This overview is taken from [RDB08]:

RDBMS Vendors	RDBMS
Computer Associates	Ingres
IBM	DB2
INFORMIX Software	INFORMIX
Oracle Corporation	Oracle
Microsoft Corporation	MS Access
Microsoft Corporation	SQL Server
MySQL AB	MySQL
NCR Teradata	
PostgreSQL Dvlp Grp	PostgreSQL
Sybase	Sybase 11

Table 2.1: RDBMS Vendors [RDB08]

Thinking of XML databases the variety of different products is smaller. The most important XML databases can be mentioned as Apache Xindice and eXist. Xindice uses the above mentioned XPath and XUpdate. For further information to Apache Xindice please see [Xin08]. eXist is an open source XML database for the saving of XML files. It uses XQuery for query processing. For further information to eXist please see [eD08].

Of course it is possible to use the same commercial or open source products, but also different *schema* within the tables can be found. The next two tables can be used to save

[2] comma-separated values

the same data, but you can see that the structure is a bit different. The following three tables should show the possible differences:

- Table 2.2 containing a special schema
- Table 2.3 containing a similar but not identical schema
- Table 2.4 with a schema and the involved data

per_id	f_name	l_name	address	department	job_description
1	Barbara	Mayer	Firstroad 4	Database	Database developer
2	Monika	Miller	Secondroad 5	Datawarehouse	Datawarehouse expert
3	Peter	Smith	Thirdroad 6	Development	Senior Software Developer

Table 2.2: Example table *one* with a special schema

pid	name	address	dep	job
1	Barbara Mayer	Firstroad 4	DB	DB developer
2	Monika Miller	Secondroad 5	DW	DW expert
3	Peter Smith	Thirdroad 6	DV	Senior DV

Table 2.3: Example table *two* with a special schema

pid	name	address	dep	job
1	Barbara Mayer	Firstroad 4	1	Database developer
2	Monika Miller	Secondroad 5	2	Datawarehouse expert
3	Peter Smith	Thirdroad 6	3	Senior Software developer

Table 2.4: Example table *three* with a special schema

We can see different tables storing the same content on the first sight, but if we take a closer look to the table schema we can find a few differences that make the query processing quite difficult. Therefore a single SQL query is not sufficient to retrieve from all three data sources. An example query could be:

```
select pid from three;
```

This quite simple query is formulated to find out all *pid*s from *table three*.

The result for the above query with the above mentioned table three is the following:

```
1
2
3
```

For receiving the same results for the table one, we have to make a minor change in the query to get the correct results, because we have to deal with different attributes within the tables. The *pid* mentioned in table three is the *per_id* in table one:

```
select per_id from one;
```

A bit more complex and better to see is the difference between the *table one* and the *table three* when thinking of the names. While in table one the name is splitted into first name and last name, we have just one attribute where the name is saved in table three.

We can see, that the 'same' data can be stored differently and so a combination of the different databases and tables can be difficult because we have to handle heterogeneity within the tables, even if we have to think of the same relational database products. The different schema can need also a mediation system that is responsible for handling the differences and hiding it from the user and make the query processing possible. At the moment we need to do at least a mapping between the tables to hide the differences. As seen above a main point of heterogeneity can be the heterogeneity of *schema*. Our approach will show the handling of mediation principles of Amos II with the help of OGSA-DAI, meaning, accessing Amos II over OGSA-DAI.

2.2.2 Distributed Databases and Partitioning

Within [OV99] different forms of partitioning are mentioned. Here will follow a short overview of the possible partitioning possibilities.

Vertical Partitioning

Vertical Partitioning means that two or more data sources are joined over a key attribute. We can think of splitting one large table into more smaller tables. A possible table can be the one mentioned above

pid, name, address, dep, job

It can also be splitted into two tables containing on the one side the personal information in our example

name, address

and in the other table the company information

dep, job

This makes it also easier to handle the different access rights to the data, if we think of having potential partners, it can be possible that we just want to show them the departments and the job descriptions of the employees but not the names and addresses, for example.

The split of Table 2.5 into two other tables should show the vertical partitioning of the tables.

pid	**name**	**address**	**dep**	**job**

Table 2.5: Example table schema with employee data

Table 2.5 can be split into two different tables. On the one side a table containing the personal data (see Table 2.6) and on the other side a second table containing just the company information (see Table 2.7).

Very important and therefore mentioned again is that the tables should have a common key, so that we can have a combination point between the two tables and that we are

pid	name	address

Table 2.6: Example table schema with personal data

pid	dep	job

Table 2.7: Example table schema with company data

still possible to combine this two data sets into one data set again to make in addition queries over these two tables. SQL therefore has created a *join*-operation, responsible for combining two tables to solve queries. Within Amos II it is also possible to retrieve query results from different tables, even from different databases that can be stored in a distributed environment.

Horizontal Partitioning

To handle large data sources can sometimes be difficult and, therefore, it can be necessary to split the data and save it into different tables. The definition of splitting huge databases into smaller ones with the help of predicates can be defined as *horizontal partitioning* [OV99]. *Table one* can contain the data of the same type like *Table two* and *Table three*. The splitting is done by different predicates. It is possible that in *Table one* all employees are saved with a pid smaller than 200, in table two all employees with a pid between 200 and 400 and in table three all employees with a pid larger than 400. Now we have to split the one table into 3 different tables to prevent that the database grows into huge dimensions. If we need to think about creating query statements it is possible using the *union* possibility to combine the three different tables to make one query for the three tables. We can think of three tables with the following schema:

- table *employee one* with the following schema **pid, name, address, dep, job**
- table *employee two* with the following schema **pid, name, address, dep, job**
- table *employee three* with the following schema **pid, name, address, dep, job**

Combining this three tables to perform one query can be quite easy. In relational databases a common *Union*-Operator is given:

```
select name from employee one
UNION
select name form employee two
UNION
select name from employee three
```

The result of this query is a combination of three different tables. The name of all employees are listed but no duplicate entries are shown. If there is for example a *Barbara Mayer* in all three tables, *Barbara Mayer* is only listed once. If you want the result of really all data within the three tables also showing duplicates you need to use *Union All*.

Partitioning over heterogeneous data sources

In the section above we have explained the term of heterogeneity. Heterogeneous data can be data that are different in type, format or database (see [OV99]). Heterogeneous data have to be mapped (creation of a mapping schema) to handle the differences, meaning to hide the differences for the user as well as possible. Within Amos II the heterogeneities of the different parts can be hidden by using wrappers and defining *derived types*. For more information to the term *derived type* please read the chapter about Amos II.

2.2.3 Distributed Databases and Transparency

Transparent systems need to hide the differences from the user. The implementation details are not known for the user and it is not necessary for the user knowing the details. In distributed databases it can be necessary that the user has to take care of data replication or the redundancy of data. The duplication of data can be necessary for reasons of performance or reliability (see [OV99]). Transparency can be understood differently. On the one side it is necessary hiding information of heterogeneities from the user, on the other side, transparent often means too that the user knows all details. The different transparencies are described in [LN07] and [Dat04] and will be shortly mentioned within the following enumeration.

- Location transparency: means, that the user does not know where the information is located or saved. It is not necessary inserting name, IP or host. It can be possible that the location of the database changes, but this should not effect the query.
- Fragmentation transparency: it should not be visible for the user, that the data is fragmented and in which way the data is fragmented. Fragmentation is an own topic, see the section describing the *partitioning*.
- Replication transparency: the user is not able to see which sites are involved when performing a query. Therefore it is also possible that different nodes are replicated and the user should not be bothered if replicated nodes are involved in the query execution.
- Interface transparency: it is hidden from the user that the data sources can be accessed by different methods. In a distributed database system, the user writes a query in a defined language and if needed the query is transformed by the system. The user does not need to think of translating the query into another language to perform the query.
- Schema transparency: the user does not need knowledge about the involved schema that are needed to perform the query. The user sends a query knowing the global schema and the translation and mapping within this schema is done by the system without bothering the user.

One of our systems used within this thesis tries to provide all functionality mentioned above. Amos II tries to hide the differences and makes an easy and understandable access possible.

2.3 Introduction into Grid Computing

2.3.1 History of Grid Computing

Grid computing has received a lot of support in the last few years by different researchers. The main concept of grid computing includes the sharing of the different resources, maybe even over the Internet. The resources can be data, people or even computational resources (see [AOD]). Within the grid, the main focus lies on the sharing of resources, in the beginning such sharing was just topic of huge research groups, but in the last few years the different approaches have grown quite fast. The evolution goes in the direction middleware and handling of different data sources and sharing these data sources within virtual organizations (see [FKT01]). The main idea is to create different virtual organizations containing different research teams and to handle the different resources available; different projects focus on different points of interest. The evolution of the grid can be summarized within three different steps, involving different research teams and different projects implemented during the different phases. In [RBJ03] the different phases are named as *generations*.

[RBJ03] has named three different phases: *first generation*, *second generation* and *third generation*. Within the next few paragraphs a short overview of the different generations will be given. The different systems and research areas will be explained shortly.

First generation

During the first phase, two main projects can be mentioned. There is the project *FAFNER* and also the project *I-WAY*. Both projects are shortly described within [RBJ03]. Different topics need to be solved, for example the communication or the handling of remote data sources (see [RBJ03]). The main idea was not the shipping of different data within the grid, more it was necessary to create linkage possibilities for the supercomputing sites.

FAFNER stands for *Factoring via Network-Enabled Recursion*, that was introduced to enable the factoring of large prime numbers. As factoring is quite expensive, there is the need to handle such topics with the help of different participants. These different participants can handle the different parts of this effort.

The second project within the first generation can be mentioned as *I-WAY*; it stands for *Information Wide Area Year* and the main idea was the integration of different networks and to enable an visualization environment.

Second generation

At the beginning of the research of grid computing, the main focus was lying on the linkage of different supercomputing centers (see [RBJ03]), but for nowadays grid computing is not just reserved for a small research area. Within the second generation a lot of the now used grid applications were developed. The main issues are mentioned in [RBJ03] as:

- *Heterogeneity*: as mentioned before, different resources for example databases with different schema, but also different computational resources that may be distributed all over the world.

- *Scalability*: handling of the resources. A grid environment can start with one computer or one database and can expand to millions of resources. Of course if we think of geographically distributed resources we have to think about the handling of the latency. It is necessary if a grid system is growing, to think about the authentication and authorization, because different research teams may be involved and therefore not every person should have access to every resource available.
- *Adaptability*: here the main idea is the handling of missing resources or failed resources, because within large virtual organizations it is quite common that resources are missing, and therefore the resources needs to be handled dynamically.

Within the second generation different research projects were introduced. One project is called *Globus*, creating an infrastructure that makes the access of different heterogeneous resources possible. One main element was the *Globus Toolkit*, responsible for the definition of the basic services needed for a computational grid. Within this toolkit also a part is involved to make the data access possible called *GridFTP*. Open Service Architecture (OGSA) is a new architecture that is based on Web Services and Globus and is one main part of the system we used for our thesis, named OGSA-DAI.

The main focus within the description lies due to data issues and therefore not all developed projects will be presented in a quite extensive way. Just the projects involving explicitly data access over the grid will be explained in a more detailed way, the other projects will just be listed within a itemization.

Storage Resource Broker also shortly SRB was developed at San Diego Supercomputer Center (SDSC). SRB is a middleware that is build on the client-server concept. SRB uses a logical name space, this name space is responsible for the identification of the data. The main concept within SRB is the finding and accessing of data stored within files. The main idea is to retrieve meta data and also the easy access to this data. For further information please read [RWM].

The NPACI HotPage is a grid portal that should give easy access to different computer-based resources. Within this portal also the access of data on different platforms or from different applications can be possible (see [RBJ03]).

Grid Portal Development Kit is established by the Grid Portal Collaboration and this Collaboration is responsible for different components, that should make the development, as the name indicates easier. The core infrastructure given here is Globus and Grid Security Infrastructure (see [RBJ03]). The main focus within this portal could be for example the file transfer or database queries.

Data Grid was a project that is led by CERN. The main idea for Data Grid is the creation of an environment that is than responsible for the analysis of the data. Data Grid is quite data-intensive as the name indicates and different projects are involved within this research project (see [RBJ03]). Different projects can be mentioned, for example the *PPDG: The Particle Physics Data Grid* (see [ppd]) or *NEES: Network for Earthquake Engineering simulation*. A grid network with different experimental facilities to simulate earthquakes and therefore to distribute the information to different involved parts. The data is saved within the grid and can be accessed over this grid network. Of course there are more projects that use Data Grid, but this should just give a short overview of the

grid possibilities and does not contain all projects developed and introduced within the last few years.

Peer-to-Peer computing is a part where the main idea is to share data and other computational resources, for example storage or other free capacity. The most important peer-to-peer computing projects are mentioned within [RBJ03] as Napster, Gnutella, Freenet or JXTA.

The different projects not focused on data are Legion, Corba, Jini and RMI, Nimrond/G Resource Broker and Grace, Cactus, Unicore and WebFlow. For additional information to the above mentioned projects please see [RBJ03].

Third generation

The second generation of grid computing was quite full of new investigations, the new generation tries to reuse the implemented parts and resources and to assemble these resources to create new grid solutions. The new focus lies on the service-oriented architectures (see [RBJ03]). Service-oriented architectures includes web services and therefore also different standards were established, the standards can be named as SOAP, WSDL or UDDI.

Also part of the third generation is the *Open Grid Services Architecture* also named as OGSA. OGSA was introduced at a meeting of the Global Grid Forum. The main idea is to handle services with the help of Virtual Organizations. Within our work we will implement new activities with the help of OGSA-DAI, a framework that can be accessed using web services and can handle different activities over heterogeneous data sources (see [AOD]). For a more detailed introduction to OGSA-DAI please read Section 4.1 starting at page 38.

Chapter 3

Amos II

Within our implementation we will use OGSA-DAI to access Amos II and therefore allow mediation with the help of the Amos II functionality. The main focus of our work should be to support mediation within the grid solution of OGSA-DAI. For a better understanding of chapter 5 a quite detailed overview of Amos II is given.

This chapter has the following structure. First an introduction to Amos II will be presented, afterwards the architecture of Amos II focusing on the different implemented wrappers will be explained. The next section gives an overview of the implemented wrappers within Amos II. The functional data model, the query processing and the query decomposition are also part of this chapter.

A distributed mediator system named Amos II (Active Mediator Object System [RJK03]) was developed at the University of Uppsala in Sweden. The first implementation and design was shown within a thesis of Vanja Josifovski together with his supervisor Tore Risch. Amos II is a distributed mediator system and the heart of the system is a DBMS[1]. In [RJK03] the main aspect of Amos II is mentioned as

> *"The core of Amos II is an open, light-weight and extensible DBMS with a functional data model."*

3.1 Introduction

In the last few years it became more and more important to handle huge amount of data. These data are stored in large databases and sometimes there is not just one centralized database, but many distributed databases located all over the world. There are different database types (e.g. relational databases, XML databases or object-oriented databases), different views of data and of course different locations. The data integration part needs a unique view of the data, this can be performed with the help of the mediator/wrapper approach of Amos II. The functionality of the wrapper-mediator approach [RJ] in Amos II consists of two parts. On the one hand, there is the mediator and on the other hand, Amos II uses wrappers. Amos II is as mentioned above a distributed mediator system and so the system consists of at least one mediator, but also a larger number of mediators is possible and can be accessed.

[1] Database management system

The wrapper can be implemented for special type of data and should make the access of the data with the help of Amos II possible. The mediators can communicate over a protocol based on TCP/IP (transmission control protocol/internet protocol). The wrapper is responsible for the access of the data. The access of Amos II can be done within a CDM (common data model) and the help of the corresponding query language *AmosQL* (see [RJK03]). With the help of *AmosQL* over types the different data sources can be accessed. As mentioned above, different wrappers were implemented, for example for accessing XML-data.

As mentioned before, Amos II uses the mediator-wrapper approach, an approach evaluated by different research teams. The idea of a mediator/wrapper was mentioned in [Wie92]. In [Wie92] a mediator is defined as:

> *'A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of application.'*

In [WG97] a list of responsibilities of a mediator are described. The most important are mentioned in [WG97] and can be summarized within the next enumeration:

- Calling the necessary wrapper for the different data sources.
- Choosing the relevant data sources.
- Choosing the necessary information or meta-information and send it to the applications.
- Finding the optimal access possibilities to create queries with low costs or small response time.
- Integration of the different data sources and receiving the intermediate results for subqueries.
- Transformation of the data to give a well understood response.

The wrapper is responsible to access the different data sources and should hide the heterogeneities for the user; the mediator should make a uniform view of the data possible and therefore enable the distributed query processing within the Amos II system.

In [RJK03] a mediator, also named in our thesis as Amos II peer, is defined to be also a DBMS and having the functionality of a DBMS. The most important involved parts are *storage manager, recovery manager, transaction manager* and *query processor.*

As mentioned before, the Amos II system uses an own query language called AmosQL [RJK03]. The client sends a AmosQL query to the Amos II peer and within the peer it will be decided if and which other peers will be involved during the query processing. The explanation of this topic can be found few pages later. This query language is responsible for performing queries over mediated data within the Amos II system. The Amos II system starts as a stand-alone system, meaning that just one mediator is involved. Additional mediators can be added and therefore also new global schema can be defined. Different mediators can be grouped and therefore bottlenecks should be prevented. A bottleneck can occur when using just one mediator with a global schema where huge amount of data should be processed, therefore a main concept is to use more than one centralized mediator

and so smaller data groups can be queried.

The mediators are grouped together and the meta information is saved within a *name-server*. In Amos II the different mediators are named as *mediator peer* or within our thesis as *Amos II peer* (see [RJK03]). For the communication between the different mediator peers, different communication structures can be given. It may not be necessary that every mediator communicate with the other mediator, the communication structure can be defined and also changed if for example a new mediator is added or a mediator being part of the communication is removed.

If we think of distributed query processing we have also to consider that parts of the distributed system are not available and therefore it may be necessary to get replicated data from other resources if possible. These mediators are described by a meta schema. This meta schema is saved in the name server. The name server (see [RJK03] and [RJ]) does not have a central schema of the different mediators. The mediators are independent and there are just meta-information about the mediators. Such meta-information can for example be the location or name of the mediators. Every mediator is defined by an own schema for the data sources the mediator can access. For the communication between mediators it is not necessary to contact the name server, because the mediators communicate directly one to another. The name server is accessed, if new mediators are added or locations and names are changed. If the mediators would use the name server for every communication this would create performance bottlenecks [RJK03].

3.2 Architecture of Amos II

This section should give an overview of the architecture of Amos II. As Amos II can be configured by the user, it is possible to access Amos II over different other systems and can interact with other systems. In our case Amos II will be accessed over OGSA-DAI. For the client it is not directly necessary to know all details about the system that is behind the one Amos II peer that is accessed with the help of OGSA-DAI. Different Amos II peers can be involved and this is hidden from the user. The user just needs information to know where one Amos II peer is located and how this peer can be accessed. Amos II can be configured in two dimensions (see [RJ]). On the one side the accessibility for example single-user or embedded systems on the other side there is also the mediation dimension (see [RJ]). As mentioned before Amos II consists at least of one mediator, this is called a stand-alone system (see [RJ]). By adding a second mediator peer Amos II transforms into a mediator system. At the start, Amos II is a stand-alone system containing just one mediator, but with the help of different system commands, the Amos II peer can be a part of a federation or grouping of mediators. A stand-alone system is a Amos II system containing of one mediator, this is also the start point of a mediation system. In addition more mediators can be formulated and added if this is necessary. The need for more than one mediator is given in special conditions where more heterogeneous data resources are given and more data sources need to be added. Amos II delivers interfaces in Java, C and Lisp and makes it possible to develop applications in C and access mediators with the AmosQL language. A main part of this work will be the implementation of a data resource within OGSA-DAI and it should be possible to perform different AmosQL queries sending them from OGSA-DAI to Amos II and getting results and responses from the Amos II

peers. The implementation of the different activities will be shown in Chapter 5.

Amos II architecture consists of three levels. Figure 3.1 is taken from [JR02] and shows the different parts of the architecture.

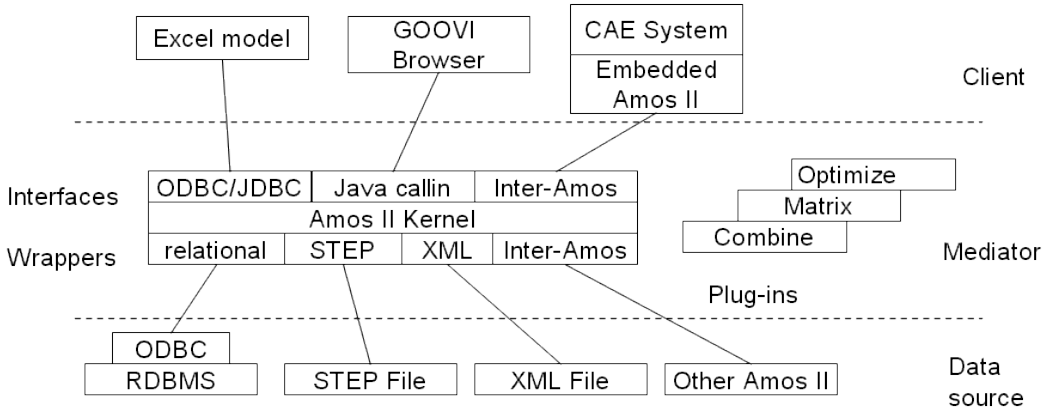

Figure 3.1: Architecture of Amos II [JR02]

The top level makes the access to Amos II possible and makes it also possible to access Amos II with the help of OGSA-DAI. The main idea is to make Amos II to a system that can be accessed by different programs and different interfaces. An example of accessing Amos II can be the java callin interface (see [ER]). The lowest level contains data sources and in between is the mediator consisting of interfaces for clients and wrappers for different data sources. As we mentioned earlier, every Amos II peer has DBMS functionality and this can be found within the Amos II kernel.
A wrapper is a program, that is able to translate data and therefore perform a query over different data sources. The different data resources can have own wrappers and it is possible to access the different data sources with the different wrappers. Possible wrapper can be responsible for translating relational databases and accessing them or for example translation and access of XML.

The wrappers need to provide the following functionality (see [RJ] and [RJK03]):

- *Schema importation*: As mentioned later, in Amos II the data model contains of three part, on the one hand *Objects*, defining all entities of the Amos II system, than *Types* and *Functions*. Every data source can be different as mentioned in Chapter 2. Database schema can be different, also different kind of databases can be involved. For the topic of schema importation a main point are the different parts of the functional data model that needs to be imported, for example the types, functions and objects. The explanation of the functional data model will follow in Section 3.3.

- *Query translation*: Every Amos query is translated into a calculus representation. This representation needs to be for itself translated into a query part that can be understood by the involved data sources. A AmosQL query can not be performed

against a XML database without translation of the AmosQL query into a query that can be understood by this database.

- *OID (object identifier) generation*: if information from external sources is needed, an OID is generated to grant an execution of commands in the source and a unique identification of the data. When using for example the *import_table* function within Amos II, this function translates the table into types, objects and functions and allows therefore the mediation of the data resource. A mediation example can be found in Section 6.2.

For the different applications there are also different interfaces necessary, depending on the form of access that is planned. Figure 3.1 shows three different interfaces, first for ODBC/JDBC, the Java call-in and the inter-amos interface. It is possible to access Amos II over GOOVI. Information concering GOOVI are taken from [CR01].

GOOVI stands for graphical object-oriented view integrator

and is an graphical user interface and is responsible for managing the different mediators in Amos II. The first GOOVI user is also the mediator administrator and the administrator defines and modifies the mediators needed in the browser. Within our work the use of GOOVI was not planned, so also no deeper information to the topic of GOOVI can be given within this thesis. If further information is needed concerning the topic *GOOVI* please read [CR01]. There are also ODBC- and JDBC-based interfaces and these interfaces allow Amos II to communicate with these standards. In these interfaces it is necessary to transform SQL into AmosQL because in the cases of OBDC and JDBC the system does not use object-oriented relational models. The third possibility is to use applications which are embedded with Amos II (for example computer aided engineering systems). There is an inter-Amos interface and this interface can perform the communication between different mediators.

The architecture of Amos II is extensively described within [JR02]. Within the kernel also extensions are possible. Different add-ons or plug-ins can be defined, to ensure better performance. The three plug-ins, that can be found in Figure 3.1, are *Optimize*, *Matrix* and *Combine*. The different plug-ins can also include new algorithms. The performance can be improved with the help of optimization, representation of data or the fusion. The plug-in optimize is responsible for the optimization of the data, this means also that the data is not redundant or too much information are stored in one data source. With the plug-in matrix is possible to formulate the data into a representation known as matrix. The plug-in combination is also important because sometimes it can be useful and necessary if more than one data source is available. Combining the different data sources to one data source may be necessary. We can see that different wrappers are within Figure 3.1 shown, but also in the last few years new wrappers have been implemented. A overview of the different wrappers implemented can be found on the next few pages. The list should give just an short overview and a even shorter description just showing the main functionality of the implemented wrappers.

3.2.1 Wrappers within Amos II

Different wrappers were implemented and so different data sources can be queried and accessed over Amos II. The following list should show the most important wrappers, who was responsible for the research and what was developed. The list may not involve all

designed wrapper, it should just show what topics are considered and maybe give some ideas for new wrappers. An idea of a new wrapper could for example be the integration of OGSA-DAI within Amos II, meaning that a wrapper is created, that the different data resources within OGSA-DAI can be accessed by Amos II.

The overview of the different implemented wrappers can be found in [amo]. We have tried to structure the classification of the different wrappers and have defined the following:

- Wrappers for *Internet resources*, for example search engines or also Internet forms. The wrappers for the Internet data can be found in Table 3.1 on page 21.
- Wrappers for *scientific databases*. The wrappers for ROOT-Files and B-Tree storage manager can be found in Table 3.3 on page 22.
- Wrappers for formats that are usually not saved within databases for example *music files* or *pictures* can be found in Table 3.2 on page 22. Within this wrappers the meta data of pictures and music files are wrapped to be accessible by the Amos II peer.
- Wrappers for accessing different data formats, for example XML and also accessing multi-database systems with the help of ODBC. The overview of these wrappers can be found in Table 3.4 on page 23.

The different parts will be shown in different tables and within these tables also a short description will be mentioned, also the full title of the document and the responsible authors and if known the year of the research.

Author	**Title**	**Description**	**Year**
J. Petrini	Accessing web forms from an object-relational database system [Pet01]	ORWIF: Object relational Wrapper of Internet Forms, Wrapper for receiving information from different web sites, foreign function calls for Amazon, Ginza and XE	2001
T. Katchaounov, T. Risch, S. Zürcher	Object-oriented mediator queries to Internet search engines [KRZ02]	ORWISE: a object-relational wrapper of Internet search engines, using existing wrapper toolkits to receive the information from the webpages, foreign function calls for example for AltaVista, Google or Cora	2002
T. Risch	Functional Queries to Wrapped Educational Semantic Web Meta-Data [Ris03]	RDFAmos, functional mediation over RDF meta-data, Ednutella	2003

Table 3.1: Summary of Internet wrappers

As mentioned above it is possible that not all implemented wrappers are mentioned, this overview should just give ideas what data can be wrapped and accessed with the help of Amos II. We can think of new wrappers, that make the access of different OGSA-DAI resources or activities possible. The implementation of wrappers are as seen in the different tables usually done with the help of *foreign functions* (see [RJK03]).

Author	Title	Description	Year
V. Petrauskas	Object-relational wrapping of music files [Pet05]	Wrapper for music files, query of different music formats: MP3 and Ogg Varbis, but no reconcilation between the different formats, foreign functions	2005
M. Jost	A wrapper for MIDI files from an object-relational mediator system [Jos]	Wrapper for Midi-Files with the help of foreign functions and making queries over such files, also loading midi-files into the Amos II database.	-
J. Elmiger	An object-relational meta-data manager for Picture Files [Elm05]	Wrapper for picture files, pictures taken from digital cameras in jpeg format, view of jpeg and saving of the meta-data of the pictures, loading of the pictures and refreshing of meta-data, Exif standard	2005

Table 3.2: Summary of music-file and picture wrappers

Author	Title	Description	Year
J. Tysklind	Wrapping a Scientific Data Management System [Tys05]	Wrapper for ROOT-Files for a library at the Cern Laboratory, different calculations for particle physics	2005
M. Ladjvardi	Storage Manager: Wrapping a B-Tree Storage Manager in an Object-relational Mediator system [Lad05]	Wrapper for the Berkeley DB, a wrapper for b-tree storage	2005

Table 3.3: Summary of scientific data wrappers

3.3 Functional data model of Amos II

Amos II works with the help of a functional data model and this model is extended with the help of object-oriented expressions. In [RJK03] it is defined that the functional data model is based on the model of Daplex. If further information is needed concerning Daplex, an explanation can be found in [Shi81]. In [RJK03], [RJ] the main concepts of the data model are described as objects, types and functions. In the next subsections short explanations of the different concepts will be presented.

3.3.1 Objects

The main concept in Amos II is the term of *objects*. The data sources within Amos II are objects. The objects can be defined in two ways, first the user can enter objects and also the system may have created objects (see [RJK03]). A possible system-designed object can be for example string. The different objects can be distinguished by their *types*. In [RJK03] the objects are distinguished in two different forms, there are literals and surrogates.
An example for surrogate can be a real-world entity, for example, person (see [RJ]). A

Author	Title	Description	Year
S. Brandani	Multi-database access from Amos II using ODBC [Bra98]	accessing different data sources of different databases, thinking of heterogeneities and using for this part ODBC	1998
H. Lin, T. Risch, T. Katchaounov	Adaptive Data Mediation over XML Data [LRK01]	DOM-Parser for XML-Files, parsing and quering of XML documents, Amos II XML-Wrapper	2001
C. Rodunger	Accessing XML data from an object-relational mediator database [Rod02]	builder module for parsing XML documents using SAX, storing XML data in Amos, integration of XML files in Amos II	2002
T. Johansson, R. Heggbrenna	Importing XML Schema into an Object-oriented database mediator system [JH03]	Importing of XML Files, Import tool for XML schema: Amos II XML Schema import tool (AXSI)	2003
L. Scheuring	Loading XML Schema-based data sources into an object-relational database system [Sch04]	Loader for XML schema	2004

Table 3.4: Summary of XML and ODBC wrappers

surrogate has an own OID[2], this OID the object receives from the system. The system is also responsible for the maintenance of these OIDs. Surrogates are objects with an OID created and destroyed by the user or the system. Literals are objects without explicit OIDs and are self-described and maintained through the system. Examples are numbers or strings. The literals can also be collections. In [RJ] the collections are defined in two different ways. First there are bags. A bag is a unordered collection with different duplicates. Second there is the term of *vector*. A vector is a ordered list of objects. It is possible that in the vector also duplicates are saved. If we think of surrogates we have also to think about deleting the objects when not needed any more, literals are from the system created objects and therefore it is not necessary deleting this objects, because Amos II uses a garbage collector responsible for the deleting of objects that are not referenced to a database. An object is than deeper specified by the type. It is possible that the object has one or more types. The object than is an instance of this type (see [RJ]). The type is used to make the different objects to an instance of this type. The topic of types will be shown within the next lines.

3.3.2 Types

In [RJ] different kinds of types are mentioned:

- *Stored types*
- *Derived types*

[2]Object identifier

- *Proxy types*

The explanation of the above mentioned types will follow in the next sections.
The types can be organized in a supertype/subtype hierarchy [RJK03]. Within the supertype/subtype hierarchy we can find multiple inheritance, meaning if one object is an instance of a type, it is also instance of all supertypes. There are different types and here will follow a short overview of the different involved types. One part of our work is finding out the different types within an Amos II peer. Amos II distinguish between two different Amos II types, on there are types created by the system and also types created by the user itself. Different examples can be found in [RJK03] and [RJ]. An example may be *Car* is a subtype of *Vehicle*. In the example it is clear, that the supertype is *Vehicle*. Also it is possible to have a new type *AB* and this could be a new *Vehicle*. For sure is, that this type is just an example because in the real world such a vehicle does not exist.

Stored types can be realized with the statement *create type*. Here now a small code example with a following explanation for better understanding. As mentioned above, we have the types *Car, Vehicle, Bus* and *AB*. If we use the following code within the Amos II environment we will create some userdefined types, in our example *Vehicle, Car, Bus and AB*. An similar example can be found in [RJK03].

```
create type Vehicle;
create type Car under Vehicle;
create type Bus under Vehicle;
create type AB under Car, Bus;
```

In the type hierarchy there are different new types now created with the help of the statements above. First there is a root of the hierarchy called *Object*. An object can be a literal, a userobject, a type, a function and data sources. The small Figure 3.2 shows the hierarchy of the system types and can be found in [RJK03].

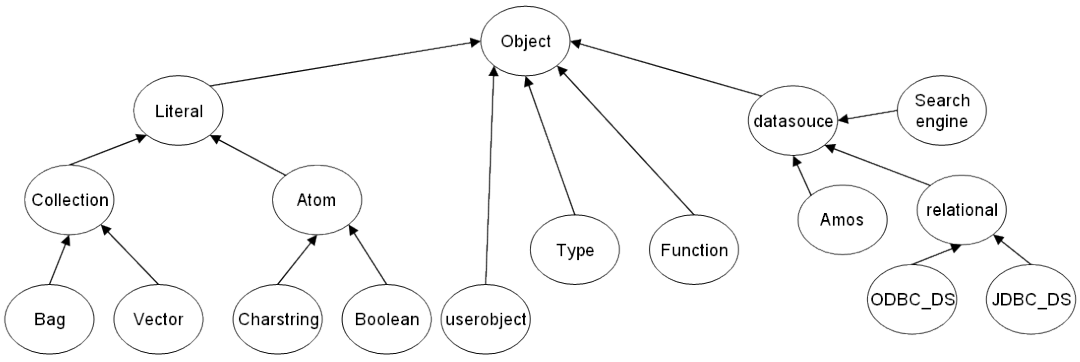

Figure 3.2: System type hierarchy [RJK03]

As mentioned there are user-objects created and defined by the user. In our examples there are new types created with the names *Vehicle, Car, Bus* and *AB*. The type AB can be a Car and a Bus. The extent of the type Vehicle are all objects of the types Vehicle, Car, Bus and AB.

We can distinguish stored and derived types. A stored type is as mentioned before a type that is saved within the mediator peer (also named as Amos II peer). A derived type is a type that is formulated for mediation reasons and can have a connection to a stored type within another Amos II peer. For data integration from other data sources it is necessary to think about derived types[3], the in Figure 3.1 shown system is extended by these types. It is necessary to think about derived types to make data integration possible and therefore a hierarchy is build with the help of local types and imported types from other mediators. Within derived types also the system type hierarchy is given and also the subtype/supertype concept is realized, especially for the reasons of ensure multiple inheritance.

The generation of derived types and the generation of *Object Identifiers* is defined within [JR99] and shows what approaches have been investigated within the topic of generating OIDs for derived types:

- using the OIDs from the supertype objects.
- using stored query expressions and with the help of these expressions formulating and defining the derived types needed.
- using new own OIDs for the derived types.

In the next section a short explanation of the three above points will be given. The first concept is quite interesting, but if we use the OIDs from the supertypes it is hardly to manage the multiple inheritance. The second part means transforming and analyzing the stored query expression and with the help of the query expression creating derived types. This form of implementation may be difficult because we have to handle stored query expressions and these expressions can not be treated like database objects. The third part is the easiest one, meaning for every derived type it is possible to create new OIDs. When creating new OIDs for the derived types it is necessary that it is possible to map the new OIDs of the derived types, with the old mentioned OIDs.

Derived types can be part of local functions and therefore used in queries, especially the OIDs can be used after generation. OIDs needs to be unique and therefore also the mapping for OIDs of derived types to the given types should be done. If we think of the correct given OIDs, it is possible that the system is performing run-time checks to ensure the validity of the derived types. When checking the validation of the derived types it is necessary to access also the given data sources to ensure that the OIDs are correctly and have corresponding attributes within the tables. Every derived type needs a corresponding type and this type can be saved within one or more data sources.

As the topic of derived types is quite intense please read the papers [JKR99] and [JR99] for further information. The next important point is the topic of the functions within the functional data model of Amos II.

[3] DT

3.3.3 Functions

Functions are necessary for the modeling of the meaning of the objects. Functions can show relationships between different objects, can set attributes of objects and can also describe methods of the objects. Functions are in Figure 3.2 on page 24 instances of the system type *Function*. A function contains of two different parts. In [RJK03], the parts are named as *signature* and *implementation*. The signature gives information about types, names, arguments and results of functions. For example, if we model the attribute *description* of the type *Car* or the attribute *color* of the type *car*, the signature will look like (see [RJK03]):

```
description(Car)->Charstring
color(Car)->Charstring
```

On the other side there are implementations. The implementation gives information about the handling of the functions. In our example means *description* to enter the database and find out the *description* of the *car*. Same interpretation can be found for the *color*. In Amos II an known approach is the usage of multi-directional functions (see [RJK03]). One example can be mentioned as finding out the *color* of a *car* with a specific name (Ferrari Modena), without iteration over the complete type *Vehicle*.

```
select color(v) from Vehicle v where description(v)='Ferrari Modena';
```

The basic functions can be classified into stored, derived, foreign, proxy functions and database procedures [RJK03], [RJ] and [JR02].

- Stored functions can be defined as the attributes of the objects. Stored functions can be found in the database and can be defined as for example a table within the relational database.
- Derived functions are functions that carry out queries over other functions. In Amos II there is a select-statement, this is a statement similar to SQL-select statement and with this statement it is possible to formulate derived functions and ad-hoc queries.
- Foreign functions are responsible for the possibility to give interfaces to external systems. Foreign functions can be used allow access to different external data resources. The overview shows for example a wrapper for Internet search engines and therefore Amos II is extended with the help of foreign functions. For further information of wrappers using foreign functions please read Section 3.2.1.
- A database procedure is a function in Amos II. A database procedure is described with the help of a procedural sub-language of AmosQL (see [RJK03]). Every procedure can have side-effects meaning that if the database procedure is inserted it can effect other databases and the queries too. As we tried to implement our activities in the way that Amos II peers are not effected and changed for other users, the usage of a database procedure is declined for our activities.
- Proxy functions will be explained in a later section within the topic of performing queries, because proxy functions, objects and types are needed to handle multi-database queries.

Functions in Amos II can be overloaded. Overloading means that different implementations of one function can be given. Within [RJK03], the different implementations are mentioned and named as *resolvents*. The most important point where the implementations

can be distinguished is because of the different arguments within the functions. Again a short description with the help of code samples for a better understanding. A similar example explaining the overloading of the functions can be found in [RJK03]:

```
create function description(Vehicle)->Charstring as stored
create function color(Vehicle)->Charstring as stored
create function description(Outlet)->Charstring as stored
```

In this example the function *description* is overloaded through *Vehicle* and *Outlet*. In [RJK03] Amos II functions are compared with relationships and attributes within the ER-Model[4]. The types are similar to the entities, the difference between functions within Amos II and relationships within ER-diagrams is the direction, meaning that within ER-diagrams the direction is neutral (see [RJK03]).

3.3.4 Proxy objects

If we consider two mediators and we want to exchange objects or meta-data between the mediators or data sources we need special objects, types and functions. To make distributed query processing and in this term also multi-database queries possible it is necessary to implement new objects, types and functions. These parts are described in [RJK03] as:

- *Proxy objects*: are objects that have corresponding objects within other mediators or data sources, the OIDs of these objects are saved within the mediator the query is done against. If we think for example of two data sources, data source one and data source two it is possible to save the objects of data source one additional to the objects of data sources two and combine these two data sources.
- *Proxy types*: are types which describes data from other mediators or data resources. The proxy objects have their own proxy types. These types are responsible for the description of the types form the data saved in other mediators or data sources.
- *Proxy functions* are functions that are stored in other data sources or mediators and describes these functions in the mediator the query is formulated against.

3.4 Queries and query processing in Amos II

This next section should explain how queries are performed within Amos II. Therefore first a introduction into the queries will be given, than a summary about the query processing process and also the distinguishing between queries against a single data resource and queries against a multi-database system, therefore the classical query processing is extended with a multi-query part.

3.4.1 Queries

This section should show, how Amos II handles queries. First a short introduction will be shown explaining how queries look like and what needs to be considered when creating queries and the corresponding results to the queries. Amos II uses as mentioned before AmosQL and a AmosQL select statement has usually the following format described in [RJK03]:

[4] entity-relationship model

```
select <result>
from <type extents>
where <condition>
```

In the section before we created some types with the help of the *create type statement*, a possible query for the local types can be (see [RJK03]):

```
select description(v), color(v)
from Vehicle v
where color(v)='blue';
```

In the query the result will be a tuple of *description* and *color* of all *vehicles* in the database where the *color* is set to *blue*.

3.4.2 Query processing

After defining a AmosQL query, the next step is to execute this query. Different steps needs to be done during the query processing. First just the query processing for local query processing was defined by [Kos00] and [LHP89] and the following phases are mentioned:

- *Query parser*: is responsible for the semantic checking and parsing. A query needs to be transformed into a calculus representation that can be understood by the system.
- *Query rewrite*: means reformulating the query. Joins may be removed when not needed, view of data can be changed and also predicate moving from one operation to another. This part can be extended in distributed databases also by moving one predicate from one site to another if possible. The query sometimes need also to be transformed. This transformation is done during the query rewrite.
- *Query optimization*: the main optimization process involves cost estimations and therefore it is necessary to know and estimate the costs needed for every step of the query.
- *Plan refinement and code generation*: the query execution plan within the before mentioned steps needs to be transformed into a plan that can be understood by the query evaluation system.
- *Query evaluation*: the query is now in a format that can be performed against a database. This step is done during the query evaluation.

A similar process for query execution is also given within Amos II, but this process is extended by the possibility to execute also queries against multiple databases. The overview of the query processing can be found in Figure 3.3. The query processor within Amos II has three main components (see [RJK03]):

- Local query compiler
- Multi-database query compiler
- QEP-Interpreter

Depending on the different queries, it is possible that:

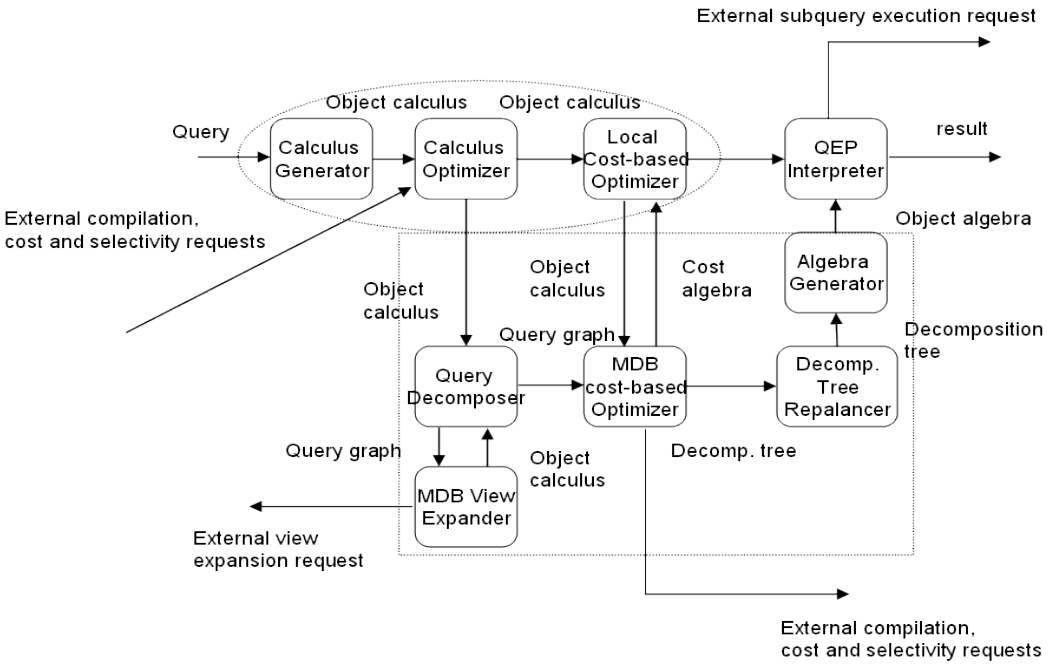

Figure 3.3: Query processing in Amos II [RJK03]

- just the local query compiler with the QEP is used or
- the multi-database query compiler with the QEP is used when formulating queries against more than one involved data source.

Within the next sections the different parts of the query processing within Amos II will be explained.

Local query processing

The first component is the local query compiler. The local query processing in Amos II means always, accessing just one data source and this topic is usually performed by the first part mentioned above *local query compiler*. In the overview Figure 3.3 the local query compiler is shown with three different parts (see [RJK03]):

- calculus generator
- calculus optimizer
- local cost-based optimizer

The next paragraph is a summary from [RJK03] explaining the query processing within Amos II. The calculus generator is responsible for transforming the incoming query into an object calculus expression. The calculus optimizer is responsible for removing and moving predicates from one place to another and removing parts of the query in the calculus expression that are not needed. The local cost-based optimizer is responsible for the cost statistics that are given. The three steps of the local query compiler are responsible for the the queries formulated for a local mediator, where just one data source is given to be used.

The local cost-based optimizer is as the name indicates responsible for finding different algebra plans and deciding with the help of different cost statistics for the most convenient. Of course it is possible due to missing or wrong statistics that wrong cost-based decisions are made. To minimize such problems, different research teams had investigated different forms of adapting this queries and the evaluation of these costs. In the literature different terms are mentioned and can be summarized above the topic *Adaptive Query processing*. For further information about the topic *Adaptive query processing* different overview papers are given, see [BB05], [GPFS] and [GPSF04].

In [JR99] and [RJK03] a query processing example is given and this example will be explained shortly also to understand how query processing is working in Amos II. First we have to formulate the types and the functions [JR99]:

```
create type person;
create function hobby(person) -> bag of string as stored;
create function name(person) -> string as stored;
create function parent(person) -> bag of person as stored;
```

Through these constructs mentioned above the following parts are constructed:

- The first statement creates a type with the name person. The other three create statements are responsible for the definition of stored functions over the type person.
- The function hobby gives a bag of character strings back.
- The function name returns a character string.
- The function parent returns a bag of person objects.

Now a new derived function can be defined, where we can find out the names of the children of a person with the hobby 'sailing' [JR99]:

```
create function sailing_children(person p)-> string as
  select name(c)
    from person c
    where parent(c)=p
      and hobby(c)='sailing';
```

The optimizer is responsible for the optimization and finding the execution plans. Ad hoc queries in AmosQL are treated as functions without arguments. An example for an ad-hoc query can be found in [JR99]:

```
select p, name(parent(p))
  from person p
  where hobby(p)='sailing';
```

Amos II is able to generate new functions for example with the name query(). The function query() is as followed described [JR99]:

```
create function query()-><person, string>
  as select p, name(parent(p))
    from person p
    where hobby(p)='sailing';
```

The query processing follows the steps in the figure. The AmosQL query is translated into a calculus representation. This calculus representation is known as ObjectLog. The calculus expression of the above query is the following [JR99]:

$\{p, nm|$
$p = Person_{nil->person}() \wedge pa = parent_{person->person}(p) \wedge$
$nm = name_{person->string}(pa) \wedge$
$'sailing' = hobby_{person->string}(p)\}$

The following explanation can be found in [JR99]. As we seen above the AmosQL query is just transformed into a calculus expression. The system generates additional variables if necessary. In this example the generated variables are *pa* and *nm*. We can see that the parent needs to be of the type person, while the name is defined to be a string and the hobby is also defined as a string and added to be part of the person. It is necessary to make it possible to find corresponding data. The lines defines also which types should the information have. Overloading of functions is possible and therefore it is necessary to look deeper to the arguments given in the functions. As mentioned above the optimizer is responsible for the reformulating the query. It is necessary to remove non relevant predicates or to reduce redundant predicates. The query optimizer is responsible for as mentioned above reduction of the predicates. In this example it is not necessary to make a type check, because the type that is checked is stored in a local data sources and therefore it is not necessary to check the type. If the function is not a stored one, may be a proxy function it is of course necessary to make a type check to ensure the right return value. The new calculus expression after the step of the calculus optimization will look like [JR99]:

$\{p, nm|$
$pa = parent_{person->person}(p) \wedge$
$nm = name_{person->string}(pa) \wedge$
$'sailing' = hobby_{person->string}(p)\}$

The last step in the query processing is the query execution plan interpreter. This part of the query processing system within Amos II is responsible for the execution of the in the steps before formulated execution plans. This plans are saved as object calculus and needs to be transformed into object algebra that than can be understood by the query execution plan interpreter.

Multi-database query processing

The main component for multi-database query processing is the multi-database query compiler. This compiler is always invoked when a query should be performed and the results of the query are delivered by at least two different data sources. For performing queries against more than one data source, a new term within Amos II is explained. When performing a query against more than one data source, it is essential to think about *derived types*. For the data integration different derived types needs to be created. Such creation depends also on the types within the data source, meaning that derived types can be formulated with the help of local stored types and also imported types from other mediators. As we can imagine it is more complex to do queries against derived types, because it is necessary to think about different parts (see [JR99]):

- Derived types needs to be formulated with the help of different local stored functions.

- As every object has an unique identifier it is necessary to think about the generation of OIDs (object identifiers).
- Creation of derived types meaning also to think about the consistency of the queries.
- Whenever it is possible try to use OIDs stored within local functions instead of generation derived types.
- When a derived type is used, it is necessary to think about the validation of the introduced types.

Whenever access of different data sources is necessary, derived types needs to be created. When creating a derived type it is mentioned, where the used type for creating this type is saved. As example we can see in the section about multi-database query processing, the different mediators needs to have derived types that the different queries can be performed against.

The really quite interesting point is to find different query execution plans, these plans have to be split into parts that can be performed by the different mediators. It is necessary to find a 'cheap' plan and to do the query at the level of this plan. Multi-database queries needs a query decomposer. A query decomposer is responsible for the decomposition of the query. The other important components during multi-database query are *MDB View Expander*, *MDB Cost-based Optimizer*, *Decomposition Tree Rebalancer* and the *Algebra Generator*. The different parts of the query processing within Amos II are described within [RJK03]. The next topic will just show a short explanation of all involved components. As mentioned above the different components are [RJK03]:

- *Query decomposer*: The query decomposer is responsible for the translating of the query and splitting the query into different subqueries. The query decomposition is performed within four phases:
  - *Predicate grouping*: deciding how the different predicates can be combined and sent to the different Amos II peers. The most important terms within this phase is the definition of the functions. These functions can be single implementation functions and multiple implementation functions. For further information on *Predicate grouping* please see [JR02].
  - *Predicate placement*: deciding where the different parts of the query (subqueries) will be placed. Different terms are mentioned for example *'Singleton site sets intersection'*. For further information to the topic of predicate placement including also the term of query fragment please see [JR02].
  - *Cost-based scheduling*: after defining the grouping of the functions and the placement of the sub-queries, the cost-estimation will follow and the decomposer will decide which execution plan will be performed. This parts can of course be extended, but as we do not need the detailed description for our implementation for further information please see [JR02].
  - *Decomposition tree distribution*: can be summarized as the transformation of the given decomposition tree into different decomposition trees and these trees are than handed over to the different mediators. For deeper information to the topic of the decomposition tree distribution please read [JR02].

- *Multi-database view expander*: The different parts of the queries can be extended by the views. The views can be referenced with the queries. During this phase it is possible that the query plan can be updated and improved and therefore redundancies can be removed. The multi-database view expander is responsible to send requests to the subqueries. It is necessary to combine the different predicates of one mediator to one group and therefore sending a group of predicates and receiving one answer. It is much more expensive to send the predicates one by one to the data source, it is much more cheaper to do a grouping and send the one request and getting one result, than performing for example 5 requests and getting 5 responses that needs to be combined at the end. Amos II uses heuristic expansion. It is not possible to really provide all possible views and so just the most promising views are chosen. The view expansion is done for all subqueries and after the expansion the query decomposer is again responsible for predicate regrouping. The summary of the different parts can be found in [JR02] and [RJK03].

- *Multi-database (MDB) query optimizer*: The multi-database query optimizer is responsible for the execution order of the predicates in the query graph. The multi-database query optimizer is responsible for choosing the direction of the data that needs to be shipped. The execution plans are described as decomposition trees.

- *Decomposition tree rebalancer*: This unit is responsible for the transformation of the decomposition tree. It is necessary to combine nodes to one. It is necessary to transform a left-deep tree into a bushy one. For further information on this topic please read [JR02].

- *Object algebra generator*: The decomposition tree needs to be changed into an other format. Usually the decomposition tree needs to be transformed into a object algebra plan to make it possible that the QEP Interpreter can handle this given plans for the distributed and multi-database queries.

An example of distributed query processing will be shown within the next few paragraphs. This example is taken from [RJK03] and will be shortly explained. Within this example a multi-database query is formulated. The system within this example contains of 3 different mediators, two of them access directly two different relational databases. Figure 3.4 is taken from [RJK03] and will be explained within the next few paragraphs.

Here will now follow a shortly description of Figure 3.4. We can see in the figure three different mediators. The mediators are differently named: The mediator on the top is the mediator receiving all queries from the client. This mediator is within this example named *M*. The two mediators under the mediator *M* are named within this figure as mediator *Ta* and *Tb*. The access of the mediators to the databases can be performed with the help of *odbc* or *jdbc*.

For accessing the data resources that are connected to another mediator it is possible to formulate the following query [RJK03]:

```
select name(p) from Personnel@Tb p;
```

This query can be interpreted as: *Please find out all names that are saved within the relational database of University B in the table Personnel.* As mentioned earlier, to access data resources that are usually accessed by other mediators it is necessary to create a proxy type. In our example the proxy type is *Personnel@Tb*, also proxy objects are necessary

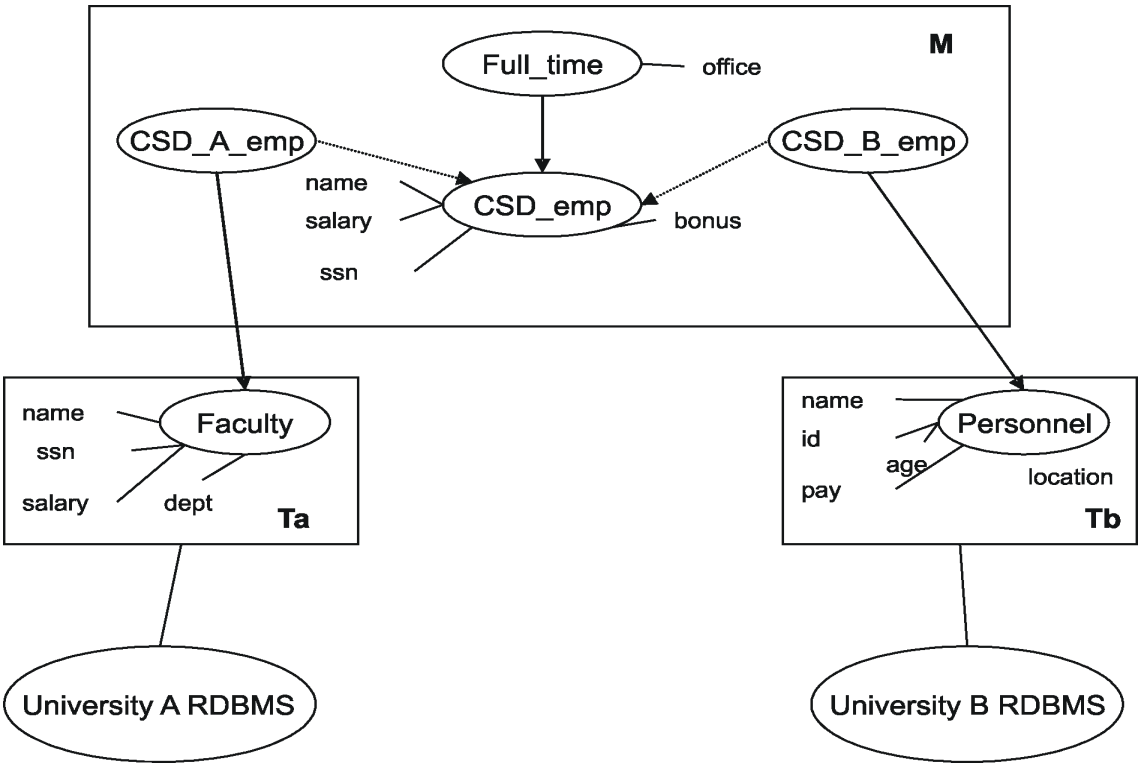

Figure 3.4: Mediation example with three mediators and two databases [RJK03]

for example the object *name* is a proxy object, because the object is not saved within mediator *M*. If the user for example accesses the mediator named *Tb* the creation of a proxy type is not necessary. A similar query can be also

```
select name(p) from Faculty@Ta p;
```

This query can be seen as a counterpart of the first query, meaning that the user wants to find out all names that are saved within the relational database of University A in the table Faculty. The query looks for us such simple, but if you think of the query decomposition, the complexity of the query is quite intensive.

Within the figure in the Mediator called *M* different derived types are created. In our case four different:

- *CSD_A_emp*
- *CSD_B_emp* and
- *CSD_emp*
- *Full_time*

We can see that the content of the different databases are similar, but the schema are not identical. On the one hand we have the table *Faculty* containing the following columns *name, salary, ssn and dept* and second the table *Personnel* containing the columns *name, id, pay, age and location.* As we have explained in Chapter 2, databases can be heterogeneous because of different reasons. Here the differences are quite obvious and easy to see. A possible derived type could be the following (see [RJK03]):

```
create derived type Emp
  under Faculty@Ta f, Personnel@Tb p
  where ssn(f)=id_to_ssn(id(p))
```

This new type is as the name indicates a derived one and contains all members that are saved within the two databases, where the *id* and the *ssn* have the same content. A similar example will be shown in our Section 6.2 using OGSA-DAI to perform queries against mediated schema.

As we do not have all entries within both databases also mapping needs to be done. The *where-part* of the AmosQL query should ensure, that just the rows of the two different tables are considered where the *id* and *ssn* are identical. The part *id_to_ssn* is a foreign function that should perform the mapping of these two columns.

The other derived types could for example be defined as [RJK03]:

```
CREATE derived type CSD_A_emp
  UNDER Faculty@Ta f
  WHERE dept(f)='CSD';
```

The derived type *CSD_A_emp* is a type that is taken from mediator *Ta* and just the members part of the department *CSD* are considered within this type. When creating such a derived type the system performs a lot of more steps, for example the importation of the external types and also the consideration of the functions that are defined over the types. Also implicitly proxy types are created and functions without implementation are considered.

A quite similar example could be [RJK03]:

```
CREATE derived type CSD_B_emp
  UNDER Faculty@Tb p
  WHERE location(p)='Building G';
```

The derived type *CSD_B_emp* contains here all data from mediator *Tb*, but the restriction is here also given including, all members of the university that are located within the *location* called *'Building G'*.

If we think for example of employees that are working in both universities it could be possible to formulate the following new type [RJK03]:

```
CREATE INTEGRATION TYPE CSD_emp
  KEYS ssn Integer;
  SUPERTYPE OF
    CSD_A_emp ae: ssn = ssn(ae);
    CSD_B_emp be: ssn = id_to_ssn(id(be));
  FUNCTIONS
    CASE ae
      name = name(ae);
      salary = pay(ae);
    CASE be
      name = name(be);
      salary = salary(be);
```

```
    CASE ae, be
      salary = pay(ae) + salary(be);
    PROPERTIES
      bonus Integer;
  END;
```

As we consider the above mentioned derived types, the type *CSD_emp* can be explained as a type where the employee is working in both universities and is therefore saved within both databases and also works in the correctly defined buildings and department. Meaning for our case:

- create a new derived type
- that is derived from two other types in our case
- CSD_A_emp and
- CSD_B_emp
- with the restrictions of
- dept and
- location
- and map the ssn to id for all types within the mediator *Ta*
- and for the case that the employee is working in both universities,
- perform an addition of the two columns *salary* and *pay*

The example should just show, that it is possible to inherit from other derived types to create new derived types.

The last derived type that needs to be mentioned is the type *Full_time*, this type is a subtype of the derived type *CSD_emp*. [RJK03] formulated the following code sample:

```
create derived type Full_time under CSD_emp e
  where salary(e)>50000;
```

Here all employees are considered of the above mentioned derived type *CSD_emp* where the salary is over 50 000. Within this new derived type called *Full_time* a new function is defined called *office*. The code sample can be found in [RJK03]:

```
create function office(Full_time)->Charstring
  as stored;
```

This means that a new type is saved as an attribute of *Full_time* and is saved as a Charstring. How the mediation is performed within OGSA-DAI accessing Amos II will be explained in Section 6.2.

3.5 Summary

The Amos II system is a quite complicated one and if we see the different query processing possibilities and also the query decomposition a lot of deeper reading needs to be done for a better understanding. This chapter should give just an overview of the different parts involved during mediation over Amos II peers. The most important points can be found within the beginning of the chapter, meaning the introduction, the explanation of the architecture of Amos II and also the functional data model. Also very interesting concerning the informations of the practical part is the example taken from [RJK03], showing the creation of the derived types and proxy types and starting the queries against the distributed data resources. As our main idea was to provide the functionality of Amos II within OGSA-DAI a similar example explaining the mediation functionality can be found within the use case concerning the mediation in Section 6.2. But for implementing and understanding for example the terms *Activities*, *Workflows* and *Data resource* of OGSA-DAI and therefore also the possibility to access Amos II with the help of OGSA-DAI please read before switching to the chapter *implementation*, Chapter 4 that chapter will explain the most important terms of the framework we will use for our implementation, OGSA-DAI. If additional information to Amos II is needed the most important paper can be mentioned as [RJK03], but also [JKR99] and [JR99] can be mentioned. For the practical part of accessing and using Amos II also different papers can be found. The most important are [Ris], [ER] and [FR08].

Chapter 4

OGSA-DAI

4.1 Introduction to OGSA-DAI

Our main idea is to combine two technologies: Amos II and OGSA-DAI. On the next few pages, an overview of OGSA-DAI will be given and also some deeper insight to this topic. OGSA-DAI stands for *Open Grid Service Architecture - Data Access and Integration.*

Within the last few years, different research teams have worked on the development of useful grid implementations. OGSA-DAI is a framework, that can be accessed with the help of web services. OGSA-DAI can be responsible for the different actions concerning data manipulation. Different steps can be mentioned as important, for example the access of the different data, but also the integration, transformation and delivery of different data sources. The main focus lies on handling and manipulating data within the grid (see [AOD]). As OGSA-DAI is also suitable to integrate heterogeneous data sources, it might be a quite interesting research point to allow the access of an Amos II peer and therefore also the access of the data that can be accessed with the help of Amos II. For Amos II different wrappers were implemented and we can also think about designing a wrapper that can also access OGSA-DAI resources. But the first approach is to extend OGSA-DAI with new features to make the access of Amos II with the help of OGSA-DAI possible. The project of developing OGSA-DAI can be split into three different phases mentioned in [AOD]:

- Phase 1: starting February 2002 until July 2003
- Phase 2: starting October 2003 until October 2005
- Phase 3: starting November 2005 until October 2008

Different research parties participated during this project, the most important is the EPCC (Edingburgh Parallel Computing Centre).

As shown in Figure 4.1, OGSA-DAI consist of several components. The activities and data resources are extension points wheres the core contains the framework itself. The persistence and configuration can be exchanged too but this is not so widely used. The framework of OGSA-DAI should provide an uniform way to access the different data resources, independently what kind of data resource should be accessed. For the different data resources different kinds of activities can be performed. An overview of the most important and for our work considered activities will follow within the next few pages.

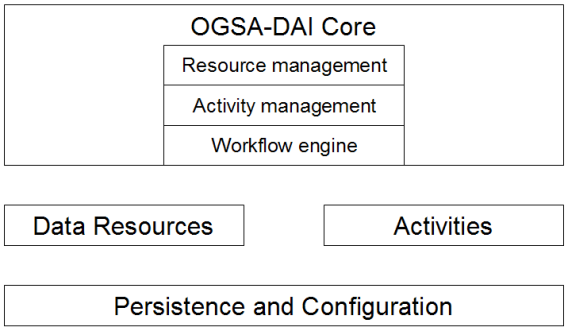

Figure 4.1: OGSA-DAI components [AOD]

4.2 Data resources within OGSA-DAI

Within OGSA-DAI different data resources can be accessed. The most important also have different activities that should enable the access of such resources and uses these resources. The data resources that can be accessed at the moment are (see [KMA]):

- relational databases,
- native XML databases and
- file systems.

The different data resources need to be accessed with the help of different drivers. OGSA-DAI has been tested with quite a lot of different commercial/non-commercial products related to relational databases and XML databases. OGSA-DAI can access different relational databases, for example *mySQL*, *IBM DB2*, *Microsoft SQL Server*, *Oracle* and *PostgreSQL*. This short list of relational databases includes just the data resources OGSA-DAI is tested with, of course it is possible that also other products can be used, these products needs to be tested by the user itself (see [AOD]). XML-databases have also some products for example *eXist* or *Apache Xindice*.

For every of these supported data resources also a set of given activities are presented. Within the properties of the data resource the list of activities that can be performed over such a data resource are given. Also other informations needs to be saved within the properties of a data resource. A data resource needs an unique ID, the creation time and the termination time. As we want to extend OGSA-DAI with the help of a new data resource, making it possible to access an Amos II peer, a deeper introduction of our own designed data resource will follow in the implementation Chapter of our work (see Section 5.1.1).

For relational databases also different activities are given, examples of activities that can be performed are for example *SQLQuery* or *ExtractTableSchema*. This two examples are here mentioned, because within our implementation we will formulate own activities that can handle these activities over a new data resource in our case *AmosResource*. Our main focus lies on a new set of activities, that on the one hand reuses the given activities for relational data resources and on the other hand writing completely new activities that can handle the special functionality of Amos II (see Section 5.1.1), for example the handling of wrapped or mediated data resources.

4.3 Extension points of OGSA-DAI

As mentioned before, OGSA-DAI is a framework and can be extended by different means. Within [AOD] different ways of extending OGSA-DAI are described:

- Extension with the help of new *activities*
- Extension with the help of new *data resources*
- Extension with the help of the *presentation layers*
- Extension with the focus on *security*
- Extension with the focus on *database access*

The focus of our work lies on the extension of OGSA-DAI by writing new activities for a new data resource. The new activities are all mentioned to be part of the new data resource in our case *AmosResource*, a data resource that should make the access of Amos II peers within the grid solution of OGSA-DAI possible. Also new activities are implemented as mentioned above, existing activities for relational data resources are rewritten to make the access of the relational data sources over Amos II peers available. More activities are formulated to grant access to a schema saved within an Amos II peer or to retrieve the results of a query against an Amos II peer (see Chapter 5). The list shows, that also other extension points are given for example the part of presentation layers or also the restriction/extension with the help of database access. As our work does not contain such extensions no deeper explanation will follow.

4.4 Activities of OGSA-DAI

OGSA-DAI delivers a quite large set of different activities that can be used to perform actions against different data resources. The activities can be splitted into three different groups (see [MKM+]):

- *Statement activities* that are responsible for the access of the different data resources and to perform different actions.
- *Translation/transformation activities* can perform the changes of the format of the data.
- *Delivery activities* that allow delivery techniques from other parties.

The different activities can be also part of a workflow. A workflow contains of different activities, where the different inputs and outputs are used to build a chain between these activities. For every of our implemented activities own workflows were created, where the different activities are chained to perform different actions. The different workflows can be distinguished by different means (see [AOD]):

- *Pipeline workflow*: different activities are performed within a chain. The output of one activity is the input for the next activity within the chain.
- *Sequence workflow*: means that there are more than one pipeline involved and the second pipeline starts for example than when the first pipeline has finished.
- *Parallel workflow*: means different workflows are executed in parallel.

4.4.1 Example activities

The list of the possible activities is quite long and the different workflows that can be performed are even longer. Different activities can be chained to receive different results. A possible chain can be (see [AOD]):

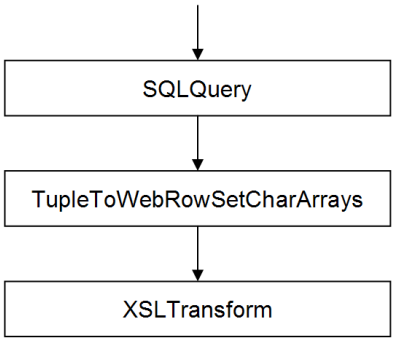

Figure 4.2: Workflow chain (see [AOD])

- SQL Query against a mySQL database. The input of this activity is a *expression* that should be a valid *SQL-query* (see [AOD]).
- The result of the above done activity is a *Tuple*.
- The result of the SQL query (format: *Tuple*) can be transformed with the help of the activity *TupleToWebRowSetCharArrays* into a *WebRowSet XML* representation (output).
- The output of the before mentioned step can be the input for the next step meaning transforming the *WebRowSet* to HTML with the help of *XSLTransform Activity*

A second workflow can contain three different steps containing three different activities, as mentioned, the output of the first activity can be seen as the input of the second activity and the output of the second activity is the new input of the third activity and so on:

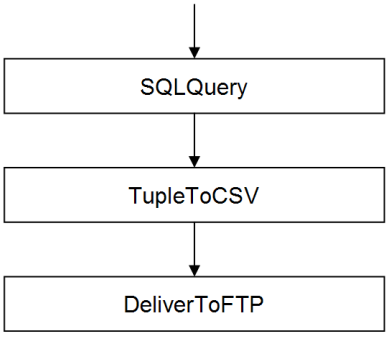

Figure 4.3: Workflow chain (see [AOD])

- SQLQuery-Activity: the input of this activity is an expression that needs to be an valid SQL-statement. The output of this activity is a list of *Tuples*.

- TupleToCSV-Activity: The output of the previous activity *Tuples* can be used as an input to the *TupleToCSV-Activity*. Within this activity the list of Tuples will be transformed into a CSV-formatted output. The output is a list of *chars*.
- DeliverToFTP-Activity: The list of chars than can be delivered to a FTP-server. For the *DeliverToFTP-Activity* different inputs are necessary, first the data of step two (chars), second the filename and also the host. Afterwards it is possible to go to the FTP Server and there will be a file with the saved data in a CSV-format.

This chain contains different activities involved, the result of one activity can be used as an input for the next activity and so on. In our work different activities were new implemented and therefore also new pipelines were created. The workflows are quite similar to the workflows given in the documentation, but the main purpose is to use our new activities with the new implemented data resource in our case *AmosResource*.

An example workflow of our implementation could contain the following three activities:

- *AmosGetAvailableTables*Activity and
- *DeliverToRequestStatus*Activity.

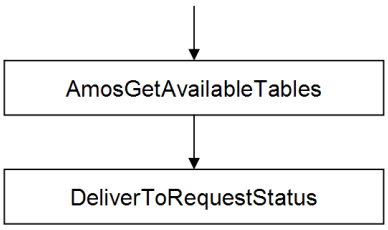

Figure 4.4: Workflow chain

The first activity is responsible to give information about a database wrapped within a Amos II peer. The main input within this activity is the name of the database where the table names should be queried. The idea is to send a request to OGSA-DAI, this request contains a query against an AmosResource and within this Amos II peer a database is queried and therefore the activity should retrieve the information of the database. For the client it should make no difference, if it sends the request against a relational database within the OGSA-DAI system or against a Amos II peer. The result of the first activity is a list of table names.
The second activity propagates the result of the first activity to the client. In our case, the result is a list of table names within the database that is accessed with the help of the Amos II peer.

Our main focus of this work lies on activities concerning the relational data resources, of course it is possible as mentioned above to perform queries also against XML data resources, therefore own activities are written for example *XPathQuery* or *XQuery*. Because of the reason, that our focus lies on relational databases no workflows containing XML data is involved, concerning more information of XML activities please read [AOD].

For every of our activities own workflows are written. For further information please read Chapter 5.

4.5 Projects using OGSA-DAI

Within the last few years a lot of projects have been presented that used the OGSA-DAI framework. The projects are summarized and mentioned in [JK08]. Here a short overview of the most important projects will be given. The most of them are related to grid computing, but also a project is involved that should show that also business applications can use this approach of accessing heterogeneous data resources.

The main advantages of OGSA-DAI can be summarized as (see [AOD]):

- OGSA-DAI is perfect to be used within the grid. The idea of using web-services makes an easy access possible.
- Workflows make it possible, that just one interaction within the grid is started and different activities can be performed within one workflow. As we mentioned above, one workflow is started and during this workflow, different activities can be chained.
- One main advantage of OGSA-DAI is the huge number of given activities for the different topics, for example accessing XML data resources or accessing files.
- For the data resources, that can be accessed at the moment with the help of OGSA-DAI, namely relational databases, XML databases and files, different activities are given to perform different actions for example queries or updates.
- OGSA-DAI is extensible, meaning that new data resources can be added, for example for this thesis the access to Amos II peers with the help of OGSA-DAI, but also new activities can be written (see for the implementation in Chapter 5)
- OGSA-DAI is used because there is no platform dependence. OGSA-DAI runs on every machine that is Java-enabled.
- OGSA-DAI provides security if wished, for example the restricted access to web services or data resources. In our thesis the point of security is not considered, of course, there is the possibility to restrict the access to the OGSA-DAI data resource that is created during the implementation.
- As mentioned before, for the client there is no difference between accessing a relational data resource and accessing the relational data resource over Amos II. The client is not able to see where the databases are located and what product is used.
- The implementation of this work is done in the programming language Java, but it is of course possible and OGSA-DAI supports this option to access the different web services with the help of clients written in other programming languages.
- OGSA-DAI uses in our implementation Axis and Tomcat, but of course also the use of Globus Toolkit is possible.

As we see the list of the advantages and reasons for using OGSA-DAI is quite long and of course other projects use parts of the OGSA-DAI framework for accessing the different data resources. Here a small overview of some projects using OGSA-DAI will follow.

4.5.1 First DIG

One of the earliest projects is First DIG and the main focus here lies to show that OGSA-DAI can also be used within commercial environments (see [fir]). The project shows how the data of the *First South Yorkshire bus operational environment* can be accessed over OGSA-DAI and how the analysis over this data resources can be done. OGSA-DAI should provide an easy access to the different data resources. Within this company different data resources are saved within different databases. The most important ones are mentioned in [SCG+]:

- Data resource named *Customer Care* where the communication between the company and the customers is saved.
- The second data resource saved the mileage for the bus services. In [SCG+] the data resource is named as *Vehicle mileage*.
- Also a data resource saves the tickets that are daily sold. In [SCG+] the data is saved within the data resource *Ticket revenue*.
- *Schedule Adherence* includes a satellite tracking system, responsible for showing if the different buses are arriving and leaving at the scheduled time (see [SCG+]).

The different data resources are saved in different databases, also geographically distributed. The data resources used within this company are *relational databases*, *ODBC resources* and *Cobol files*. For the client it is necessary that the distributed data resources are accessible and also that the client has a single access point. Also the different locations of the data resources are hidden from the user. For accessing the different data resources the company used OGSA-DAI and this project shows that OGSA-DAI could also be used within economic topics, not just for grid environments.

4.5.2 VOTES

VOTES stands for *Virtual Organisations for Trials and Epidemiological Studies* and the different data saved can be used by different research teams. Therefore *virtual organizations* were built to enable the access of different data by different persons. Not every data should be accessible by all users and here the *virtual organizations* are a easy possibility to ensure the secure access of the different data. The main focus of VOTES were the access and the storage of the different clinical data and therefore also *patient information*, *studies* and other *data* were saved. For further information about the medical data saving within grid solutions please read [SSA] and [Sin04].

4.5.3 ADMIRE

ADMIRE (Advanced Data Mining and Integration Research for Europe) is a quite new project. The main idea of ADMIRE is to create a framework that can use heterogeneous distributed data resources. The main ideas of ADMIRE are (see [adm]) :

- The creation of a model to enable data exploration, data integration and data mining.
- Also important is the fact of extraction needed information.
- The integration and interpretation of heterogeneous data is important to support the data mining and knowledge experts.

Within ADMIRE a library should exist, containing different configurable web-services. These web-services are built upon OGSA-DAI. The combination of the different data resources can be quite difficult. Within [ABC+08] different difficulties are mentioned, for example different owners of the data, different access protocols or as mentioned before different types of data resources for example relational databases or data saved in the XML-format.

One main challenge is also the solving of structural heterogeneities. Within OGSA-DAI a set of different activities for performing different actions are given and can be quite easy extensible and therefore also used for the integration within ADMIRE.

To prove the concept of ADMIRE different usage scenarios will be introduced and discovered (see [ABC+08]):

- *Customer relationship management*: the interest of research is to find out, why the clients changes within the mobile phone market the provider for example. Within this usage scenario a large number of records is given, for example the saved number of messages and phone calls. The different data needs to be sampled, preprocessed and afterwards mined to get predictions for the reasons of changing the mobile phone provider.
- *Gene expression in the developing embryo data*: the idea is to save and understand the different data of the normal development of a house mouse. The main idea is to find the spatial gene expression patterns. Therefore a large number of data needs to be saved.
- *Environmental factors affecting river management*: this project should show which environmental factors can effect the river *Váh* in Slovakia. Different data needs to be considered from different data owners, for example meteorological data or water measurements. Within this project the data mining should make it possible to predict parameters for the potential risk of floods (see [ABC+08]).
- *Common factors in data-center operational incidents*: the main idea here is to receive data from the different data centers and therefore to receive the possibility to predict future operational incidents and also to avoid the incidents if there are predicted soon enough.

The main ideas are to handle the different data sources from different data owners and therefore make future work concerning data mining easier.

4.5.4 SEE-GEO

SEE-GEO stands for SEcurE access to GEOspatial services. The main idea is here to use OGSA-DAI to provide secure access to different data resources, for example geospatial data (see [see]). SEE-GEO should also increase the availability of the different computational resources. Focus lies also on the federation of the different data resources. In [AOD] an overview of SEE-GEO is provided. As mentioned above the main focus lies on the federation of different data resources, within SEE-GEO two are mentioned (see [AOD]):

- census statistics
- borders data

The two different data resources are accessed over two different services, on the one hand, *Geo-data access service (GDAS)* and on the other hand, *web feature service (WSF)*. OGSA-DAI was used to enable a service that can access two different data resources and therefore combine two different data resources to retrieve one result.

4.5.5 GEOGrid

GEOGRid stands for *Global earth organization (GEO) grid* and was developed in the *Grid Technology Research Center, National Institute of Advanced Industrial Science and Technology (AIST)* in Japan (see [MKN+]). GEOGrid contains different data from different machines and satellites. The following two are the most important [Sek]:

- Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER)
- Phased Array L-band synthetic Aperture Radar (PALSAR)

These two machines provide a large number of data. ASTER more than one hundred terabyte and PALSAR within 5 years over one petabyte of data.

OGSA-DAI is used to allow the access of the different data resources and also the integration of the *virtual organization* concept that different research teams and members are grouped to virtual organizations and different access possibilities are given for the different researchers.

4.6 OGSA-DQP

4.6.1 Introduction to the OGSA-DQP query processing

OGSA-DQP is described in [OD], [AMP+03] and [NAN+]. The main ideas in OGSA-DQP[1] are service orchestration and data integration. OGSA-DQP (see [AMP+03]) has the following possibilities:

- The processing of queries over services or Grid Database Service (GDS) (see [AMP+03]).
- Dynamically finding out which resources are necessary to perform a query and the evaluation of this query.
- Using implicit parallelism to handle complex requests.
- Having consistent access to database meta data and can interact with databases.

The main steps of the OGSA-DQP query processing are summarized within Figure 4.5 (see [ogs]) and each number in this Figure describing a processing step is explained within the next few paragraphs. OGSA-DQP contains of two different parts (see [OD]):

- *The OGSA-DQP Coordinator*: the coordinator is responsible for the handling of the distributed queries. A query plan consists of an execution schedule showing which part needs to be performed to retrieve a satisfying result. Within a coordinator there are two different data resources. The one data resource is called *DQP Factory Data Resource*, the second one is the *DQP Data Resource* (see [ogs]).

[1] OGSA-Distributed Query Processing

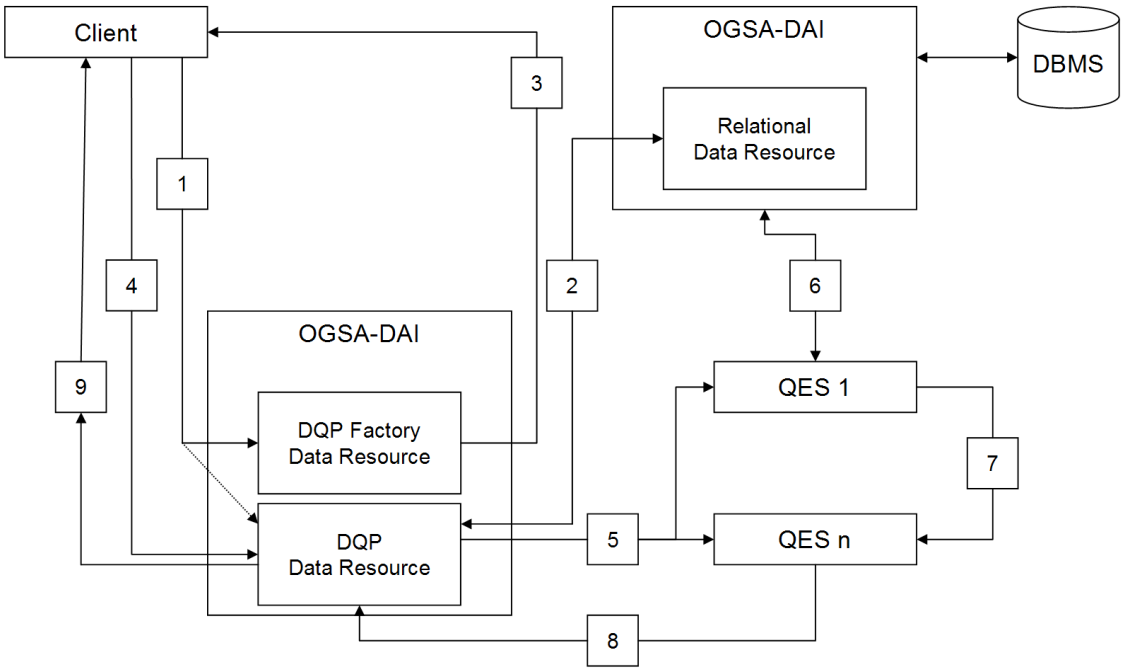

Figure 4.5: OGSA-DQP [ogs]

- *The Query Evaluation Service* is also called *Evaluator* (QES in Figure 4.5) and is responsible for the execution of the query plans. The query plans are not directly generated by the coordinator but with the help of a compiler. Every evaluator is responsible for a part of the query that the coordinator has assigned to it.

1. As for every OGSA-DAI interaction it is necessary to perform the deployment of the resources. This part is a quite complicated one, so for further information how to deploy an OGSA-DAI resource the main reference can be found in [OD]. The technical background for setting classpath for example or other steps necessary to retrieve results from OGSA-DAI are not mentioned within this thesis, as they are always the same for the different projects and are not indicated to be special for our purposes. After the deployment a client is able to access the deployed Data resource (within OGSA-DQP the DQP Factory Data Resource) and than can configure a OGSA-DQP Data Resource.

   The access of the *DQPFactory Data Resource* is done like for every other data resource within OGSA-DAI, with the help of activities. The activity is called *DQP-Factory activity*. The framework of OGSA-DAI handles than the interaction with the Data Resource. The *DQPFactory Activity* is as mentioned above responsible for the access of the *DQP Factory Data Resource* and this data resource is than responsible for the deployment of the other resource called *DQP Data resource*. With the help of a XML document it is exactly defined the configuration of the Data Resource. The configuration of the DQP Data Resource includes the databases that should be evaluated and also the different evaluators (see [OD]) that are used to perform the query. If the activity is performed correctly and contains the right parameters within the above mentioned XML document a DQP resource is created and a resource ID is given. For the topic of resource ID please read Chapter 5 of our implementation

containing also the information of the topic of *resource ID* because for the access of Amos II also a new data resource is created and therefore a unique ID is necessary.

2. Step two shows that the DQP data resource can access the different data resources involved. OGSA-DQP at the moment can just access relational databases but for the future also the access of XML resources or files could be possible. In the second step as the figure indicates, the DQP Data resource communicate with the relational data resource and the relational data resource accesses the in step one defined databases. The different schema of the databases than are imported within the DQP Data Resource. A quite similar step can also be performed within Amos II. Here also schema of relational databases can be imported with a special function.

3. After step one and step two are performed, the *DQP Factory Data Resource* sends the result of the request done in step one to the client again. Here the necessary information is saved, for example the resource ID of the resource that is dynamically deployed in step one (see [ogs]).

4. Step 4 shows that a client sent a valid SQL query within a *DQPQueryStatement*-Activity to the *DQP Data Resource*. The *DQP Data Resource* than performs some important steps that are not visible within the figure, namely parsing, optimizing and scheduling of the query (see [ogs]). After the three steps are performed within the *DQP Data resource* a query plan is given. The creation of the query plan is done in this step and within this query plan different partitions are involved. For every partition (also sub-query) the included evaluators are defined.

5. The different sub-queries are sent to the evaluators (in the figure described and named as *QES*).

6. The sub-queries can be performed with the help of evaluators that access directly the data resource or

7. by evaluators that interact with other evaluators (see step 7 of the figure).

8. After all parts of the query plan are executed, steps 8 and 9 are performed. Meaning in step 8 that the evaluators give the results to the DQP Data Resource and the DQP Data Resource passes the result if asked to the client.

In Figure 4.5 within step 1 a dotted line can be found. This line indicates that it is possible to access directly the DQP Data Resource if the deployment is already performed, meaning that the step of sending the parameterized XML document is already created and the data resource called within our case *DQP Data Resource* is already set up and has a resource ID. When this step is already performed, it is not necessary to do the steps one to three, because the deployment and configuration is already done and does not need to be performed every time when the client starts a *distributed query processing* request (see [ogs]).

The different steps may sound quite complicated, especially the deployment of the OGSA-DQP framework and the setting up of the *DQP Factory Data Resource* is quite difficult. As we have the possibility to perform distributed queries against different data sources with the help of Amos II peers, this should make an easy access to mediated data resources possible.

4.6.2 Comparison OGSA-DQP and Amos II

OGSA-DQP and Amos II are quite similar, the main purpose of these two approaches is to perform queries against distributed databases. While in OGSA-DQP as mentioned before, the data can be stored within a defined grid environment, Amos II can contain of one or more Amos II peers and these peers can access the different data sources. It is quite easy to access the distributed databases due to the fact, that for every Amos II peer it can be defined which resources can be accessed. This is done with the help of wrappers where it is defined, which the different data resources can be accessed.
The main differences are:

- OGSA-DQP allows queries only on relational databases whereas Amos II can perform mediated queries on every kind of data resource for which a wrapper with an object oriented view is provided.
- OGSA-DQP allows to distribute the computation efforts for the query plan calculation on computing resources not involved in providing the mediated data or the access to it.
- Amos II supports a more extensive mediated query language in AmosQL compared to the subset of SQL supported by OGSA-DQP.
- A Amos II peer providing a mediated schema can delegate the access to the mediated data to other peers so that the client executing the query has not to be aware of these details.
- OGSA-DQP is better suited for ad-hoc mediation and allows to easily delete a mediation configuration.

Chapter 5

Design and Implementation

5.1 Overview

The main focus of this work is the extension of the given grid solution OGSA-DAI with mediation capabilities. OGSA-DAI is as mentioned before a framework offering the access to different data resources with the help of activities. Amos II uses the wrapper/mediator approach and is in our solution responsible for the mediation and uniform access to data sources. This thesis tries to connect these two independent systems of two different research areas to a combined mediator/grid solution and should reuse the advantages of these systems. The grid solution OGSA-DAI was first introduced to enable the shared access to huge data amounts by various research teams. Amos II, the second used system within our implementation, focuses on the mediation of different data sources. The approach of using object-functional databases was chosen to ensure the inheritance between the different types and also to perform queries over data sources that can not be represented within a relational table (see [JKR99]). The need of this work is given to use the data sources mediated within Amos II within the grid and therefore different software components are implemented to connect the grid framework of OGSA-DAI with the mediation system of Amos II. The main requirements for the implementation are:

- service-based access to Amos II via OGSA-DAI client activities. This means, the user has not to be aware of the location of the Amos II peer and the access is performed only via OGSA-DAI activities.
- performing an activity to receive the schema information (meta data) from the Amos II peer and information retrieval with the help of the declarative query language AmosQL.
- leveraging the use of the retrieved information by providing a consistent set of activities that can be composed with the existing OGSA-DAI activities (see [AHH+]). To allow this composition of activities, well known input and output values should be used.
- SQL-based query access to wrapped RDBMS in Amos II. This includes also the retrieval of schema information about the tables saved in the databases.
- non-intrusive access to Amos II meaning that other users are not affected and the Amos II system is not changed by the actions performed against the Amos II peer.
- usage of existing Amos II peers accessible in a TCP/IP based network.

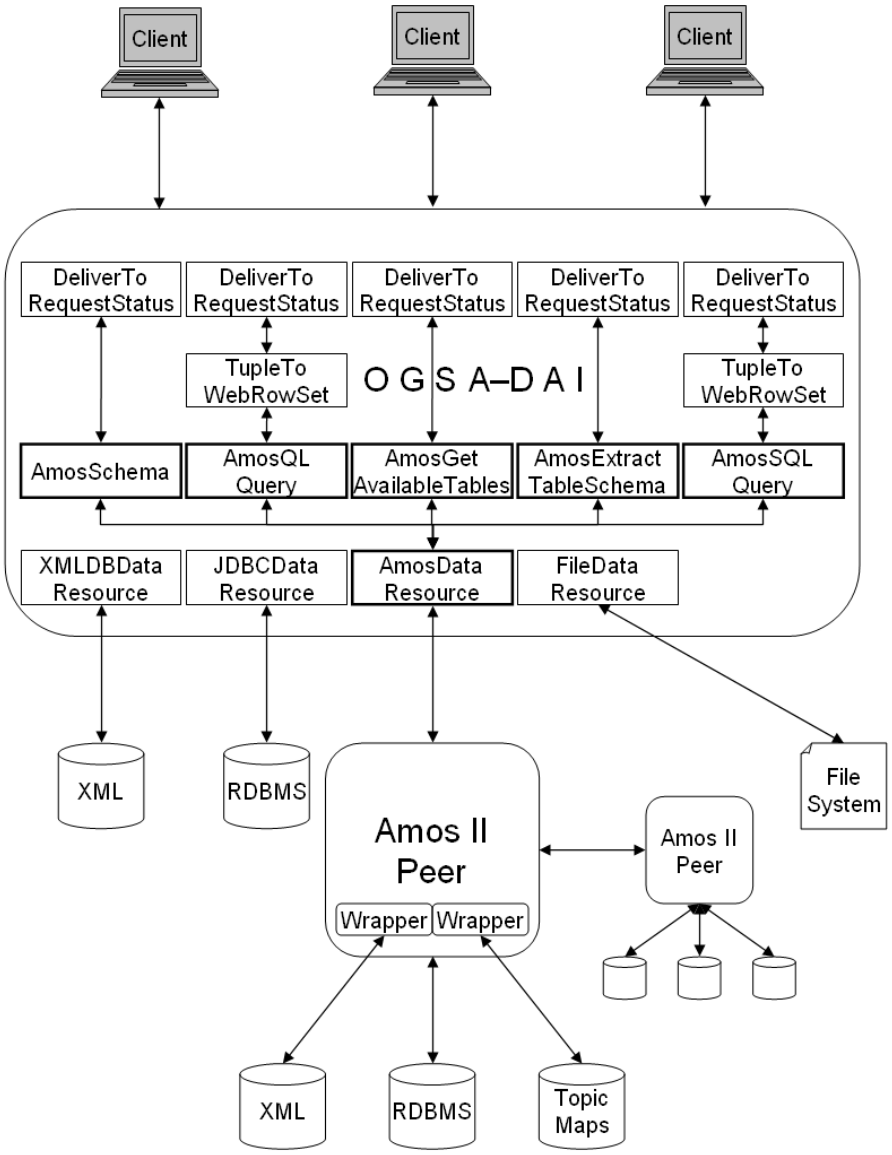

Figure 5.1: Overview of new activities and data resource

In the next two subsections this thesis discusses the concrete possibilities to extend OGSA-DAI and access Amos II. After this introduction of the extension and access possibilities the implemented activities will be described in detail. In advance, Figure 5.1 shows the new activities and the data resource designed and implemented in this chapter. Bold depicted are all extensions to the default OGSA-DAI framework.

5.1.1 Extension Points of OGSA-DAI used

As we mentioned before (see chapter OGSA-DAI), different extension points are provided by OGSA-DAI. In our approach, we will use two types of them.

1. A data resource to provide low-level access to Amos II on the OGSA-DAI server.
2. A set of new activities to perform schema and data retrieval.

Data Resource

OGSA-DAI has a set of different data resources that allow the access to RDBMS, XML-DBs and CSV-files (see [AOD]). Since none of these default data resources allow the access to Amos II a new data resource specific to Amos II needs to be designed. The responsibilities of this data resource are straightforward:

- Providing a shared connection object that allows activities to perform queries and schema retrievals.
- Configuring the host and name server of the Amos II peer which the connection object should access.
- Defining which activities can use this data resource.

Activities

The implemented activities can be split into two different logical groups. All activities are using the data resource described above, but different parts of the Amos II system will be used. On the one hand access to the mediation enabling object-functional data model and on the other hand direct access to wrapped RDBMS. In both cases a client application that wants to perform queries on the data resource needs information about the corresponding schemas. Since these schema information is fundamentally different among those groups two metadata representations are used. For the mediation data model a new specific format is used (see 5.3.2) and for the direct access the format known from the relational OGSA-DAI activities is used (see [AOD]).

The representation format for the RDBMS access should be identical to the format delivered by the default *ExtractTableSchemaActivity*. This default activity allows the meta data gathering using a connection provided by the default RDBMS data resource. Amongst other functionality the default activities to use the RDBMS data resource allow the retrieval of table names and the execution of selected SQL queries returning a bag of tuples. The corresponding activities for the wrapped RDBMS in Amos II should follow these conventions so that users and applications that are comfortable with the default activities can easily adopt to this Amos activities.

The newly designed activities for the object-functional data model have no direct corresponding default activities. Therefore the challenge lies in designing and implementing activities that follow proven OGSA-DAI guidelines but on the other hand are capable to take advantage of the Amos functionality especially mediation. Two building blocks turned out to be indispensable. First retrieving the types with their methods and attributes considering the type hierarchy. This system meta-data allows the Amos expert to compose AmosQL queries to retrieve information. The execution of AmosQL queries and the allocation of the results is the second building block of the mediation-enabling activities.

5.1.2 Access Points of Amos II

The Amos data resource described above has to provide access to an Amos II peer. In the Amos documentation (see [Ris] and [ER]) the access possibilities are described as external interfaces to Amos II. The following programming language specific interfaces are included in the Amos II download:

- C
- Lisp
- Java

Further interfaces building on top of these interfaces have been developed (e.g. for PHP see [Wer04]). Besides this programming language choice, Amos II can be accessed by different means. First the client application can choose between a call-level interface named callin interface and a callout-interface that makes use of foreign functions. The callout interface allows the execution of external subroutines programmed in C whereas the callin interface does not provide this opportunity. The callout interface does not fit to our main requirement defined above that the mediator should not be affected by the access via OGSA-DAI. This leads to the decision to use the callin interface. For this type the developer can choose between an embedded query interface and an fast-path interface. The embedded query interface uses a declarative query language AmosQL that is interpreted and executed by the Amos peer targeted (see [Ris]). To avoid the overhead of interpretation and compilation predefined functions can be called with the fast-path interface. Even though it is recommended (see [Ris]) to make extensive use of this interface this implementation uses the embedded query interface since it does not require to define derived functions as this could probably affect other users of the mediator.

Beside the callin-interface options the client application has two possibilities to get a connection to a peer. The client-server connection allows the concurrent access of multiple clients to a Amos II server. Hereby the server and the client can communicate over a TCP/IP based network. The tight connection on the other hand starts an Amos II server directly linked in a C program. For obvious reason this approach is not feasible as this implementation should allow to access any Amos II peer reachable in the network.
To recapitulate our current findings, the callin, embedded query interface is preferred using the client-server connection. As described above multiple programming language interface allow this kind of access. Since OGSA-DAI is a grid framework implemented in Java running on an application server it would be obvious to prefer the Java interface to avoid a language gap.
Further advantages can be taken of the fact that the Java interface reports exceptional behavior as Java exceptions. So the client application connecting and accessing Amos II can respond appropriately and easily to these exceptions. Nevertheless the Java interface is built upon the native interface and so the native libraries (DLLs[1] using windows) must be available on the host running OGSA-DAI.

5.2 Data Resource

5.2.1 Introduction and Configuration

The implementation of the OGSA-DAI data resource follows the guidelines and requirements defined in [AOD]. This includes a unique ID that distinguishes the data resource from other data resources available on a OGSA-DAI server. This ID must be provided to client applications so that they can include the Amos II data resource in their workflows. A persistent configuration is not needed for every data resource but for the Amos II data resource it is used to store the name of the targeted Amos II peer and the host on which

[1] Dynamic link library

it is located. These configuration parameters are stored along with the data resource id in a configuration file that has the same name as the data resource.
Furthermore this file provides a type, a creation time and a termination time to the framework (see [AOD]). The type can be chosen among a few different constants and allows the framework to identify the resource as data resource, session resource, etc. The creation and termination time allows a framework controlled creation and termination of the resource.
A list of activities is also content of the configuration file and defines which activities are permitted to access this data resource. Last but not least the data resource implementation class is defined by its fully qualified class name. If a resource wants to make use of the framework security support, a login provider can be defined. Since Amos II has no built in security support the OGSA-DAI login provider would be a simple opportunity to restrict the access to the peer over OGSA-DAI.
For the Amos II data resource the configuration file has the following content:

```
id=AmosResource
type=uk.org.ogsadai.DATA_RESOURCE
creationTime=null
terminationTime=null
PROPERTIES
END
CONFIG
dai.data.resource.nameserver.name=amos2
dai.data.resource.nameserver.host=192.168.0.11
END
ACTIVITIES
at.ac.univie.AmosQLQuery=at.ac.univie.AmosQLQuery
at.ac.univie.AmosSQLQuery=at.ac.univie.AmosSQLQuery
at.ac.univie.AmosSchema=at.ac.univie.AmosSchema
at.ac.univie.AmosGetAvailableTables=at.ac.univie.AmosGetAvailableTables
at.ac.univie.AmosExtractTableSchema=at.ac.univie.AmosExtractTableSchema
END
dataResourceClass=at.ac.univie.dataresource.amos.AmosDataResource
```

Setting the creation and termination time to null means, that the data resource is created on demand and not automatically started at predefined start time. The null value for the termination indicates that the resource is not automatically destroyed at a given point in time. All five activities permitted to access the Amos II data resource are described in following Sections and were implemented for this thesis.

5.2.2 Functional Description

When the data resource is accessed for the first time a client-server connection to the Amos II peer is established. In the data resource implementation class this requires two steps:

1. Initializing the Java interface to Amos II. When using windows as operating system DLLs are loaded by the Java interface. Therefore these DLLs must be available. Common options are setting the Java library path or specific to windows making the DLLs available in the system environment variable PATH.

2. Accessing the configured Amos II peer via the constructor call of the *callin.Connection* class of the Amos II Java interface. This constructor allows to set the name server name and host.

If the connection establishment was successful the retrieved *callin.Connection* is available for all activities that target this data resource. If it failed an appropriate exception is thrown and this ensures that the client can assign the exception to the data resource. The framework takes care of the error propagation. The *callin.Connection* object allows amongst other functionality to use the embedded query and the fast-path interface. When the data resource is no longer needed the *releaseConnection* method of the AmosDataResource class is called by the framework. This method releases the established connection by executing the AmosQL statement *exit*.

UML Class Diagram

Figure 5.2 shows the UML class diagram context of the implemented *AmosDataResource* class mentioned above. *AmosDataResourceState* and its implementation *SimpleAmosDataResourceState* represent the configuration. *AmosResourceAccessor* and its implementation *SimpleAmosResourceAccessor* represent the resource accessor as described in [AOD]. *AmosConnectionUseException* is thrown if the connection establishment fails as described earlier.

Figure 5.2: UML class diagram of the Amos II data resource

5.3 Direct-Mediator-Access-Activities

5.3.1 Motivation

In the attempt of this thesis to combine the benefits of the wrapper/mediator and the grid approaches, these activities should allow clients to access Amos II peers in their workflows.

The focus lies hereby on the access of an existing Amos II peer and using its configured mediation. Despite the fact that only one peer can be targeted at the same time of one data resource the targeted peer may be configured to access further peers.

5.3.2 AmosSchema-Activity

Brief Description

As we implemented our own set of activities to extend OGSA-DAI, the structure of the brief description is identical to the one used in [AOD]. A detailed description of every activity always follows after the respective brief description. The brief description can be found in Table 5.1 on page 56. The brief description contains the different important points for example: *summary*, *activity name* or *inputs*.

Summary	This activity retrieves the types and functions of an Amos II peer configured in a Amos data resource.
Activity name	*at.ac.univie.AmosSchema*
Server class	*at.ac.univie.activity.AmosSchemaActivity*
Client toolkit class	*at.ac.univie.activity.client.AmosSchema*
Inputs	*type*: type *java.lang.String* required. Extent of the schema query.
Outputs	*data*: type *AmosMetaData*
Configuration parameters	none
Activity input and output ordering	none
Activity contracts	none
Behaviour	Retrieves the metadata in a XML representation.

Table 5.1: Brief description AmosSchema-Activity

Description

This activity uses AmosQL statements to retrieve types and functions of a peer. As described in the user manual (see [FHJ+]), several system meta-data functions are predefined for this purpose. The current implementation uses the following ones:

- *typenamed*: to retrieve a type with a given name
- *allfunctions*: to retrieve all functions
- *subtypes*: to retrieve all subtypes for a given type
- *methods*: to retrieve all methods for a given type
- *attributes*: to retrieve all attributes for a given type
- *kindoffunction*: to determine the kind of a given function (derived, stored, generic and overloaded)

The activity input parameter *type* determines the start type for the meta-data retrieval. Currently two values are supported. *OBJECT* retrieves the meta-data for all types including the built-in types whereas *USEROBJECT* considers only all user-defined types. The retrieval follows in both cases the same procedure.

1. Execute an *allfunctions* AmosQL query to retrieve all functions and save them for further usage.
2. Retrieve the type of each saved function with a query using *kindoffunctions*.
3. Execute a *methods* and a *attributes* query to retrieve the method and attribute identifiers for the start type. These identifiers are used to lookup the corresponding complete function information retrieved in the first step.
4. *Subtypes* to determine all subtypes of the start type and continue with step 3 for all retrieved subtypes using them as new start type for each recursive call.

This algorithm leads to a linear increasing effort in relation to the queried types. After all meta-data is retrieved the result of the activity is created from the object-oriented representation. Concrete the XML output format is created by traversing the type tree.

Usage example

To show the usage of this activity an example from the Amos II tutorial [FR08] is used. A part of the worldcup schema defined there is applied on an Amos II peer that is afterwards targeted by the activity. As instructed in the tutorial the types can be created with a few AmosQL *create type* statements, for example *create type Referee under Person*. Here a type *Referee* is created as a subtype of *Person*. The whole schema can be seen in Figure 5.3. Here the rectangles represent types, the ellipse an attribute and the *is-a* arrows subtype relationships. To perform the query for only these shown types the activity input

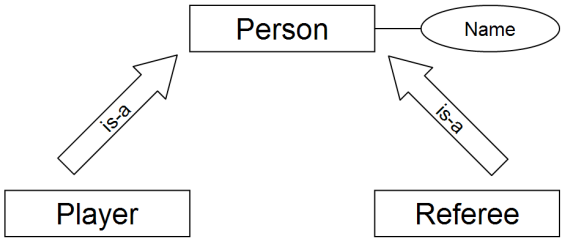

Figure 5.3: Extended entity relationship diagram (see [FR08])

type must be set to *USEROBJECT*. As mentioned above only the meta-data of the tpye *USEROBJECT* and its descendants are retrieved. The XML representation delivered to the client follows here:

```
<type id="37" name="USEROBJECT">
  <type id="1013" name="PERSON">
    <attribute id="74" name="NAME" kind="STORED" />
    <method id="1016" name="PERSON.NAME->CHARSTRING" kind="STORED" />
    <type id="1014" name="REFEREE" />
    <type id="1015" name="PLAYER" />
  </type>
</type>
```

Types and functions (attributes and methods) can be distinguished via the XML element name. The *id* represents the unique Amos object identifier and the XML attributes *name* and *kind* further specify types and functions. The nesting of the *type* elements reproduces the inheritance hierarchy. The complete worldcup tutorial example can be found in Section

6.1.
A client application can use the activity in a workflow as shown in the following code sample.

```
AmosSchema schemaActivity = new AmosSchema();
schemaActivity.setResourceID(new ResourceID("AmosResource"));
schemaActivity.addType(AmosSchema.USER_DEFINED_TYPES);
DeliverToRequestStatus deliver = new DeliverToRequestStatus();
deliver.connectInput(schemaActivity.getDataOutput());
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(schemaActivity);
pipeline.add(deliver);
```

The *AmosSchema* activity is here targeted on a configured Amos data resource (*AmosResource*). The workflow contains beside the AmosSchema activity only a *DeliverToRequestStatus* activity to make the resulting XML available to the client application.

Class-Diagram

Four class diagrams are used to describe the implementation of the AmosSchema-Activity, three for the server activity (see Figures 5.4, 5.5 and 5.6) and one for the client activity (see Figure 5.7).

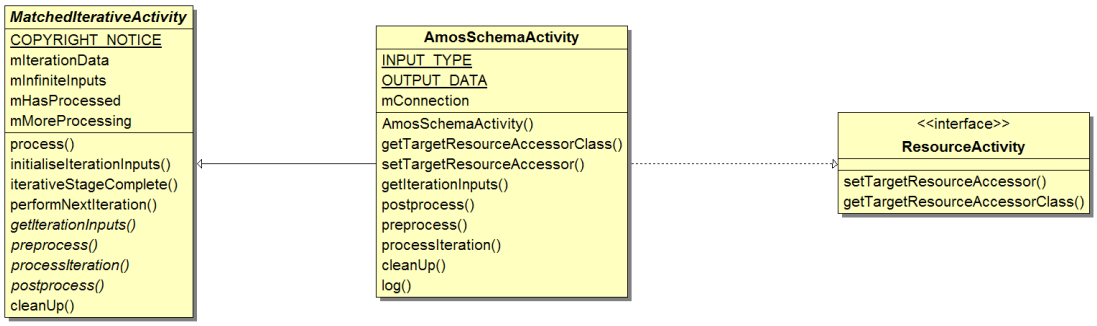

Figure 5.4: Server activity class diagram of AmosSchema-Activity

In the server class diagram (see Figure 5.4), *AmosSchemaActivity* is the main activity class directly used and configured in the OGSA-DAI framework. The task to perform the schema retrieval is delegated to the *AmosSchemaRetriever* class shown in Figure 5.5. This seperation of concerns allows to test the schema retrieving functionality in a test class without depending on a configured and working OGSA-DAI environment. The result of the retrieval is an instance of *AmosMetaData*. *AmosType* and *AmosFunction* are the object-oriented counterparts to the Amos basic building blocks type and function. *AmosFunctionKind* is an *enum* with all possible function kinds. For the transformation of the object network into XML *AmosSchemaToXMLConverter* respectively the implementation *AmosSchemaToXMLConverterImpl* are used. This can be seen in Figure 5.6.

In the client class diagram (see Figure 5.7) only the class *AmosSchema* is necessary. For client applications using this activity in their workflows it is indispensable to use the methods provided by this class correctly. The method *addType* must be called with one of the predefined *String* constants *ALL_TYPES* and *USER_DEFINED_TYPES* and the method *getDataOutput* delivers the XML output.

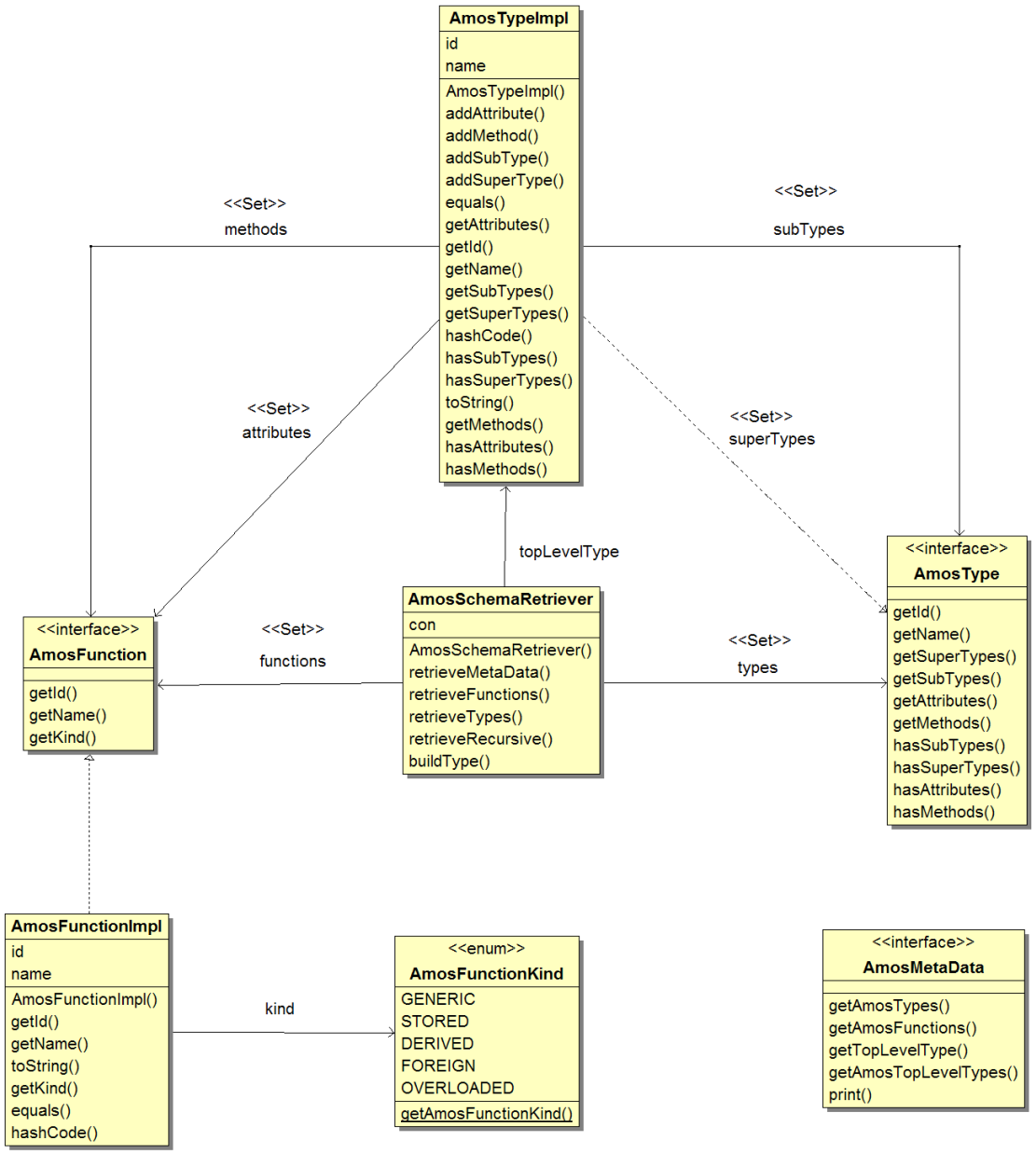

Figure 5.5: Retrieval helper classes of AmosSchema-Activity

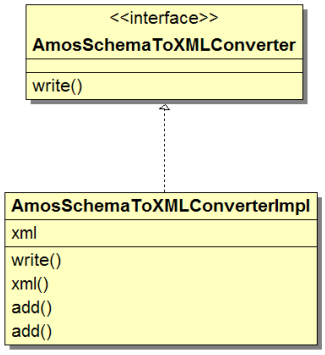

Figure 5.6: XML conversion helper classes of AmosSchema-Activity

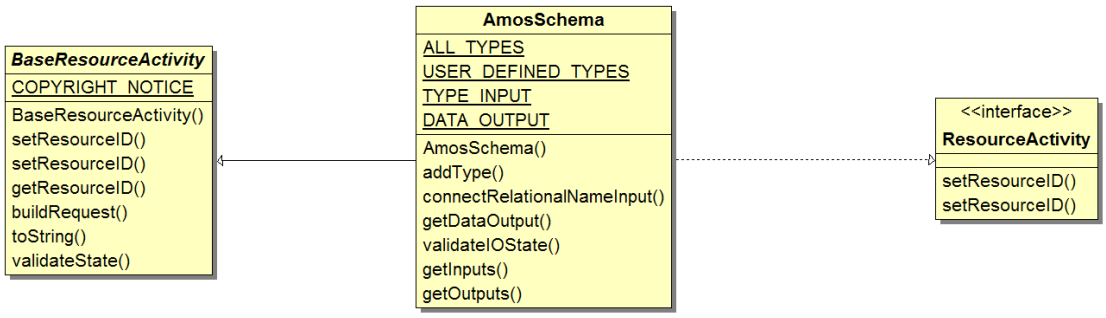

Figure 5.7: Client activity class diagram of AmosSchema-Activity

5.3.3 AmosQLQuery-Activity

Brief Description

As mentioned above, this brief description should show the most important points summarized in Table 5.2.

Summary	This activity performs an AmosQL query against the peer configured in a Amos data resource.
Activity name	*at.ac.univie.AmosQLQuery*
Server class	*at.ac.univie.activity.AmosQLQueryActivity*
Client toolkit class	*at.ac.univie.activity.client.AmosQLQuery*
Inputs	*expression*: type *java.lang.String* required. AmosQL query.
Outputs	*data*: list of *uk.org.ogsadai.tuple.Tuples* with corresponding *uk.org.ogsadai.tuple.TupleMetadata*.
Configuration parameters	none
Activity input and output ordering	none
Activity contracts	none
Behaviour	Executes a AmosQL query and retrieves the results as tuples.

Table 5.2: Brief description AmosQLQuery-Activity

Description

The Amos data resource provides a *callin.Connection* instance of the Amos Java interface to the AmosQL-Activity. This instance is used to execute the query against the peer. A detailed description of the following steps can be found in [ER].

1. Execute the query against the peer using the *execute* method of the *callin.Connection* instance to get a *scan* result containing a rows and columns representation of the query result.
2. Determine the data types of each column by examining the first row.
3. Iterating over each row and reading the column values using the data types retrieved in the previous step.

To determine the data types a Java interface call once for each column in the first row is executed. The Amos II data types must be mapped to data types supported by the OGSA-DAI meta-data description found in *uk.org.ogsadai.tuple.TupleTypes*. The following *callin.Tuple* methods are available to perform this task.

- *isInteger* mapped to *TupleTypes._INT*
- *isString* mapped to *TupleTypes._STRING*
- *isDouble* mapped to *TupleTypes._DOUBLE*
- *isObject* mapped to *TupleTypes._OBJECT*
- *isTuple* mapped to *TupleTypes._OBJECT*

Usage example

A minimalistic code sample shows the usage in an OGSA-DAI client application.

```
AmosQLQuery query = new AmosQLQuery();
query.setResourceID(new ResourceID("AmosResource"));
query.addExpression("select name(host(t)),year(t) from tournament t;");
DeliverToRequestStatus deliver = new DeliverToRequestStatus();
TupleToWebRowSetCharArrays toWebRowSet = new TupleToWebRowSetCharArrays();
toWebRowSet.connectDataInput(query.getDataOutput());
deliver.connectInput(toWebRowSet.getResultOutput());
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(toWebRowSet);
pipeline.add(deliver);
```

The usage of the activity in an client application is similar to the AmosSchema activity. *addExpression* allows to set the AmosQL query and *TupleToWebRowSetCharArrays* is necessary for converting the *tuple* objects into XML. As shown in the *PipelineWorkflow* this can occur only after the *AmosQLQuery* and must occur before *DeliverToRequestStatus* makes the result available to the client.

Class-Diagram

Two class diagrams are used to describe the implementation of the AmosQL-Activity, one for the server activity (see Figure 5.8) and one for the client activity (see Figure 5.9).

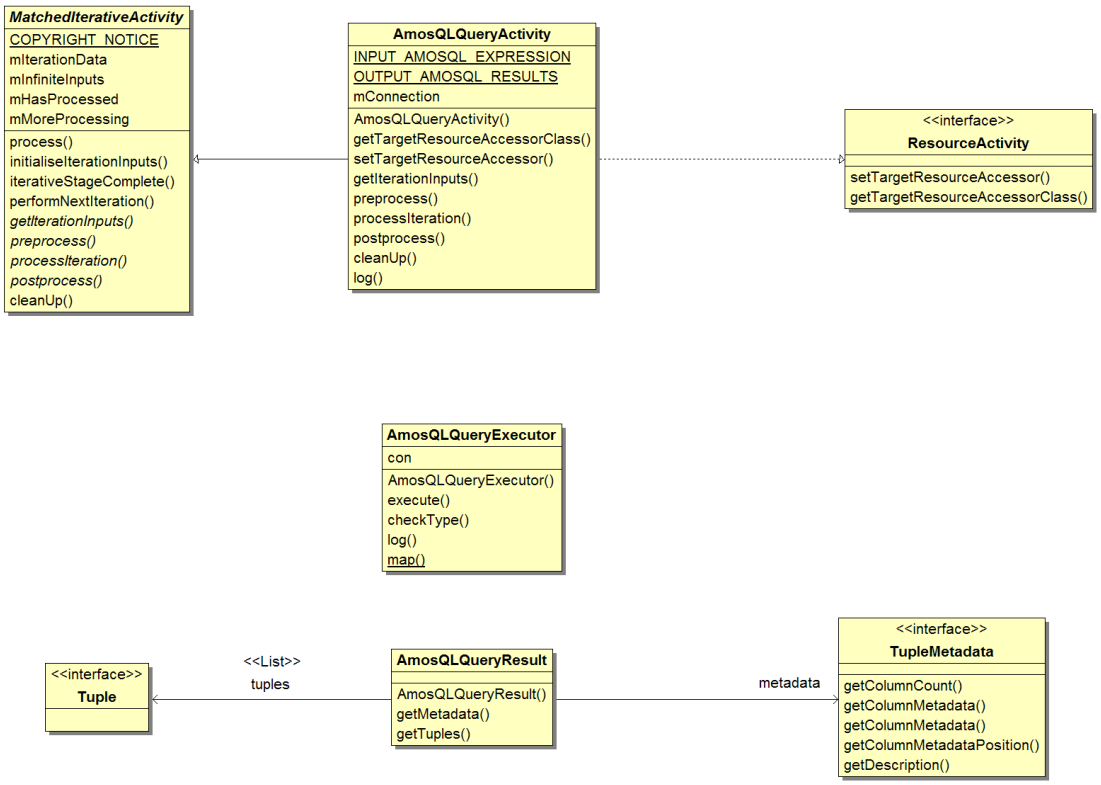

Figure 5.8: Server activity class diagram of AmosQL-Activity

In the server class diagram (see Figure 5.8) *AmosQLQueryActivity* is the main activity class directly used and configured in the OGSA-DAI framework. The task to execute the AmosQL query and retrieve the results is delegated to the *AmosQLQueryExecutor* class. As mentioned above this allows to test the functionality without a running OGSA-DAI. The query result is an instance of the AmosQLQueryResult. This instance contains both data and meta-data. Both parts of the result are represented reusing existing OGSA-DAI classes (*org.ogsadai.tuple.Tuple* and *uk.org.ogsadai.tuple.TupleMetadata*).

In the client class diagram (see Figure 5.9) only the class *AmosQLQuery* is necessary. The usage of this class is obvious. The method *addExpression* sets the AmosQL query used and the method *getDataOutput* delivers the tuple output.

5.4 Wrapped-Data-Sources-Activities

5.4.1 Motivation

Beside the activities mentioned above to directly access the mediated schema of a Amos II peer it can be useful to access a wrapped data source using its specific meta-data and

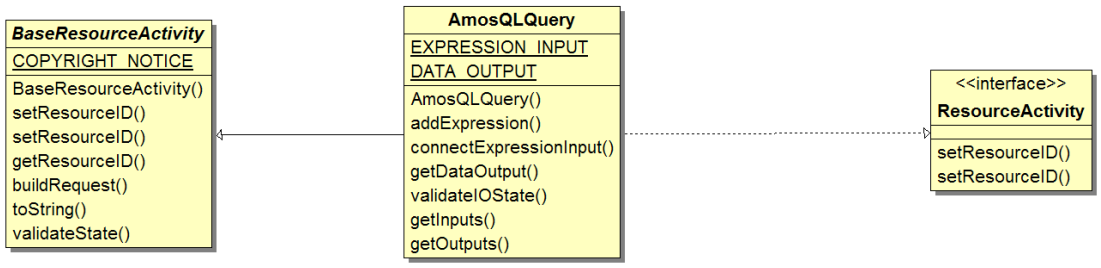

Figure 5.9: Client activity class diagram of AmosQL-Activity

query language. Since relational databases are widely used activities to access data stored in databases are implemented. As blueprint for this activities serve some of the existing *relational activities* shipped with the OGSA-DAI 3.0 release (see [AOD]):

- *GetAvailableTables*-Activity: lists all tables in a database.
- *ExtractTableSchema*-Activity: extracts meta-data for a given table.
- *SQLQuery*-Activity: executes SQL query and returns Tuples.

Reasons to use this new activities to access a wrapped data source can be:

- Retrieving meta-data of a data source. This allows a better understanding for the necessary steps when using mediated queries.
- Reusing existing SQL queries that are complex to build.
- Using queries that offer better performance as compared to accessing a mediated schema. This can of course only be applied if only one data source provides the result of the query.

To use this new activities the Amos II server must be configured with a relational database as wrapped data source. The wrapped access to the database is only supported via JDBC (see [FHJ+]). Therefore the server must be started embedded in a Java program. This configuration is called *javaamos*. Of course a JDBC driver, usually a JAR-file, must be included in the classpath when *javaamos* is started. Currently the drivers for MySQL, Firebird and MSSQL have been tested by the Amos developers. For this thesis *MySQL 5.0* and the corresponding driver *MySQL Connector/J 5.1* have been used.

The configuration of the JDBC data resource is done in two steps:

1. Create an instance of the *jdbc* type that represents a wrapped relational JDBC data source in Amos II. The constructor for this type demands a name for the data source and the JDBC driver class (e.g. *com.mysql.jdbc.Driver*). This name is further referred to as *relational name*.
2. Connect to the database using the *connect* function that takes the relational name, a JBBC connection URL (e.g. *jdbc:mysql://localhost:3306/worldcup*) and the username and password as parameters.

Now the wrapped data source can be accessed via its relational name.

5.4.2 AmosGetAvailableTables-Activity

Brief Description

The brief description is entered within the Table 5.3.

Summary	This activity retrieves the table names of relational database wrapped in a Amos data resource.
Activity name	*at.ac.univie.AmosGetAvailableTables*
Server class	*at.ac.univie.activity.AmosGetAvailableTablesActivity*
Client toolkit class	*at.ac.univie.activity.client.AmosGetAvailableTables*
Inputs	*relational_name*: type *java.lang.String* required. Name of the wrapped data source.
Outputs	*data*: list of table names as *java.lang.String*s.
Configuration parameters	none
Activity input and output ordering	none
Activity contracts	none
Behaviour	Executes a AmosQL query to retrieve the table names.

Table 5.3: Brief description AmosGetAvailableTables-Activity

Description

Amos II as shown in the motivation above provides types and functions for wrapped data sources. Beside the shown connection functionality separate functions are provided to access meta-data of wrapped data sources. This activity uses one of this functions and therefore executes AmosQL query to retrieve the table names. The function *tables(Relational r)* returns a list of tuples where each tuple contains a table name. Then these names are written as list output by the activity.

Usage example

To show the usage of this activity a MySQL database is configured as wrapped data source in a Amos II peer. The database contains the persons, players and referees as shown in the worldcup example of the Amos II Tutorial (see [FR08]). The extended entity-relationship diagram for this example can be seen in Figure 5.3 on page 57. Only two activities are required to build the workflow pipeline to retrieve the table names.

```
AmosGetAvailableTables query = new AmosGetAvailableTables();
query.setResourceID(new ResourceID("AmosResource"));
query.addRelationalName("WORLDCUPDB");
DeliverToRequestStatus deliver = new DeliverToRequestStatus();
deliver.connectInput(query.getDataOutput());
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(deliver);
```

The AmosGetAvailableTables client activity needs a configured Amos data resource (*AmosResource*) and the name of a wrapped relational data source (*WORLDCUPDB*). The DeliverToRequestStatus takes care of making the result available to the client application.

It uses the AmosGetAvailableTables activity output as input and therefore in workflow the DeliverToRequestStatus activity is placed after the AmosGetAvailableTables activity. After executing the workflow the result is available in the request status and contains the three table names of the simplified worldcup database.

Class-Diagram

The class diagrams describing the implementation of the AmosGetAvailableTables-Server-Activity can be found in Figure 5.10 and the one for the client in Figure 5.11.

Figure 5.10: Server activity class diagram of AmosGetAvailableTables-Activity

For the server activity (see Figure 5.10) *AmosGetAvailableTablesActivity* is the main activity class directly used and configured in the OGSA-DAI framework. The AmosQL query to fetch the table names is executed by *AmosRelationalTablesRetriever* for better testability.

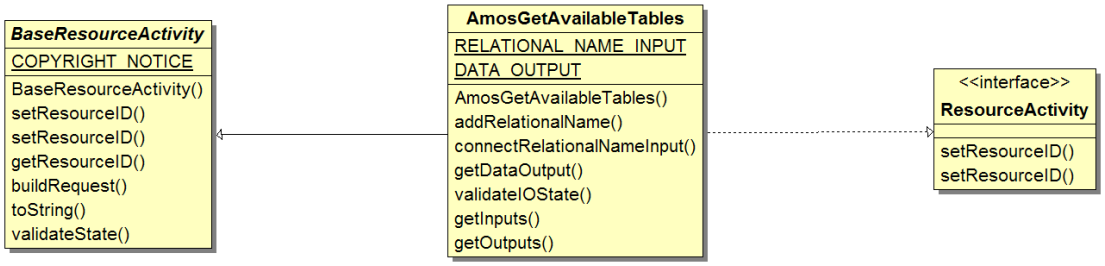

Figure 5.11: Client activity class diagram of AmosGetAvailableTables-Activity

As shown in the client class diagram (see Figure 5.11) the class *AmosGetAvailableTables* can be used to retrieve the table names. By setting the wrapped data source via *addRelationalName* *getDataOutput* delivers the table names.

5.4.3 AmosExtractTableSchema-Activity

Brief Description

The Table 5.4 shows the brief description of the *AmosExtractTableSchema*-Activity.

Summary	This activity retrieves the meta-data for a table in a relational database wrapped in a Amos data resource.
Activity name	*at.ac.univie.AmosExtractTableSchema*
Server class	*at.ac.univie.activity.AmosExtractTableSchemaActivity*
Client toolkit class	*at.ac.univie.activity.client.AmosExtractTableSchema*
Inputs	*relational_name*: type *java.lang.String* required. Name of the wrapped data source. *table_name*: type *java.lang.String* required. Name of the table.
Outputs	*data*: list of *TableMetaData*.
Configuration parameters	none
Activity input and output ordering	none
Activity contracts	none
Behaviour	Executes AmosQL queries to retrieve the meta-data for a table.

Table 5.4: Brief description AmosExtractTableSchema-Activity

Description

This activity should return the same output when targeted on a given table in a relational database as the *ExtractTableSchema*-Activity provided by OGSA-DAI. Since the OGSA-DAI uses JDBC in its activity the new Amos activity attempts to reach the same functionality. As shown in the previous activity description Amos II provides functions to retrieve meta-data of a wrapped relational data source. These functions should retrieve the meta-data like the JDBC methods called by the OGSA-DAI activity. For the Amos activity the following functions are used (see [FHJ+]):

- *columns(Relational r, Charstring table_name)*: retrieve column names for a table.
- *primary_keys(Relational r, Charstring table_name)*: retrieve primary keys.
- *imported_keys(Jdbc j, Charstring fktable)*: retrieve foreign keys.
- *exported_keys(Jdbc j, Charstring pktable)*: retrieve foreign keys of other tables referencing this table.

To retrieve the meta-data for one table these four functions are called via AmosQL statements. All information retrieved by the functions can be easily used to build the meta-data object that this activity returns. For the conversion to XML the classes with the same functionality provided by OGSA-DAI have been only slightly modified. It would have been possible to reuse the OGSA-DAI implementation but for the following use-cases fine-grained performance measurement functionality was needed and therefore new classes were implemented.
If you are familiar with the meta-data returned by the OGSA-DAI activity you would have

probably noticed that the Amos functions mentioned above do not cover all meta-data required. The missing meta-data and possible options to gain the missing information are discussed in the part dealing with restrictions of this activity functionality.

Usage example

The previous AmosGetAvailableTables-Activity can be performed to receive the table names of the relational data source. A second step is to retrieve the meta-data of one of these tables. Of course the prerequisites for the previous activity apply also for this activity. Again only two activities are required to build the workflow pipeline to retrieve the meta-data of one table.

```
AmosExtractTableSchema query = new AmosExtractTableSchema();
query.setResourceID(new ResourceID("AmosResource"));
query.addRelationalName("WORLDCUPDB");
query.addTableName("person");
DeliverToRequestStatus deliver = new DeliverToRequestStatus();
deliver.connectInput(query.getDataOutput());
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(deliver);
```

The AmosExtractTableSchema client activity needs a configured Amos data resource (*AmosResource*), the name of a wrapped relational data source (*WORLDCUPDB*) and the name of the table *person* for which the meta-data should be retrieved. The DeliverToRequestStatus allows the client to access resulting meta-data. It uses the AmosExtractTableSchema activity output as input and therefore in the workflow the DeliverToRequestStatus activity is placed after the AmosGetAvailableTables activity. After executing the workflow the XML representation of the meta-data is available in the request status.

Class-Diagram

The class diagrams describing the implementation of the AmosExtractTableSchema-Server-Activity can be found in Figure 5.12 and the one for the client in Figure 5.13.

For the server activity (see Figure 5.12) *AmosExtractTableSchemaActivity* is the main activity class directly used and configured in the OGSA-DAI framework. *AmosRelationalSchemaRetriever* contains the core functionality to retrieve the meta-data. The existing implementations for the OGSA-DAI interfaces *KeyMetaData*, *ColumnMetaData* and *TableMetaData* are to some extent optimized for the JDBC retrieved meta-data. Therefore the interface implementations *KeyMetaDataImpl*, *ColumnMetaDataImpl* and *AmosTableMetaData* are provided.

As shown in the client class diagram (see Figure 5.13) the class *AmosExtractTableSchema* can be used to retrieve the meta-data. By setting the wrapped data source via *addRelationalName* and the table via *addTableName*, *getDataOutput* delivers the meta-data.

Restrictions

As mentioned in the description of this activity not all meta-data information can be provided by the four Amos functions used. Beside the *catalog name* and the *schema name*

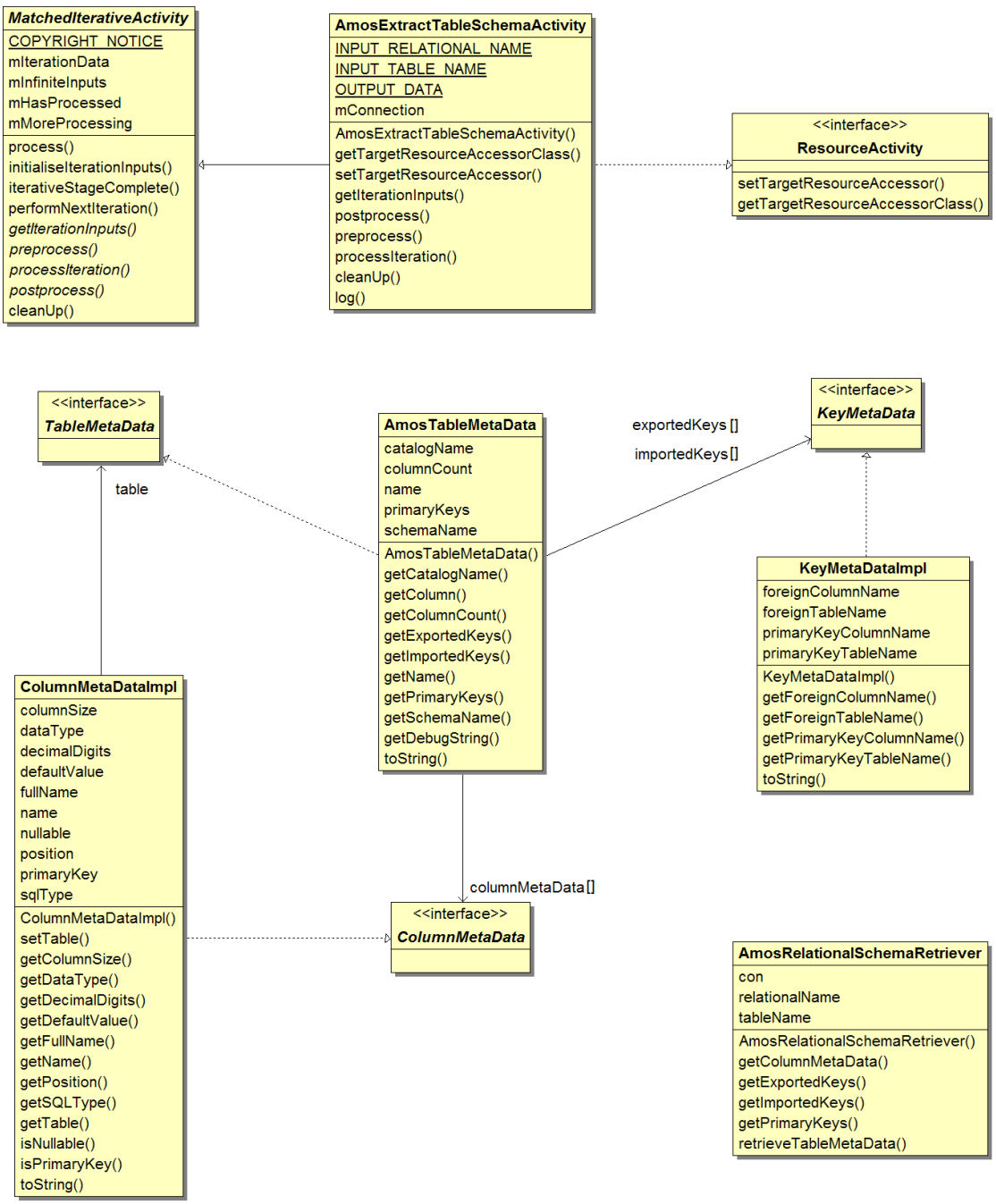

Figure 5.12: Server activity class diagram of AmosExtractTableSchema-Activity

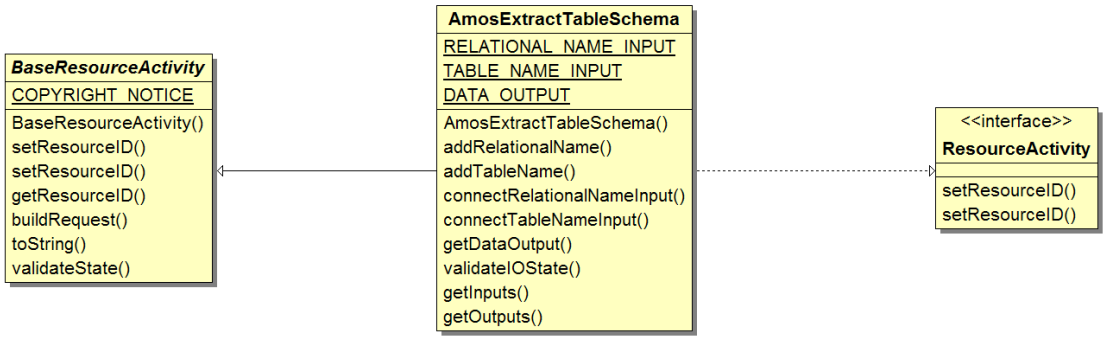

Figure 5.13: Client activity class diagram of AmosExtractTableSchema-Activity

some column meta-data is missing:

- column size
- decimal digits
- default value
- sql type

To overcome this deficiency, database vendor specific SQL queries could be executed that retrieve the missing meta-data. If the activity should behave in this way, an optional input parameter is necessary. This parameter must identify a set of SQL queries capable to retrieve the meta-data. Of course this would further require specific query logic for each supported database vendor/version. Purpose would be the execution of SQL queries and the mapping of the results to the missing meta-data.

5.4.4 AmosSQLQuery-Activity

Brief Description

In Table 5.5 the overview of the short summarized activity can be found.

Description

This activity tries to provide the same functionality as the OGSA-DAI activity *SQLQuery*. Whereas the functionality is provided by JDBC calls in the OGSA-DAI activity the Amos II function *sql(Relational r, Charstring query)* described in the manual (see [FHJ+]) takes care of this task in the Amos counterpart. This function is called via an AmosQL query using the embedded-query interface. A quick look at the brief description table shows that meta-data describing the output relation is required too. Since the Amos function that executes the SQL query does not return any kind of meta-data only a very limited form of meta-data is provided in the result. Since the information for the meta-data is taken from the result tuples meta-data is only provided if the SQL query returns at least one tuple. A more detailed description can be found in the restrictions chapter of this activity.

Usage example

The previous AmosGetAvailableTables and AmosExtractTableSchema activities can be used to retrieve meta-data helpful to build an SQL query used in this activity. Of course the prerequisites for the previous activity apply also for this activity.

Summary	This activity executes a SQL query on a relational database wrapped in a Amos data resource.
Activity name	*at.ac.univie.AmosSQLQuery*
Server class	*at.ac.univie.activity.AmosSQLQueryActivity*
Client toolkit class	*at.ac.univie.activity.client.AmosSQLQuery*
Inputs	*relational_name*: type *java.lang.String* required. Name of the wrapped data source. *expression*: type *java.lang.String* required. SQL query.
Outputs	*data*: list of *Tuple*s and describing *TupleMetadata*.
Configuration parameters	none
Activity input and output ordering	none
Activity contracts	none
Behaviour	Executes a AmosQL query containing a SQL query to retrieve tuples and meta-data.

Table 5.5: Brief description AmosSQLQuery-Activity

```
AmosSQLQuery query = new AmosSQLQuery();
query.setResourceID(new ResourceID("AmosResource"));
query.addExpression("select * from person;");
query.addRelationalName("WORLDCUPDB");
DeliverToRequestStatus deliver = new DeliverToRequestStatus();
TupleToWebRowSetCharArrays toWebRowSet = new TupleToWebRowSetCharArrays();
toWebRowSet.connectDataInput(query.getDataOutput());
deliver.connectInput(toWebRowSet.getResultOutput());
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(toWebRowSet);
pipeline.add(deliver);
```

The AmosSQLQuery client activity needs a configured Amos data resource (*AmosResource*), the name of a wrapped relational data source (*WORLDCUPDB*) and of course the SQL query itself *select * from person;*. Beside the often mentioned *DeliverToRequestStatus* activity the *TupleToWebRowSetCharArrays* activity transforms tuples to an XML representation. These dependencies are reflected in the workflow. The XML result is available in the request status after the execution.

Class-Diagram

For the server activity (see Figure 5.14), *AmosSQLQueryActivity* is the main activity class directly used and configured in the OGSA-DAI framework. The AmosQL/SQL query is executed by *AmosSQLQueryExecutor* to allow testing the functionality without depending on OGSA-DAI.

The client class diagram (see Figure 5.15) shows that the class *AmosSQLQuery* can be used to execute an SQL query against a wrapped data source. By setting the wrapped data source via *addRelationalName* and the SQL query via *addExpression* the tuples and

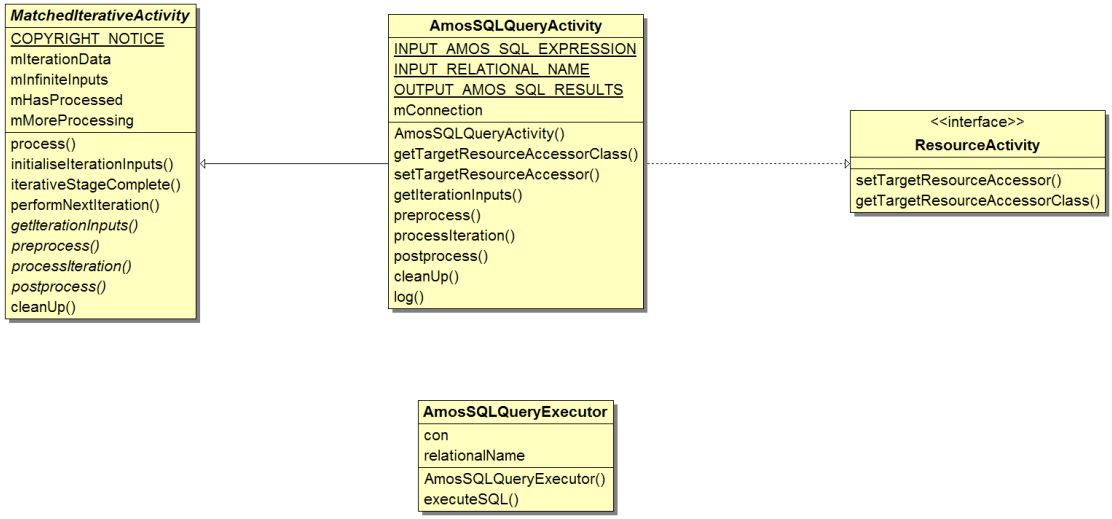

Figure 5.14: Server activity class diagram of AmosSQLQuery-Activity

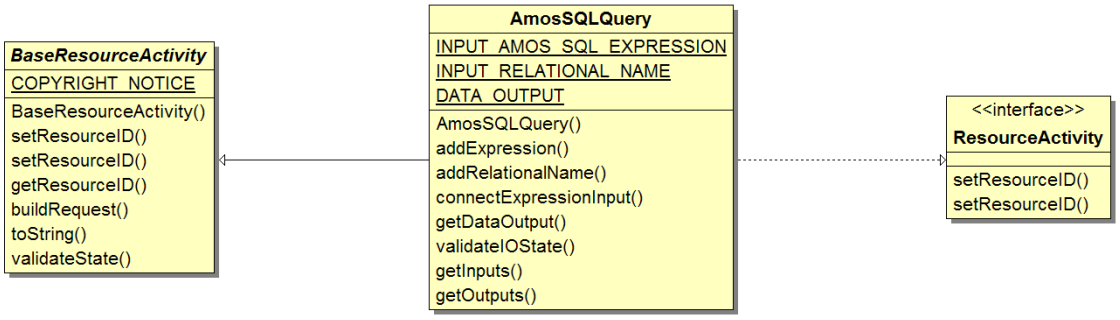

Figure 5.15: Client activity class diagram of AmosSQLQuery-Activity

the corresponding meta-data are returned.

Restrictions

For each column in the result relation a *uk.org.ogsadai.tuple.ColumnMetadata* describing this column should be available in the output of the activity. Due to the above mentioned lack of meta-data provided by the Amos function *sql* the following parts of the interface are not set with the correct values.

- name: column name.
- type: column type as defined in *uk.org.ogsadai.tuple.TupleTypes*.
- precision: number of decimal digits in the column.
- nullable: whether null values are allowed, not allowed or unknown.
- column display size: maximal length of column.

Currently default values are used instead of the correct values. E.g., *column_1*, *column_2...column_n* is used for the name if the result is not empty.

Chapter 6

Use Cases and Performance

6.1 Use Case Amos Schema Retrieval

6.1.1 Motivation

As shown in the AmosSchema activity description (see Section 5.3.2) meta-data about types and functions can be retrieved. To discuss the usability of this activity a more complex Amos schema is retrieved. This should allow to estimate the provided information in the resulting XML document (see Figure 6.1). The Figure 6.1 shows the three different parts involved: the client, OGSA-DAI and the Amos II Peer. The client sends a AmosQL query to OGSA-DAI to retrieve the schema saved within the Amos II Peer. The schema saved in the Amos II peer can be found in Figure 6.2. In the base case a client unaware of the types and functions stored in a schema should be able to formulate AmosQL queries after executing the AmosSchema activity.

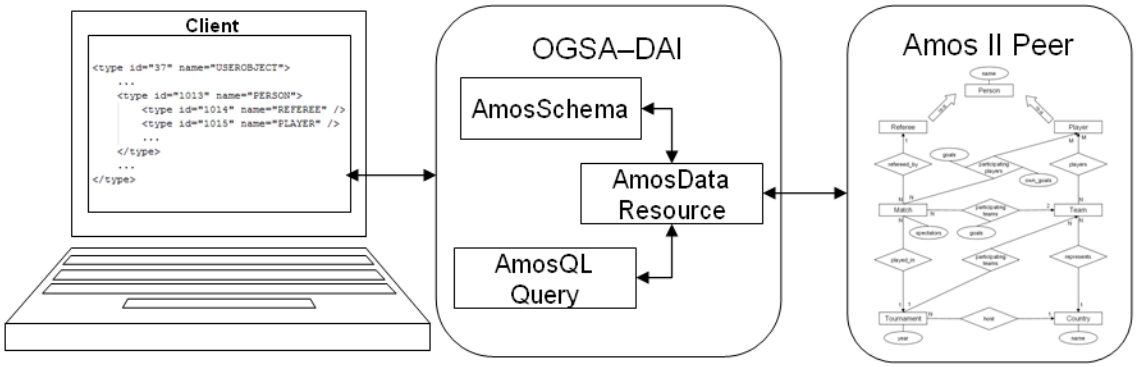

Figure 6.1: Schema retrieval using the AmosSchema activity

6.1.2 Test Schema

Figure 6.2 shows the Extended ER schema used in the Amos II tutorial [FR08]. The meta-data of a small part of this schema is discussed in Section 5.3.2. The reasons for choosing this schema are:

- The tutorial schema is a reference schema provided by acknowledged Amos II experts.

- It shows the most important features of the mediator system and each feature is discussed on the basis of an example of the concrete schema.

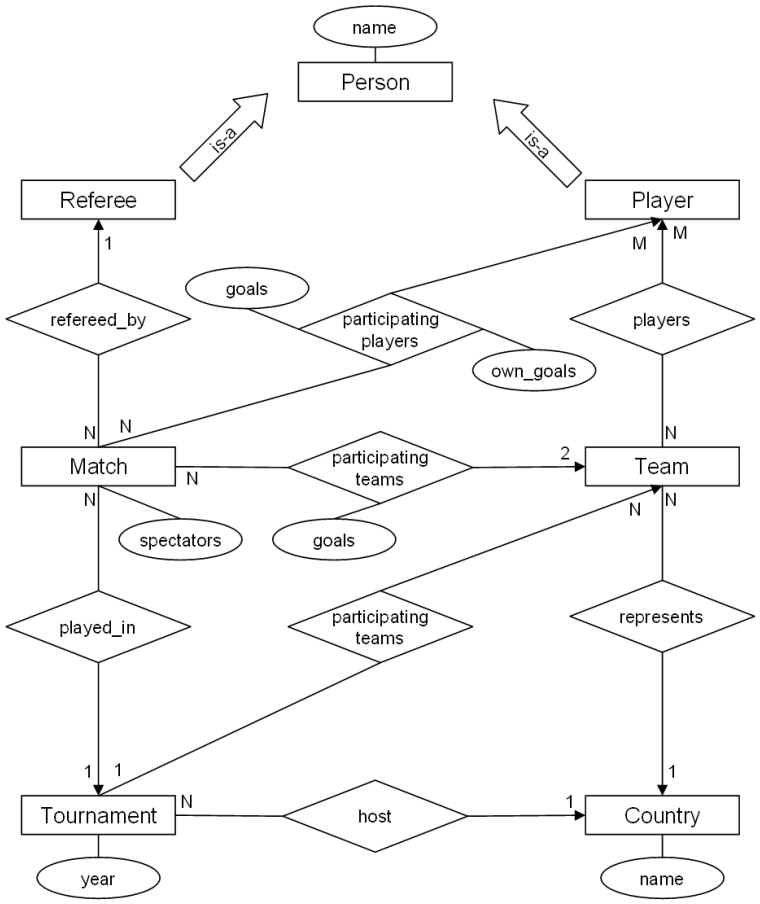

Figure 6.2: Extended ER schema of Amos tutorial example [FR08]

6.1.3 Results

The resulting XML representation for the schema.

```
<type id="37" name="USEROBJECT">
  <type id="999" name="COUNTRY">
    <attribute id="74" name="NAME" kind="OVERLOADED" />
    <method id="1002" name="COUNTRY.NAME->CHARSTRING" kind="STORED" />
  </type>
  <type id="1023" name="TEAM">
    <attribute id="1027" name="REPRESENTS" kind="GENERIC" />
    <attribute id="1044" name="PLAYERS" kind="GENERIC" />
    <method id="1028" name="TEAM.REPRESENTS->COUNTRY" kind="STORED" />
    <method id="1045" name="TEAM.PLAYERS->PLAYER" kind="STORED" />
  </type>
  <type id="998" name="TOURNAMENT">
    <attribute id="1004" name="HOST" kind="GENERIC" />
    <attribute id="457" name="YEAR" kind="OVERLOADED" />
```

```
    <attribute id="1039" name="PARTICIPATING_TEAMS" kind="OVERLOADED" />
    <method id="1040" name="TOURNAMENT.PARTICIPATING_TEAMS->TEAM"
      kind="STORED" />
    <method id="1000" name="TOURNAMENT.YEAR->INTEGER" kind="STORED" />
    <method id="1005" name="TOURNAMENT.HOST->COUNTRY" kind="STORED" />
  </type>
  <type id="1013" name="PERSON">
    <attribute id="74" name="NAME" kind="OVERLOADED" />
    <method id="1016" name="PERSON.NAME->CHARSTRING" kind="STORED" />
    <type id="1014" name="REFEREE" />
    <type id="1015" name="PLAYER" />
  </type>
  <type id="1024" name="MATCH">
    <attribute id="1033" name="PLAYED_IN" kind="GENERIC" />
    <attribute id="1050" name="PARTICIPATING_PLAYERS" kind="GENERIC" />
    <attribute id="1039" name="PARTICIPATING_TEAMS" kind="OVERLOADED" />
    <attribute id="1030" name="REFEREED_BY" kind="GENERIC" />
    <attribute id="1052" name="GOALS" kind="OVERLOADED" />
    <attribute id="1228" name="GOALS2" kind="GENERIC" />
    <attribute id="1036" name="SPECTATORS" kind="GENERIC" />
    <attribute id="1226" name="GOALS1" kind="GENERIC" />
    <method id="1037" name="MATCH.SPECTATORS->INTEGER" kind="STORED" />
    <method id="1060" name="MATCH.PARTICIPATING_TEAMS->TEAM" kind="DERIVED" />
    <method id="1229" name="MATCH.GOALS2->INTEGER" kind="DERIVED" />
    <method id="1227" name="MATCH.GOALS1->INTEGER" kind="DERIVED" />
    <method id="1051" name="MATCH.PARTICIPATING_PLAYERS->PLAYER"
      kind="DERIVED" />
    <method id="1230" name="MATCH.GOALS->INTEGER" kind="DERIVED" />
    <method id="1034" name="MATCH.PLAYED_IN->TOURNAMENT" kind="STORED" />
    <method id="1031" name="MATCH.REFEREED_BY->REFEREE" kind="STORED" />
  </type>
</type>
```

Using the diagram from Figure 6.2 and the retrieved XML the following paragraphs estimate the usefulness of the AmosSchema activity to handle the core concepts described in the tutorial (see [FR08]).

Types

As already shown in the usage example of the activity description the XML representation uses *type* elements and their nesting to show the type hierarchy. The attributes of the XML element *type* define the Amos object identifier and the type name. For example *REFEREE* and *PLAYER* are subtypes of *PERSON*.

Functions

A stored function has the semantic of an attribute to a given type and therefore it is represented as *attribute* element in XML. The attributes of the XML element define the Amos object identifier and the function name. The *kind* attribute has either the value *OVERLOADED* for attribute/function names that occur more than once or *GENERIC* for unique attribute/function names. Further information to a stored function can be

found in the method that has a name consisting of the describing type, the function name and the stored object type. For example *PERSON* has the attribute name described by the XML element *attribute id="74"...* and *method id="1016"*.

Derived functions are similar to stored functions. The only difference in the XML representation is the value of the *kind* attribute which has either the values *DERIVED* or *OVERLOADED* depending on the uniqueness of the derived function name.

6.1.4 Conclusion

The current implementation allows the retrieval of the most important schema information from an Amos II peer. This information describes types and functions available for these types. With this information a client developer can design AmosQL queries to fetch desired data. Some advanced concepts (see [FR08]) are currently not or not sufficiently considered.

- *Inverse valued functions*: Not considered in the XML representation.
- *Set valued functions*: The cardinality of the return value is not represented in XML.
- *Tuple returning functions*: Tuple property of the return value is not represented in XML.

A further improvement to the AmosSchema activity could be a more compact XML representation of the schema. For example a attribute of a type is currently shown as *attribute* and *method* element of the owning *type* element. A reduction to a single XML element containing all information of the attribute would increase the comprehensibility.

6.2 Use Case Mediation

6.2.1 Motivation

The main purpose of this use case is to show the mediation possibilities of Amos II. As mentioned earlier, it is possible to access different data sources with the help of Amos II. We tried to figure out how to access multiple databases and how to create mediated queries against such databases. The databases in our case should be different concerning schema and also number of rows. Different test AmosQL queries were started to retrieve the results from the different databases. From a Amos II peer, it is possible to retrieve results from each database wrapped by the peer or to retrieve a mediated result from more than one database with the help of a *derived type*. In our example we have two different databases and over these two databases we will start OGSA-DAI queries to show the mediation process.

6.2.2 System overview

Figure 6.3 should give a short overview of our system where the mediation possibilities are shown. As described in Section 2.2.2 on page 10, the *persons* information is vertically partitioned. The join keys are in this case the *ssn* and *id*.

The data used in the first database is listed in Table 6.1 and the data of the second database in Table 6.2.

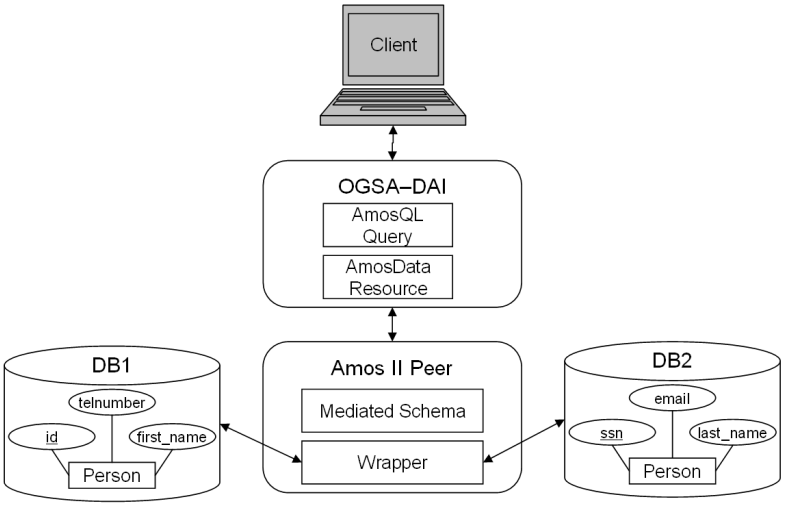

Figure 6.3: System overview for mediation

ssn	first_name	telnumber
1	Aaron	0800Aaron
2	Bertha	0800Bertha
3	Chris	0800Chris

Table 6.1: Data stored in DB1

6.2.3 Mediation

As the mediation is performed by the Amos II peer itself and not by one of the newly implemented activities it must be configured in the client before the AmosQLQuery activity can fetch mediated results. For the example configuration mentioned above the following AmosQL statements must be executed on the peer:

```
// DB1
jdbc('db1','com.mysql.jdbc.Driver');
connect(relational_named('DB1'),
  'jdbc:mysql://localhost:3306/db1', 'user', 'password');
import_table(relational_named("DB1"),"PERSON");
// DB2
jdbc('db2','com.mysql.jdbc.Driver');
connect(relational_named('DB2'),
  'jdbc:mysql://localhost:3306/db2', 'user', 'password');
import_table(relational_named("DB2"),"PERSON");
// Mediation
create derived type MediatedPerson
  subtype of PERSON_DB1 db1, PERSON_DB2 db2
  where ssn(db1) = id(db2);
create function full_name(MediatedPerson p)
  ->charstring as select first_name(p) + ' ' + last_name(p);
```

A detailed description of the used AmosQL statements can be found in [FHJ+]. *jdbc* and *connect* provide a connection to a JDBC database. *import_table* maps the relational model of the database to Amos types and functions. For example the *person* table from database

id	last_name	email
1	Adams	a.a@mail.com
2	Berkins	b.b@mail.com
3	Carter	c.c@mail.com
4	Doe	j.d@mail.com

Table 6.2: Data stored in DB2

DB1 is accessible as type *PERSON_DB1* and the attribute *first_name* is accessible using a function with the same name. This mapping is also known as view.
The mediation step itself is performed in two steps. First a derived type *MediatedPerson* is created that joins the two views on the relational databases. Afterwards a function *full_name* merges the first and last name distributed in the databases.

6.2.4 Retrieval

For the retrieval only a trivial AmosQL query has to be executed using a workflow as described in the corresponding activity description. This query resolves the heterogeneity between the two data sources.

```
select id(p),full_name(p),telnumber(p),email(p) from MediatedPerson p;
```

The result of the query for the concrete data mentioned above is shown in Table 6.3. The

full_name	telnumber	email
Aaron Adams	0800Aaron	a.a@mail.com
Bertha Berkins	0800Bertha	b.b@mail.com
Chris Carter	0800Chris	c.c@mail.com

Table 6.3: Result of mediation query

results show that the mediation meets the expectations. The person with the id 4 from DB2 is excluded due to the derived type condition and the full name is constructed as desired. As the mediation is done by the peer limitations can only arise from missing functionality in Amos II.
The following Sections show a usage example of the AmosQL-Activity and compare its performance with other possible mediation solutions.

6.2.5 Compared solutions

The AmosQL-Activity used in a OGSA-DAI workflow is compared for the following performance measurements (see Section 6.2.6) to three other mediation options. In the AmosQL-Activity workflow the Amos II peer retrieves data from the data sources and performs the mediation. The already mediated result is transformed in the workflow first to a tuple representation and afterwards to a XML *WebRowSet*. The Amos II peer configuration used for the mediation of four data sources as described in Section 6.2.6 can be found in Section D.1.1 and the OGSA-DAI workflow is listed in Section D.1.2.

Amos II Direct

As described above the AmosQL-Activity is integrated in a OGSA-DAI workflow and multiple steps are executed beside the execution of the AmosQL statement against the

Amos II peer. To estimate the overhead of these steps the execution time of a AmosQL statement using the Java API is measured. This AmosQL statement causes the Amos II peer to access the data sources, perform the mediation and return the results to the Java client application. No further processing of the mediated data is performed. The Amos II peer configuration used for the mediation of four data sources as described in Section 6.2.6 can be found in Section D.1.1. The configuration is identical to the AmosQL-Activity solution. The relevant parts of the Java client application is listed in D.1.3.

OGSA-DAI Mediation

OGSA-DAI provides the built-in *TupleMergeJoin* activity (see [AOD]) that allows joins and projections on two lists of ordered tuples. These tuples are retrieved by a single SQLQuery-Activity for each data resource necessary for the mediation. The merged tuples are afterwards transformed to XML using the *TupleToWebRowSet* activity and this XML can be transformed with an XSL document to achieve the desired mediation. Finally the resulting XML is propagated to the client.
For the system described in Figure 6.3 a inner join using the *id* and *ssn* columns would be appropriate. As only one of these columns is necessary for the result shown in Figure 6.3 either a projection or the XSL transformation could omit the no longer used column. The combination of the first and last name (mediation) is undertaken in the XSL transformation. The client workflow and the XSL document used to mediate four data sources as described in Section 6.2.6 are available in Section D.2.

Client-side Mediation

Mediation can also be achieved solely by a Java client application. The used implementation performs the following steps:

1. Object-relational mapping for the necessary databases with the help of the Java Persistence API (JPA) (see [BK06]) based on JDBC.
2. Inner join of the mapped results using a associative array.
3. Mediation using a formatted output of the object fields.

Therefore the whole process of retrieving, joining and mediating the data sources is performed by the client Java application.

6.2.6 Performance measurements

Test Procedure

For the performance tests the time between the submission of the query from the client and the availability of the mediated data to the client is measured.
When using the AmosQLQuery-Activity for the mediation a minimalistic workflow containing the AmosQLQuery-Activity and a TupleToWebRowSet-Activity is executed before the results are propagated to the client. The other solutions are used as described above in Section 6.2.5. For all compared solutions each test is repeated 10 times. This should avoid outliers as no dedicated server is used and therefore other processes could influence the results.

Test Data

Two test cases are considered where a mediation using two or four data sources is performed. For the test case using two relational data sources the schema described in Figure 6.3 is used. The *person* tables contain 32768 (2 to the power of 15) rows filled with random values except of course for the *id* and *ssn* as these columns are used for the join.
The second test case using a four data sources extends the 2 data source test case by adding two additional data sources described in Figure 6.4. *unique_number* and *social_number* are the join keys and both tables are filled with 32768 random value rows.

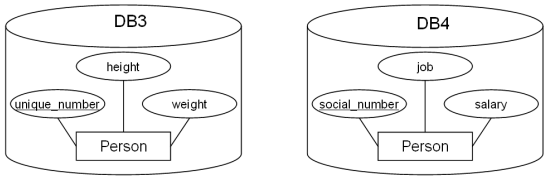

Figure 6.4: Additional Data Sources

Test Environment

The databases, AMOS II, OGSA-DAI and the client applications are all running on a common desktop computer.

Results

Table 6.4 shows the results achieved for the different solutions when two data sources are used. Each setting for each solution was tested multiple times and the best result was taken for the results table. Using the best results avoids influences of the test environment as no dedicated server is used and otherwise outliers due to other running processes on the test environment may change the average execution time. Table 6.5 shows the results when 4 data sources are involved in the mediation process.

Figure 6.5 shows a graphical representation of the performance results shown in Table 6.4. The relative perfomance of each solution compared to the slowest solution at a given setting is shown in Figure 6.6. The absolute and relative results for four data sources are shown in the Figures 6.7 and 6.8.

The interpretation for the retrieved results are:

- Retrieving the mediated data of a Amos II peer is for most settings almost as fast as accessing this data using a complete OGSA-DAI workflow with an AmosQLQuery-Activity. The difference in the execution time is due to the steps that are executed additionally to the direct access. These steps account for about 20% of the total workflow execution time. Among them is the conversion of the result set delivered by the Amos II Java API into OGSA-DAI tuple objects, the retrieval of the metadata and the conversion to XML with the help of the *TupleToWebRowSet* activity.
- The OGSA-DAI mediation has a relatively large overhead compared to the client-side mediation resulting in comparative poor performance when less data is accessed. The comparison of the results with 2 and 4 data sources and the same number of rows shows a good scalability at least for this number of data sources.

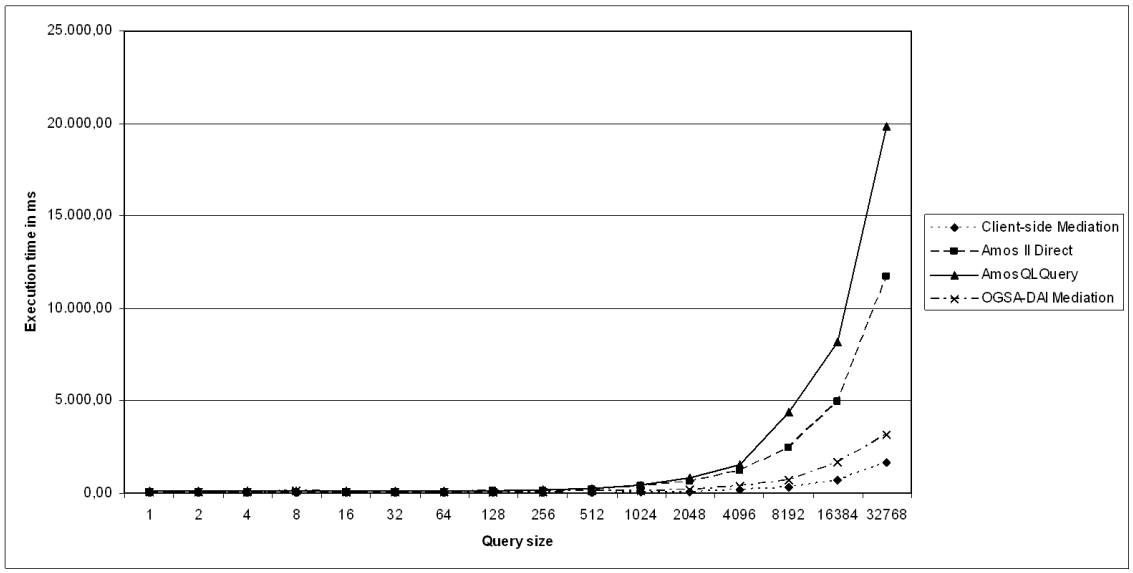

Figure 6.5: Absolute execution time for 2 data sources

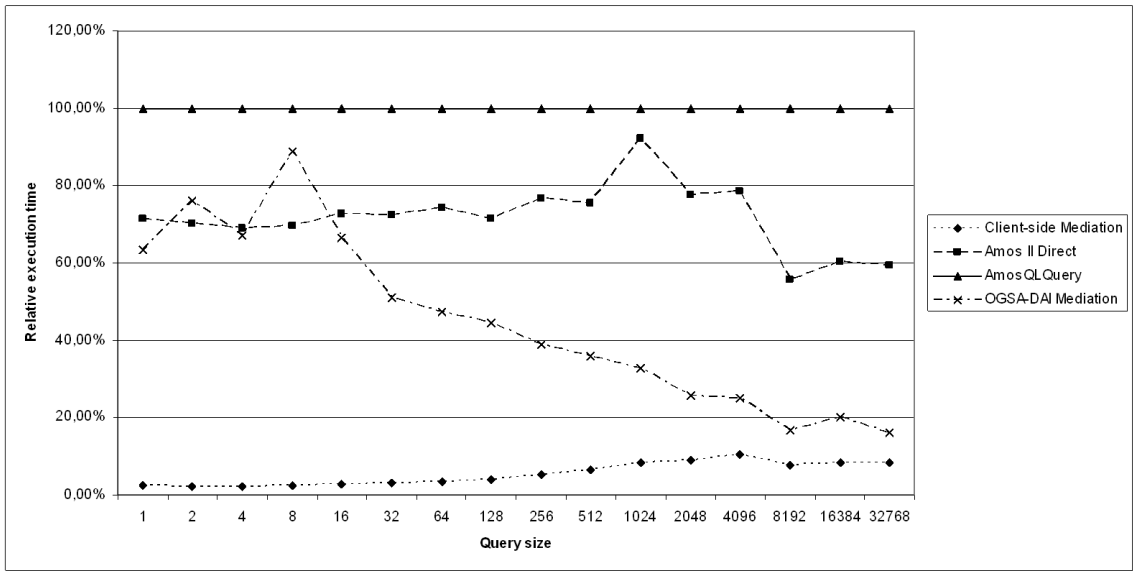

Figure 6.6: Relative execution time for 2 data sources

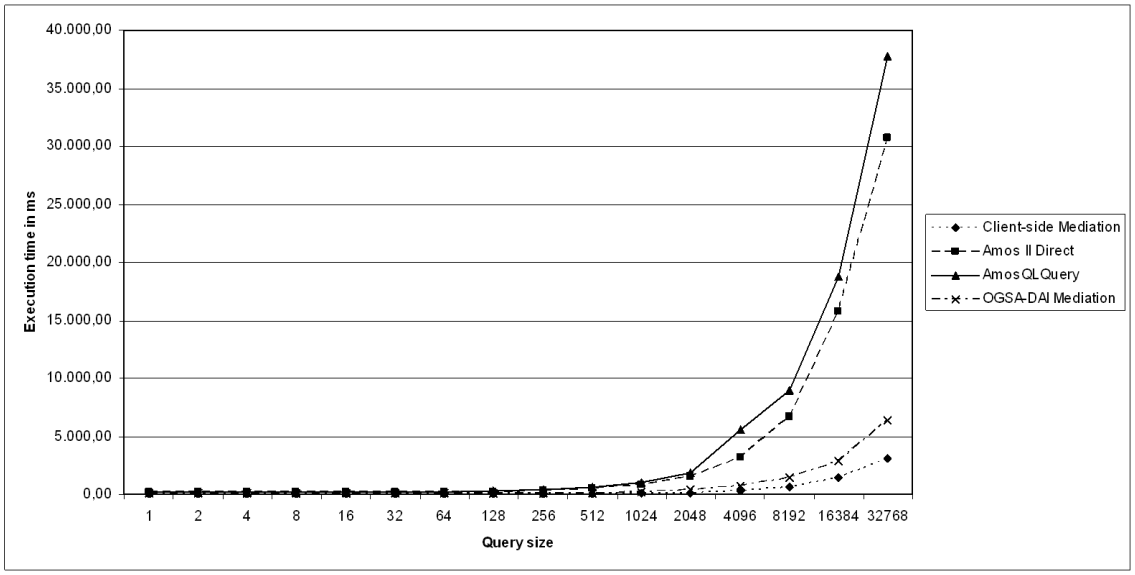

Figure 6.7: Absolute execution time for 4 data sources

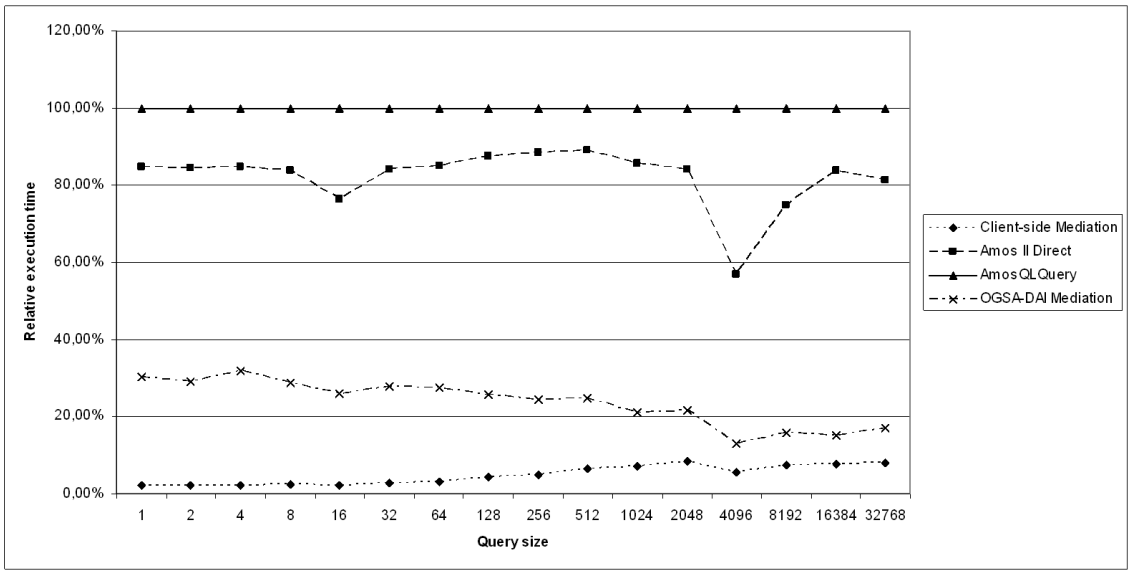

Figure 6.8: Relative execution time for 4 data sources

Rows	Client-side	Amos Direct	AmosQL	OGSA-DAI
1	2,57	74,29	104,11	66,15
2	2,46	74,81	106,75	81,37
4	2,50	75,53	109,35	73,39
8	2,60	75,79	108,86	96,70
16	2,91	79,89	109,72	72,96
32	3,37	82,30	113,76	58,12
64	4,23	90,84	122,44	57,80
128	6,31	108,65	152,40	67,86
256	10,19	144,77	188,56	73,75
512	18,56	218,63	289,32	103,46
1024	36,20	403,65	437,28	143,41
2048	74,77	644,14	828,64	212,07
4096	162,83	1.223,96	1.559,77	388,56
8192	338,81	2.436,10	4.382,17	728,31
16384	684,07	4.937,22	8.205,78	1.655,09
32768	1.663,15	11.747,54	19.828,90	3.164,71

Table 6.4: Execution time in ms for 2 data sources

- The client-side mediation using the JPA was fastest. A further analysis of the steps performed by the client application showed that the object-relational mapping containing the four queries against the data sources needs about 90% of the whole execution when 32768 rows are queried for each of the four data sources. The remaining 10% divide in 2% joining the four results with the common key and 8% for the mediation of the first and last name.

6.2.7 Conclusion

In the first part of this use case the mediation using an Amos II peer was illustrated. The suitability of Amos II for this task and how this mediation capability can be used via OGSA-DAI is demonstrated. The performance results show that the AmosQLQuery mediation is slower than other solutions for the given simple mediation case. As the comparison to the direct Amos II access shows this is mostly due to the used Java API interface and the default configuration of the Amos II peer used for the mediation. Beside the performance aspects implementing client applications was easiest for the solutions involving Amos II. Amos II client applications have just to adapt the AmosQL query when for example switching from two to four data sources. Furthermore client developers performing this step have not to be aware of the distribution aspects of the data sources. They are only confronted with the mediated schema if no other data retrieval is intended. In contrast the OGSA-DAI workflow needs significant changes if the number of data sources is altered. Using two data sources requires for example one *TupleMergeJoin*-Activity whereas four data sources require three *TupleMergeJoin*-Activities. This is due to the fact that this activity can only join exactly two tuple lists (see [AOD]). Furthermore the XSL stylesheet performing the last step of the mediation has to be adapted too. Even the effort on the server for adding new data sources is low as for example the combination of first name and last name has not to be changed.

Rows	Client-side	Amos Direct	AmosQL	OGSA-DAI
1	4,32	165,81	195,80	59,11
2	4,15	164,99	195,52	57,10
4	4,30	166,77	197,11	63,08
8	4,73	169,50	201,91	58,08
16	5,04	176,78	231,01	59,74
32	6,48	187,74	223,30	61,97
64	7,95	209,70	246,64	68,12
128	12,47	255,00	291,03	74,42
256	19,61	352,71	398,55	97,11
512	37,41	510,18	572,56	141,74
1024	71,41	867,27	1.012,67	211,91
2048	149,69	1.530,58	1.821,50	395,89
4096	311,60	3.189,69	5.591,01	721,84
8192	658,05	6.706,82	8.967,73	1.416,90
16384	1.452,24	15.723,42	18.787,28	2.870,59
32768	3.056,93	30.687,63	37.686,58	6.384,11

Table 6.5: Execution time in ms for 4 data sources

6.3 Use Case SQL

6.3.1 Motivation

As shown in Section 5.4 direct access to wrapped data source in a Amos II peer can be a useful feature. Several activities to directly access relational databases have been implemented. The main focus of the AmosGetAvailableTables-Activity and the Amos ExtractTableSchema-Activity is to show that the desired meta-data can be retrieved. The AmosSQLQuery-Activity must not only deliver correct result but also in a timely manner. This can mean low overhead for small result sets and scalability to handle large result sets.

6.3.2 Compared solutions

OGSA-DAI SQLQuery-Activity

To compare the performance of the AmosSQLQuery-Activity with SQLQuery-Activity packaged with OGSA-DAI is the most obvious choice as large parts of the computation effort are identical. To be more concrete, both approaches use a very similar OGSA-DAI workflow.

1. Query data resource
2. Convert to a *WebRowSet* XML representation
3. Optimize the XML representation
4. Delivery of the XML to the client

Only the first step in this workflows differs when comparing the activities.
On the one hand the data resource providing the low level access to the relational database is different. *JDBCDataResource* used by the SQLQuery-Activity provides a JDBC *Connection* instance using a standard implementation found in every textbook covering this

topic. Details to *AmosDataResource* can be found in Section 5.2 on page 53 but the main differences are the intermediary Amos II peer between the relation database and the data resource and the *callin.Connection* instance provided by the data resource that allows amongst others the required execution of AmosQL statements.
On the other hand the activities responsible for executing the queries and transforming the results into tuples so that the subsequent activity can process the output of these activities. The SQLQuery-Activity uses standard JDBC methods to transform the results and retrieve the required meta-data information. The AmosSQLQuery-Activity has much more limited options to retrieve the results and their accompanying meta-data (see [FHJ+]). But these options are sufficient to retrieve the results and limited meta-data information.

Java Persistence API

The Java Persistence API (JPA) described in [BK06] is a state-of-the-art object/relational mapping facility that simplifies the handling of relational data in Java applications. The reasons to compare this database access approach are:

- Java and thereby JDBC-based this allows better comparability due to the usage of the same JDBC driver as in the SQLQuery-Activity and AmosSQLQuery-Activity.
- While the other two approaches transform their native results into XML, JPA creates so called entity object instances. Thereby all three solutions transform their native relational data representation into a more common format.
- Similar usage complexity using a client-centric perception as the mapping to XML respectively Java objects is transparent to the client. In all three solutions the main factor influencing the processing time is the query to the database.

It must be noticed that JPA primary intention is not to use native SQL queries but the *Java Persistence Query Languge*. But in the end queries in this language are translated to SQL by the JPA as this is the only supported query language supported by the underlying JDBC and the relational database itself.

Amos II Direct

To understand how much of time used for AmosSQLQuery-Activity is due to the integration in a OGSA-DAI workflow the execution time for the direct access to Amos II is measured. This saves the effort for iterating over the Amos II Java API result set object to build the OGSA-DAI *Tuple* representation including the retrieval and mapping of the metadata and the transformation in a XML *WebRowSet*.

6.3.3 Performance measurements

Test Procedure

The performance tests are executed from a client-centric point of view. This means that the time is measured between the submission of the query from the client and the availability of the relational data to the client.
A description of the workflow executed for AmosSQLQuery-Activity can be found in Section 5.4.4. The only difference to the workflow presented there is a performance optimization recommended by the OGSA-DAI documentation [AOD]. The time measurement

is started on the client after all activities and the workflow are configured. Since the workflow is executed synchronously the first request status available to the client contains the relational data result. For the SQLQuery-Activity the workflow is almost identical, of course the query activity and the targeted data resource are not the same. The time measurement is done in the same way as for the AmosSQLQuery-Activity.
For the tests with the JPA, the start of the time measurement is after the entity manager and the persistence context (see [BK06]) are created. The measurement is stopped when the result data is available to the client, the entity manager and the persistence context are stopped afterwards.
For all solutions each test is repeated 10 times. This should avoid outliers.

Test Data

The test data is stored in a single table in a MySQL 5 database. The table named *person* contains 65536 (2 to the power of 16) rows where each row contains an id (1-65536), name, age and email using appropriate integer and character data types. The database was filled with random values for name, age and e-mail to avoid identical data rows that could lead to unexpected results. As id is the primary key and thereby an index of this table the processing for retrieving subsets of the available data is negligible when using the id as restriction parameter for the SQL query. The simplicity of the data should direct the attention of the performance measurements on the processing effort needed by the other parts in the processing chain. Furthermore this avoids the discussion of potential query optimizations. The concrete query for tests is:

```
select * from person where id < ?;
```

For the placeholder the values 1,2,4... 65536 are substituted.

Test Environment

The database, AMOS II, OGSA-DAI and the client applications are all running on a common desktop computer. As network performance restrictions between either of these components would have similar affects on all solutions, simulating such restrictions is not considered.

Results

Table 6.6 shows the results achieved for the four solutions. Each setting for each competitor was tested in multiple runs and the best performance is used for the results table. This procedure avoids influences of the test environment as no dedicated server is used and therefore other processes could influence single results and with it average results.

Figure 6.9 shows a graphical representation of the performance results shown in Table 6.6. As the absolute performance of the solutions is difficult to estimate at a glance a relative comparison presentation is performed in Figure 6.10. For each number of retrieved rows (query size) the execution time of the slowest solution is normalized to 100 percent.

The interpretation for the retrieved results are:

- Both the SQLQuery and the AmosSQLQuery bear the consequences of the overhead using the OGSA-DAI framework. This leads to dramatically worse performance compared to the JPA when applied to small result sets.

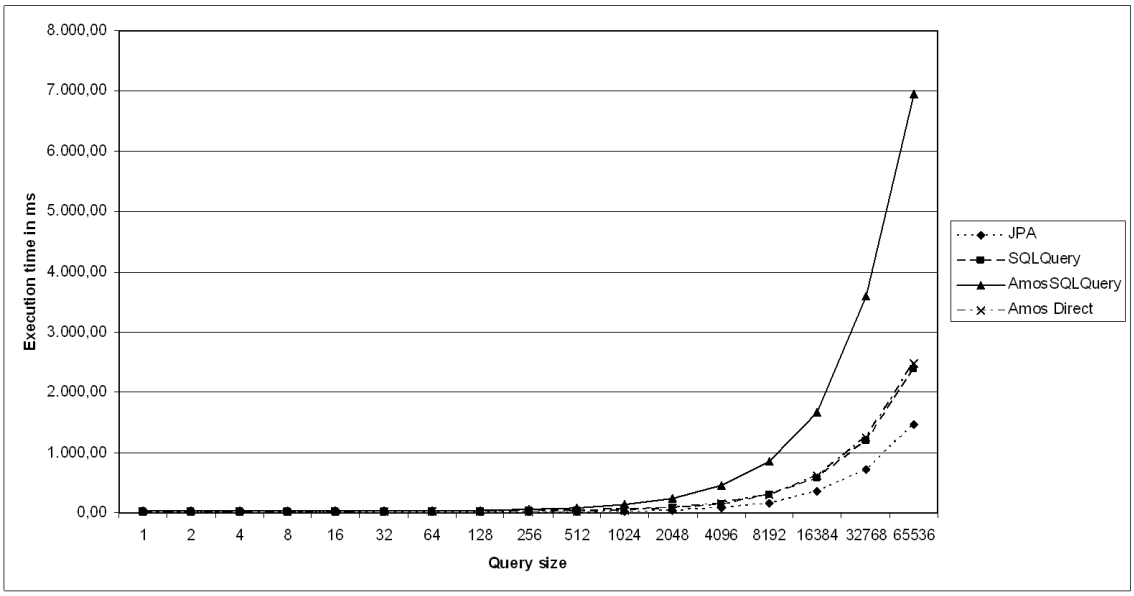

Figure 6.9: Absolute execution time

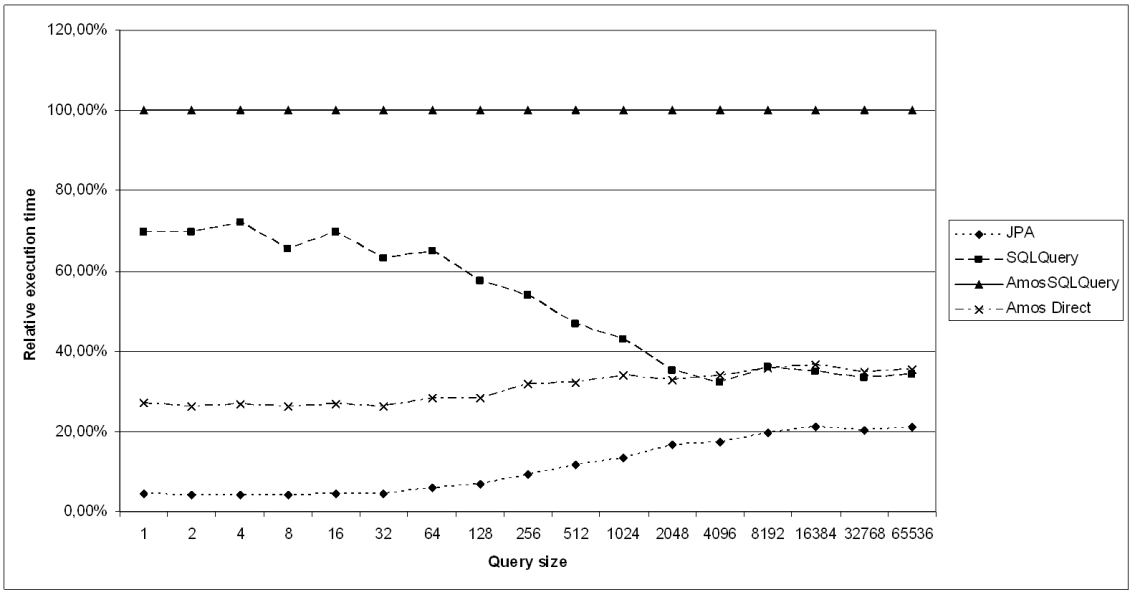

Figure 6.10: Relative execution time

Rows	JPA	SQLQuery	AmosSQLQuery	Amos Direct
1	1,53	23,93	34,31	9,25
2	1,39	24,06	34,54	9,02
4	1,39	24,74	34,31	9,24
8	1,48	23,43	35,77	9,34
16	1,60	24,83	35,58	9,54
32	1,81	25,12	39,78	10,37
64	2,34	25,86	39,87	11,25
128	3,33	27,37	47,73	13,53
256	5,38	31,01	57,67	18,35
512	9,66	39,17	83,71	27,02
1024	17,58	56,24	130,92	44,49
2048	41,00	86,11	245,00	80,19
4096	78,84	146,70	455,34	154,94
8192	166,17	305,86	847,43	302,69
16384	350,01	579,85	1.658,89	607,07
32768	721,04	1.193,50	3.583,18	1.245,05
65536	1.466,76	2.390,49	6.954,70	2.473,45

Table 6.6: Execution time in ms

- As the query size increases the marginal costs of retrieving one additional row are decreasing quickly for the AmosQLQuery and the SQLQuery. This correlation becomes linear with increasing row sizes. Nevertheless the marginal costs for the AmosSQLQuery remains higher than for the SQLQuery which has higher marginal costs than the JPA.
- The direct access to a Amos II peer wrapping a relational data source is slower for large query sizes than the SQLQuery workflow.

6.3.4 Conclusion

The proof-of-concept implementation AmosSQLQuery shows that this direct access activity to a wrapped data source in an Amos II peer allows the usage of SQL queries on a wrapped data source in an Amos II peer. Comparisons to other solutions identified necessary improvements when either low latency for small query sizes or low overhead for large query sizes are required. Not reflected in the numbers above is the fact that the missing meta-data retrieved by the AmosSQLQuery can be necessary for some usage scenarios. In these cases one more additional SQL queries would be necessary depending on database engine used. As this would lead only to a constant increase of the execution time independent of the query size the general estimation would not change. The results of the direct access to the Amos II peer show that significant improvements to the AmosSQLQuery workflow execution time are possible if the direct access is accelerated.

Possible improvement options to the existing AmosQLQuery implementation can be identified in the following areas.

- *Source code*: Optimizations using a profiler to determine which step of the activity processing is the most time-consuming beside the already identified comparatively slow direct access.

- *Amos interface*: Usage of the fast-path interface could reduce the overhead for small query sizes. According to [ER] this interface type is significantly faster than the embedded query interface. A part of this performance follow from the direct call to derived functions and stored procedures Therefore no declarative AmosQL query needs to be interpreted.
- *Amos configuration*: Optimizing the configuration of the Amos II peer so that it has enough memory available to handle large result sets.

Chapter 7

Conclusion and Future Work

This thesis gives an overview about mediation and grid computing solutions and their origin in distributed systems. The functionality of the solutions is described on the basis of simple examples. Furthermore projects where some of this solutions have been applied are presented. In the practical part the possibility to extend the grid solution OGSA-DAI with the mediation functionality of the mediator/wrapper solution Amos II is discussed. During the design and development of these extension for providing the functionality three main tasks have been accomplished:

- Providing a OGSA-DAI data resource that allows appropriate OGSA-DAI activities to access an Amos II peer.
- Schema and data retrieval of a possible mediated schema in an Amos II peer via appropriate activities. For the data retrieval the declarative query language AmosQL can be used.
- Schema and data retrieval of a wrapped relational data source in an Amos II peer via appropriate activities. These activities work almost identically to their OGSA-DAI counterparts allowing for example SQL as query language.

The resulting Java classes have been designed to be easily extensible and in accordance with the best practices suggested by OGSA-DAI documentation.

7.1 Lessons learned

Despite its great functionality OGSA-DAI proved to be comparatively easy to extend with new functionality. This is mostly due to the extensive documentation available (see [AOD]). Amos II on the other hand even though concise and mature in its concepts and implementation provided some obstacles to overcome before it was accessible from OGSA-DAI. The main reason is the programming language C used for Amos II. As OGSA-DAI is intended to be extended with Java classes the Java interface for Amos II was used. This interface can't hide its purpose as it relies heavily on JNI and making it therefore difficult to use in the Java framework OGSA-DAI.

Beside these more low-level aspects the mapping of different data representations was partly difficult. For example the data retrieval activity for a mediated schema has to output tuples using OGSA-DAI own data types so that the activity is composable with the existing OGSA-DAI transformation activities.

7.2 Software Documentation

The implementation and Javadoc documentation of the newly designed software can be found in *http://www.gridminer.org/amos2ogsadai*. To deploy and run the software additional third-party components are necessary. Installation instructions for OGSA-DAI 3.0 and binary and source downloads are available on [AOD]. Amos II is used in our thesis too and documentation and a binary release can be found on the website (see [amo]).

The main classes are also described in the class diagrams in the Chapter 5. There at least one class diagram is provided for the data resource and each activity. For the concrete steps necessary for the deployment of the data resource and the activities the instructions are available in the Appendix C.

7.3 Possible Improvements

Improvements to the current implementation are possible especially in two areas. On the one hand the current version often uses a straightforward approach where a more sophisticated approach could deliver better results in terms of functionality or performance. For example the declarative embedded query interface could be substituted by the fast path interface therefore allowing better performance. Or missing meta data when directly accessing a wrapped relational database could be fetched with vendor specific SQL queries. It would also be useful to revise the currently used AmosQL for retrieving meta data of schema.

On the other hand closely related functionality could be implemented. Whereas the current implementation allows to submit a fixed SQL query string to a wrapped relational database in an Amos II peer the possibility to use parameterized queries would be convenient.

This thesis discussed only mediation using relational databases whereas the many other existing Amos II wrappers are not considered. For example Internet search engines, XML data or ODBC databases.

7.4 Future work

Beside these obvious improvements new approaches could be explored. The following Sections suggest such possibilities that attracted interest during the development of this thesis.

7.4.1 Support for Streaming

Many of the built-in OGSA-DAI activities are designed for streaming and therefore it would provide a benefit if future data retrieval activities accessing an Amos II data resource would support this feature. Currently for example the AmosQLQuery-Activity fetches all data returned by a AmosQL query before the activity propagates any of these data to the next activity in the workflow. Drawbacks of this approach are delays increasing with the number of rows returned by the query and increased memory consumption as the whole object-oriented or XML representation is kept in the memory.

7.4.2 Mediation on the Fly

This thesis discusses the approach of accessing an Amos II peer where the mediation is already undertaken. Another approach could be to create an Amos II peer and its mediation configuration on the fly and use it to execute a query against existing OGSA-DAI data resources.

Figure 7.1 shows the rough architecture of this approach. The *Data Resource* represents one of the existing data resources built-in OGSA-DAI. Reusing existing data resources has the advantage that other activities or workflows can proceed as usual and are not affected if the data in the data resources is used for mediated queries too. The *Amos II Mediator* is created on demand and mediates over data delivered by one or more of these data resources. It is up for discussion which query language the client should use to perform mediated queries and how the mediator Amos II gets the needed information to target the queries required for mediation against the data resources.

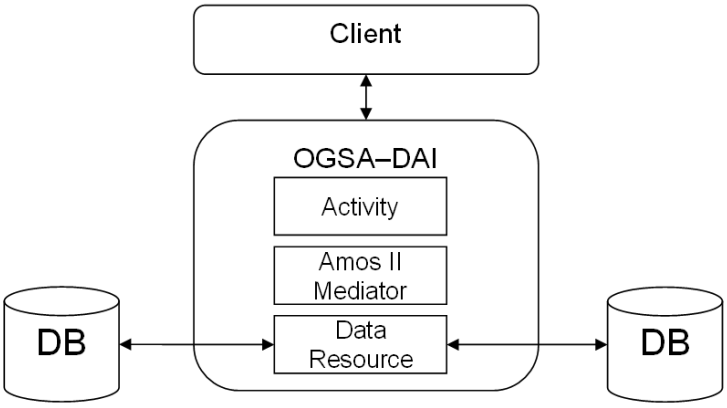

Figure 7.1: Ad hoc Mediation

7.4.3 Amos II Wrapper for OGSA-DAI

Beside adding the mediation functionality of Amos II to OGSA-DAI a inverse approach could attempt to make OGSA-DAI resources available in Amos II. Figure 7.2 shows a possible architecture for this approach.

The well-known data resources (*XMLDBCollectionProvider*, *JDBCConnectionProvider* and *FileAccessProvider*) provide the low level access to XML, relational and file data. These data resources are configured in OGSA-DAI middleware as usual and therefore not only the specific *WrapperActivity* shown in the Figure can access these data resources. As Amos II is not capable to directly access OGSA-DAI data resources the *WrapperActivity* hides many of the implementation details of the mentioned data resources and therefore simplifies the task for the *Wrapper* located in the Amos II peer. The task of this Amos II wrapper could be performed with the help of foreign functions (see [FHJ+]).

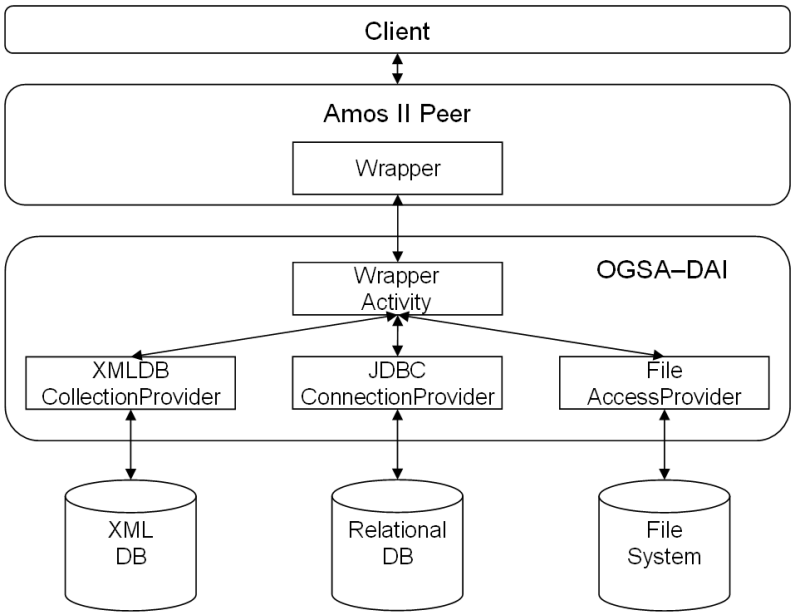

Figure 7.2: Amos II Wrapper for OGSA-DAI

Appendix A

Abstract in German

Die Speicherung der Daten und Informationen in den letzten Jahren war fast zu 100 Prozent immer in der Hand von zentralen Datenbanksystemen. Eine kleine Anzahl an Programmen und Experten waren involviert in der Handhabung dieser Datenquellen. Die Unternehmensstruktur hat sich verändert, früher kleine Unternehmen expandierten und viele ehemals kleine Unternehmen haben Bereiche die auf verschiedene Städte verteilt sind. Um eine bessere Handhabung der Daten zu gewährleisten, ist es keinesfalls mehr üblich, das Daten in einem Datenbanksystem gespeichert sind. Die unterschiedlichen Abteilungen die mit den gespeicherten Daten arbeiten, sind geographisch verteilt und eine größere Anzahl an Personen ist involviert innerhalb der Abteilungen die als *Database Department* oder auch spezialisierter als *Data Warehouse Department* oftmals bezeichnet werden.

Während früher Daten oftmals innerhalb von Files gespeichert wurden, ist es heute auch möglich strukturierte und semi-strukturierte Daten und Objekte zu speichern. Die bekanntesten und innerhalb dieser Arbeit von den involvierten Systemen zur Verfügung gestellten sind: Speicherung der Daten in Files, innerhalb von relationalen Datenbanken, XML Datenbanken und objekt-orientierten Datenbanken. Neben unterschiedlichen Produkten die Heterogenitäten auslösen können, können auch unterschiedliche Schemabeschreibungen innerhalb von gleichen kommerziellen und open-source Datenbanken verschiedene Probleme verursachen, die dann mit Hilfe von Mediation korrigiert werden müssen. Drei verschiedene Arten von Datenpartitionierung können genannt werden. Der Begriff der horizontalen Partitionierung inkludiert das Aufsplitten der gleichen Daten in mehrere Datenbanken. Diese Art der Partitionierung wird üblicherweise anhand eines Schlüssels erledigt. Vertikale Partitionierung erklärt den Ausdruck, dass zusammengehörige Daten in unterschiedlichen sogar verteilten Datenbanken abgespeichert sein können und mittels eines Schlüssels (keys) wieder kombiniert werden können. Partitionierungen können aber auch über heterogene Datenquellen erfolgen. Unter Heterogenität versteht man die Unterschiedlichkeit der beteiligten Datenquellen, dies könnte sein Format, Typ oder Datenbank. Um Abfragen über heterogene Datenquellen durchführen zu können, wurde bereits einiges an Forschungsarbeit investiert. Das Mapping der verschiedenen Datenquellen wird in der Informatik auch Mediation genannt.

Der Wrapper-Mediator Ansatz ist am weitesten verbreitet, um Abfragen gegen ein mediiertes Schema durchzuführen. Der Wrapper erfüllt den Zugang zu den Daten und sorgt dafür das Heterogenitäten verdeckt werden. Als nächsten Schritt ist der Wrapper auch zuständig für die Übersetzung und Durchführung von Abfragen. Der Mediator stellt

ein einheitliches Schema für die unterschiedlichen Datenquellen zur Verfügung und ist verantwortlich für die Umwandlung und Aufteilung der Query in kleinere Teil-Queries die mit Hilfe des Wrappers an die unterschiedlichen Datenquellen geschickt werden. Mediatoren können zentralisiert und verteilt sein. Handelt es sich um einen verteilten Mediator gibt es innerhalb von Amos II einen speziellen Mediator mit dem Namen *nameserver* und der ist zuständig für die Speicherung der Metadaten der Mediatoren. Amos II ist ein System, welches von uns verwendet wird um den Wrapper-Mediator Ansatz innerhalb von Gridsystemen zu zeigen.

Der Begriff des Gridcomputing umfasst in erster Linie die gemeinsame Nutzung von Computerresourcen. Loose coupling und der einfache Zugriff auf Resourcen innerhalb des Grid-Netzwerks sind die wichtigsten Punkte die erwähnt werden sollten. OGSA-DAI ist eine Middleware Lösung, die einen einheitlichen Zugriff mit Hilfe von Webservices auf Datenquellen innerhalb des Grids ermöglichen.

Der praktische Teil dieser Arbeit soll ein neues System sein, welches die zwei Ansätze miteinander kombiniert. Der Ansatz der Mediation innerhalb von OGSA-DAI ist gegeben durch OGSA-DQP aber hier sind nicht alle Bereiche abgedeckt, die Amos II zur Verfügung stellt. OGSA-DQP ermöglicht einen Zugriff auf relationale Datenquellen innerhalb von OGSA-DAI. Diese Arbeit erläutert die Erweitungsfähigkeit der zwei Systeme und soll die Vorteile der beiden Ansätze miteinander verknüpfen. Innerhalb dieser Arbeit werden beide Systeme vorgestellt und die Kombinationsfähigkeit dieser Systeme erläutert. Der praktische Teil dieser Arbeit beschäftigt sich mit der Erweiterung von OGSA-DAI mit Hilfe von neuen Aktivitäten und soll den Zugriff auf Amos II ermöglichen. Um die Aktivitäten an Amos II weiterleiten zu können, wird das Java call interface verwendet, welches dann die AmosQL Abfragen an die verschiedenen Amos II Peers schickt. Die Arbeit wird abgerundet durch verschiedene Performance-Analysen und Use-Cases die die Funktionalität der Implementierung zeigen sollen.

Appendix B

Lebenslauf

Persönliche Daten

Name:	Barbara Selista
Geburtsdatum:	06.08.1981
Geburtsort:	Wien, Austria
Adresse:	1110 Wien, Dopplergasse 10/41
Staatsangehörigkeit:	Österreich
Familienstand:	ledig
Tel:	0664/4525117
Email:	barbara.selista@gmx.at

Ausbildung

1987 - 1991:	Volksschule Molitorgasse, Wien
1991 - 1995:	Hauptschule Enkplatz, Wien
1995 - 2000:	Handelsakademie Marienanstalt, Wien
2000 - 2008:	Studium der Wirtschaftsinformatik Universität Wien *Schwerpunkte*: Electronic Commerce Public Utility Management *Diplomarbeitsthema*: Mediators in a distributed environment

Kenntnisse

Sprachen:	
Deutsch:	fließend
Serbisch:	Grundkenntnisse
Englisch:	in Wort und Schrift
Französisch:	Grundkenntnisse
Computerkenntnisse:	
	Java, HTML, XML, MS Office, Windows, UML, MySQL

Appendix C

Deployment

C.1 Required software

The following software must be installed before the deployment of the new data resource and activities can be performed. Concrete examples (e.g. path names) apply to a Windows workstation as the development of the activities was undertaken using this operating system.

- The Amos II binary release has to be extracted to a local directory (e.g. C:\amos2). The implementation described in this thesis was tested with Amos II release 8 version 2.
- Java 5 SDK has to be installed.
- The Tomcat 5.0.x servlet container has to be available and has to use the Java 5 SDK.
- OGSA-DAI Axis 3.0 has to be deployed on the servlet container.
- Apache Ant 1.7.x has to be available. Using Windows as operating system it is recommended to add the *bin* subdirectory to the *PATH* environment variable.
- OGSA-DAI Axis 3.0 binary release has to be extracted to a local directory. This release is needed for the the build script described in the following section.

C.2 Deployment

The source code provided on the website is a zipped version of an Eclipse 3.4 project. Eclipse is not necessary for the deployment of the data resource and activities as an Ant build file (*build.xml*) is available in the root directory. If Eclipse is used a description can be found in *usage-eclipse.txt*.
The Ant build file allows to deploy the data resource and the activities. The following Ant targets are available to perform these tasks.

- deploy-amos-data-resource installs the data resource in OGSA-DAI.
- deploy-server-activities installs the activities in OGSA-DAI.
- expose-activities-to-resource configures the data resource to be targeted by the activities.

As the deployment depends on the installation location of Amos II, Tomcat 5, OGSA-DAI etc. several Ant properties are set at the top of the build file. These properties or references to properties files in the project are marked with comments. A more exhaustive description is available in the file *readme.txt*.

Appendix D

Use Case Mediation Code

D.1 Amos Mediation

D.1.1 Amos II Peer configuration

```
// Data source 1
jdbc('db1','com.mysql.jdbc.Driver');
connect(relational_named('DB1'),
  'jdbc:mysql://localhost:3306/use_case_mediation_1', 'user', 'password');
import_table(relational_named("DB1"),"PERSON");
// Data source 2
jdbc('db2','com.mysql.jdbc.Driver');
connect(relational_named('DB2'),
  'jdbc:mysql://localhost:3306/use_case_mediation_2', 'user', 'password');
import_table(relational_named("DB2"),"PERSON");
// Data source 3
jdbc('db3','com.mysql.jdbc.Driver');
connect(relational_named('DB3'),
  'jdbc:mysql://localhost:3306/use_case_mediation_3', 'user', 'password');
import_table(relational_named("DB3"),"PERSON");
// Data source 4
jdbc('db4','com.mysql.jdbc.Driver');
connect(relational_named('DB4'),
  'jdbc:mysql://localhost:3306/use_case_mediation_4', 'user', 'password');
import_table(relational_named("DB4"),"PERSON");
// Joining the data sources using the primary keys
create derived type A_B_C_D
  subtype of PERSON_DB1 db1, PERSON_DB2 db2, PERSON_DB3 db3, PERSON_DB4 db4
  where ssn(db1) = id(db2)
  and ssn(db1) = unique_number(db3)
  and ssn(db1) = social_number(db4);
// Performing the mediation
create function long_name(A_B_C_D p)
  ->charstring as select first_name(p) + ' ' + last_name(p);
```

D.1.2 AmosQL-Activity Mediation Client Workflow

```
// Lookup the server and execution resources
ServerProxy serverProxy = new ServerProxy();
serverProxy.setDefaultBaseServicesURL(new URL(
  "http://localhost:8080/dai/services/"));
DataRequestExecutionResource drer = serverProxy.
  getDataRequestExecutionResource(
  new ResourceID("DataRequestExecutionResource"));

// Define involved activities
AmosQLQuery query = new AmosQLQuery();
query.setResourceID(new ResourceID("AmosResource"));
// Set the AmosQL query
query.addExpression(
  "select id(p),long_name(p),telnumber(p),email(p)," +
  "height(p),weight(p),job(p),salary(p) " +
  "from A_B_C_D p where id(p)<=32768;");
TupleToWebRowSetCharArrays toWebRowSet = new TupleToWebRowSetCharArrays();
DeliverToRequestStatus deliver = new DeliverToRequestStatus();

// Connect inputs and outputs
toWebRowSet.connectDataInput(query.getDataOutput());
deliver.connectInput(toWebRowSet.getResultOutput());

// Create the workflow
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query);
pipeline.add(toWebRowSet);
pipeline.add(deliver);

// Excecute the workflow
drer.execute(pipeline, RequestExecutionType.SYNCHRONOUS);
```

D.1.3 Amos II Direct Java Client Application

```
// Java API connection to Amos II peer
Connection con = new Connection("");
String query = "select id(p),long_name(p),telnumber(p),email(p)," +
  "height(p),weight(p),job(p),salary(p) " +
  "from A_B_C_D p where id(p)<=32768;";
// Execution of the query
con.execute(query);
```

D.2 OGSA-DAI Mediation

D.2.1 Client Workflow

```
// Lookup the server and execution resources
ServerProxy serverProxy = new ServerProxy();
serverProxy.setDefaultBaseServicesURL(new URL(
  "http://localhost:8080/dai/services/"));
DataRequestExecutionResource drer = serverProxy
  .getDataRequestExecutionResource(new ResourceID(
  "DataRequestExecutionResource"));

// Define involved activities
SQLQuery query1 = new SQLQuery();
query1.addExpression("select ssn, first_name, telnumber "
  + "from person where ssn<=32768");
query1.setResourceID("MySQLDataResourceMediation1");
SQLQuery query2 = new SQLQuery();
query2.addExpression("select id, last_name, email "
  + "from person where id<=32768");
query2.setResourceID("MySQLDataResourceMediation2");
SQLQuery query3 = new SQLQuery();
query3.addExpression("select unique_number, height, weight "
  + "from person where unique_number<=32768");
query3.setResourceID("MySQLDataResourceMediation3");
SQLQuery query4 = new SQLQuery();
query4.addExpression("select social_number, job, salary "
  + "from person where social_number<=32768");
query4.setResourceID("MySQLDataResourceMediation4");
// Use 3 TupleMergeJoin to inner join DS 1 with DS 2,
// DS 3 with DS4 and the results with each other
TupleMergeJoin mergeJoin1 = new TupleMergeJoin();
mergeJoin1.addColumnIds1("ssn");
mergeJoin1.addColumnIds2("id");
mergeJoin1.addProjectColumnIds2(new String[] { "last_name", "email" });
TupleMergeJoin mergeJoin2 = new TupleMergeJoin();
mergeJoin2.addColumnIds1("unique_number");
mergeJoin2.addColumnIds2("social_number");
mergeJoin2.addProjectColumnIds2(new String[] { "job", "salary" });
TupleMergeJoin mergeJoin3 = new TupleMergeJoin();
mergeJoin3.addColumnIds1("ssn");
mergeJoin3.addColumnIds2("unique_number");
mergeJoin3.addProjectColumnIds2(new String[] { "height", "weight",
  "job", "salary" });
TupleToWebRowSetCharArrays toWebRowSet = new TupleToWebRowSetCharArrays();
XSLTransform xslTransform = new XSLTransform();
FileReader reader = new FileReader(new File("mediation_metadata4.xsl"));
xslTransform.addXSLT(reader);
DeliverToRequestStatus toRequestStatus = new DeliverToRequestStatus();
```

```
// Connect inputs and outputs
mergeJoin1.connectData1Input(query1.getDataOutput());
mergeJoin1.connectData2Input(query2.getDataOutput());
mergeJoin2.connectData1Input(query3.getDataOutput());
mergeJoin2.connectData2Input(query4.getDataOutput());
mergeJoin3.connectData1Input(mergeJoin1.getResultOutput());
mergeJoin3.connectData2Input(mergeJoin2.getResultOutput());
toWebRowSet.connectDataInput(mergeJoin3.getResultOutput());
xslTransform.connectXMLInput(toWebRowSet.getResultOutput());
toRequestStatus.connectInput(xslTransform.getResultOutput());

// Build the workflow
PipelineWorkflow pipeline = new PipelineWorkflow();
pipeline.add(query1);
pipeline.add(query2);
pipeline.add(query3);
pipeline.add(query4);
pipeline.add(mergeJoin1);
pipeline.add(mergeJoin2);
pipeline.add(mergeJoin3);
pipeline.add(toWebRowSet);
pipeline.add(xslTransform);
pipeline.add(toRequestStatus);

// Execute the workflow
drer.execute(pipeline, RequestExecutionType.SYNCHRONOUS);
```

D.2.2 XSL Transformation Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:wrs="http://java.sun.com/xml/ns/jdbc" version="1.0">
  <xsl:output method="html" indent="yes"/>
  <xsl:template match="/">
  <webRowSet>
    <properties>
      <!-- ... -->
    </properties>
    <metadata>
      <!-- ... -->
    </metadata>
    <data>
      <!-- Iterate over all retrieved rows -->
      <xsl:for-each select="wrs:webRowSet/wrs:data/wrs:currentRow">
      <currentRow>
        <!-- Key column -->
        <columnValue>
          <xsl:value-of select="wrs:columnValue[1]"/>
        </columnValue>
        <!-- Mediation of first name and last name -->
```

```
        <columnValue>
          <xsl:value-of select="concat(wrs:columnValue[2],
          ' ' ,wrs:columnValue[4])"/>
        </columnValue>
        <!-- Other column values-->
        <columnValue>
          <xsl:value-of select="wrs:columnValue[3]"/>
        </columnValue>
        <columnValue>
          <xsl:value-of select="wrs:columnValue[5]"/>
        </columnValue>
        <columnValue>
          <xsl:value-of select="wrs:columnValue[6]"/>
        </columnValue>
        <columnValue>
          <xsl:value-of select="wrs:columnValue[7]"/>
        </columnValue>
        <columnValue>
          <xsl:value-of select="wrs:columnValue[8]"/>
        </columnValue>
        <columnValue>
          <xsl:value-of select="wrs:columnValue[9]"/>
        </columnValue>
        </currentRow>
      </xsl:for-each>
    </data>
    </webRowSet>
  </xsl:template>
</xsl:stylesheet>
```

Bibliography

[ABC+08] M. Atkinson, P. Brezany, O. Corcho, L. Han, J. van Hemert, L. Hluchy, A. Hume, I. Janciak, A. Krause, D. Snelling, and A. Wöhrer, *Advanced data mining and integration research for europe admire white paper motivation, strategy, overview and impact version 0.8 (draft)*, admire1.epcc.ed.ac.uk/docs/ADMIRE-WhitePaper.pdf, 10 2008. 45

[adm] *Admire - advanced data mining and integratin research for europe*, admire1.epcc.ed.ac.uk. 44

[AHH+] M. Antonioletti, N. P. Chue Hong, A. C. Hume, M. Jackson, K. Karasavvas, A. Krause, J. M. Schopf, M. P. Atkinson, B. Dobrzelcki, M. Illingworth, N. McDonnell, M. Parsons, and E. Theocharopoulos, *Ogsa dai 3.0 the whats and the whys.* 50

[amo] *Amos ii overview page*, http://user.it.uu.se/ udbl/amos/. 21, 91

[AMP+03] M. N. Alpdemir, A. Mukherjee, N.W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith, *Service based distributed querying on the grid*, ICSOC2003, LNCS 2910, 2003, pp. 467–482. 46

[AOD] Open Grid Services Architecture Data Access and Integration (OGSA-DAI), http://www.ogsadai.org. viii, 1, 3, 13, 15, 38, 39, 40, 41, 42, 43, 45, 52, 53, 54, 55, 56, 63, 79, 83, 85, 90, 91

[BB05] S. Babu and P. Bizarro, *Adaptive query processing in the looking glass*, 2005. 30

[BK06] C. Bauer and G. King, *Java persistence with hibernate*, Manning Publications, 2006. 79, 85, 86

[Bra98] S. Brandani, *Multi database access from amos ii using odbc*, 1998. 23

[CR01] Kristofer Cassel and Tore Risch, *An object-oriented multi-mediator browser*, 2nd International Workshop on User Interfaces to Data Intensive Systems, May, June 2001. 20

[Dat04] C.J Date, *An introduction to database systems*, Addison-Wesley, 2004. 12

[eD08] eXist DB, http://exist-db.org/, 10 2008. 8

[Elm05] J. Elmiger, *An object relational meta-data manager for picture files*, 2005. 22

[ER] D. Elin and T. Risch, *Amos ii java interfaces.* 19, 37, 52, 61, 89

[FHJ+] S. Flodin, M. Hansson, V. Josifovski, T. Katchaounov, T. Risch, and M. Skoeld, *Amos ii release 10 user manual*, http://user.it.uu.se/ udbl/amos/doc/amos_users_guide.html. 3, 56, 63, 66, 69, 77, 85, 92

[fir] *First dig: First data investigation on the grid*, www2.epcc.de.ac.uk/ firstdig. 44

[FKT01] I. Foster, C. Kesselman, and S. Tuecke, *The anatomy of the Grid: Enabling scalable virtual organizations*, Intl. J. Supercomputer Applications, 15(3), 2001. 13

[FR08] G. Fahl and T. Risch, *Amos ii tutorial*, 2008. viii, 37, 57, 64, 73, 74, 75, 76

[GPFS] Anastasios Gounaris, Norman W. Paton, Alvaro A. A. Fernandes, and Rizos Sakellariou, *Adaptive query processing: A survey*. 30

[GPSF04] Anastasios Gounaris, Norman W. Paton, Rizos Sakellariou, and Alvaro A. A. Fernandes, *Adaptive query processing and the grid: Opportunities and challenges*, IEEE, International Workshop on Database and Expert System Applications (2004). 30

[JH03] T. Johansson and R. Heggbrenna, *Importing xml schema into an object oriented database mediator system*, 2003. 23

[JK08] M. Jackson and A. Krause, *An introduction to distributed data management and ogsa-dai*, www.ogsadai.org.uk/courses/gridka08/slides/GridKa08.pdf, 09 2008. 43

[JKR99] Vanja Josifovski, Timour Katchaounov, and Tore Risch, *Optimizing queries in distributed and composable mediators*, 1999. 25, 37, 50

[Jos] M. Jost, *A wrapper for midi files from an object-relational mediator system*. 22

[JR99] V. Josifovski and T. Risch, *Functional query optimization over object-oriented views for data integration*, 1999. 25, 30, 31, 37

[JR02] Vanja Josifovski and Tore Risch, *Query decomposition for a distributed object oriented mediator system*, 2002. viii, 19, 20, 26, 32, 33

[KMA] Karasavvas K., Atkinson M., and Hume A., *Redesigned and new activities v.1.13*, http://www.ogsadai.org.uk/documentation/ogsadai3.0. 39

[Kos00] Donald Kossmann, *The state of the art in distributed query processing*, ACM Computing Surveys (CSUR) **32** (2000), no. 4, 422–469. 28

[KRZ02] T. Katchounov, T. Risch, and S. Zürcher, *Object oriented mediator queries to internet search engines*, 2002. 21

[Lad05] M. Ladjvardi, *Storage manager: Wrapping a b tree storage manager in an object relational mediator system*, 2005. 22

[LHP89] G. Lohmann L. Haas, J.C. Freytag and H. Pirahesh, *Extensible query processing in starburst*, Proceedings of the ACM Sigmond Conference on Management of Data, 1989. 28

[LN07] U. Leser and F. Naumann, *Informationsintegration*, dpunkt verlag, 2007. 12

[LRK01] H. Lin, T. Risch, and T. Katchaounov, *Adaptive data mediation over xml data*, 2001. 23

[MKM+] Atkinson M., Karasavvas K., Antonioletti M., Baxter R., Borley A., Chue Hong N., Hume A., Jackson M., Krause A., Laws S., Paton N., Schopf J.M., Sudgen T., Tourlas K., and Watson P., *A new architecture for ogsa-dai.* 40

[MKN+] M. Matsuoka, S. Kodama, R. Nakamura, N. Yamamoto, H. Yamamoto, K. Iwao, S. Tsuchida, and S. Sekiguchi, *Overview and current state of the geo (global earth observation) grid*, www.gwu.edu/ spi/MatsuokaKodama.pdf. 46

[NAN+] Alpdemir M. N., Mukherjee A., Paton N.W., Watson P., Fernandes A. A. A., Gounaris A., and Smith J., *Ogsa-dqp: A service-based query processor for the grid.* 46

[OD] OGSA-DQP, *Ogsa-dqp*, http://www.ogsadai.org.uk/documentation/ogsa-dqp3.2/userdoc/. 46, 47

[ogs] *Ogsa dqp background*, http://www.ogsadai.org.uk/documentation/ogsa-dqp_3.2/userdoc/background.html. viii, 46, 47, 48

[OV99] M. T. Özsu and P. Valduriez, *Principle of distributed database systems*, Prentice Hall, 1999. 1, 5, 6, 7, 10, 11, 12

[Pet01] J. Petrini, *Accessing web forms from an object relational database system*, 2001. 21

[Pet05] V. Petrauskas, *Object relational wrapping of music files*, 2005. 22

[ppd] *Particle physics data grid*, http://www.ppdg.net. 14

[RBJ03] David De Roure, Mark A. Baker, and Nicholas R. Jennings, *The evolution of the grid*, Grid Computing: Making the Global Infrastructure a Reality, 2003. 13, 14, 15

[RDB08] Overview RDBMS, http://rdbms.ca/database/vendors.html, 10 2008. x, 8

[Ris] Tore Risch, *Amos ii external interfaces.* 37, 52, 53

[Ris03] T. Risch, *Functional queries to wrapped educational semantic web metadata*, 2003. 21

[RJ] Tore Risch and Vanja Josifovski, *Distributed data integration by object-oriented mediator servers.* 16, 18, 19, 22, 23, 24, 26

[RJK03] Tore Risch, Vanja Josifovski, and Timour Katchaounov, *Functional data integration in a distributed mediator system*, Functional Approach to Data Management - Modeling, Analyzing and Integration Heterogeneous Data, 2003, pp. 211–238. viii, 1, 16, 17, 18, 19, 21, 22, 24, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37

[Rod02] C. Rodunger, *Accessing xml data from an object relational mediator database*, 2002. 23

[RWM] A. Rajasekar, M. Wan, and R. Moore, *Mysrb and srb - components of a data grid*, www.npaci.edu/DICE/Pubs/hpdc11-mysrb.pdf. 14

[SCG+] T. M. Sloan, A. Carter, P.J. Graham, D. Unwin, and I. Gregory, *First data investigation on the grid: First dig*, www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/067.pdf. 44

[Sch04] L. Scheuring, *Loading xml schema based data sources into an object relational database system*, 2004. 23

[see] *Ogsa-dai case-study see-geo*, www.nesc.ac.uk/action/esi/download.cfm?index=3672. 45

[Sek] S. Sekiguchi, *A design of the geo grid: Systems of systems federating geospatial data and services*, http://www.nesc.ac.uk/talks/ahm2007/keynote2.pdf. 46

[Shi81] David Shipman, *The functional data model and the data language daplex*, ACM Transactions on Database Systems **6** (1981), no. 1, 140–173. 22

[Sin04] Richard Sinnott, *Grid based clinical trial scenarios*, www.nesc.ac.uk/talks/staff/ClinicalTrialsOutlineScenariosv2.pdf, 2004. 44

[SSA] R.O. Sinnott, A. J. Stell, and O. Ajayi, *Supporting grid-based clinical trials in scotland*, labserv.nesc.gla.ac.uk/projects/votes/edinIHR2.pdf. 44

[Tut08] SQL Tutorial, www.w3schools.com/sql/default.asp, 10 2008. 7

[Tys05] J. Tysklind, *Wrapping a scientifc data management system*, 2005. 22

[Wer04] C: Werner, *Php integration with object relational dbms*, 2004. 53

[WG97] Gio Wiederhold and Michael Genesereth, *The conceptual basis for mediation services*, IEEE Expert: Intelligent Systems and Their Applications, 1997. 17

[Wie92] G. Wiederhold, *Mediators in the architecture of future information systems*, The IEEE Computer Magazine, March 1992. 17

[Xin08] Apache Xindice, http://xml.apache.org/xindice/, 10 2008. 8

[XPa08] XPath, http://www.w3.org/TR/xpath, 10 2008. 8

[XQu08] XQuery, http://www.w3.org/TR/xquery/, 10 2008. 8

[XUp08] XUpdate, http://xmldb-org.sourceforge.net/xupdate/index.html, 10 2008. 8