FAKULTÄT FÜR !NFORMATIK

Contribution to the WebRowSet Technology for Advanced Data Preprocessing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Thomas Lustig
Matrikelnummer 0026005

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Univ. Prof. Dr. Peter Brezany
Mitwirkung:

Wien, 19.11.2008

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

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, November 2008 Thomas Lustig

Abstract

Distributed computing is a very well known term that can be found in a wide range of today's life - from cluster projects like Folding@Home till the SOAP Web services of cloud computing providers (e.g. Amazon, Google, etc.) with the common idea of sharing IT components.

The Austrian GridMiner project at the Institute of Scientific Computing at the University of Vienna has the aim to develop an e-Science infrastructure to support knowledge discovery tasks in databases (KDD). For this kind of applications, the Grid serves as the underlying architecture allowing to extend storage and computing power. One major task of the KDD is the preprocessing stage, where huge datasets have to be processed. It consumes approximately 60% of the entire KDD processing time. That is the reason why it offers a significant potential for optimization. The preprocessing stage itself is often performed on workstation computers and not on the Grid, because the human interaction at some important subtasks of this stage provides a better quality than having everything done automatically. Therefore two major problems which need to be solved occur when carving out the data from the high performance Grid to a standard workstation: much lower main memory and CPU power. Just these issues are addressed by this Master Thesis.

The GridMiner architecture includes a subsystem of data access and integration based on the middleware called OGSA-DAI. To accelerate the preprocessing task, in our approach, the needed basic statistic calculations are shifted from the client side to the server side (the Grid). The statistics are calculated in advance inside the GridMiner. To optimize the main memory (RAM) consumption, the software provides an out-of-core design. To evaluate the datasets in the WebRowSet format, it was necessary to generate synthetic data in this format. A generator of the WEKA software provides random data with different features.

The already existing APIs and libraries were evaluated by comparing them concerning memory consumption and data access functions. The runtime behavior of the standard implementation of the WebRowSet interface in Java 1.5 was measured. For a new implementation, it was important to create a clear design that combines different design patterns.

The result is a newly developed OGSA-DAI activity that creates statistics on the input data. These statistics can be used by the client for requested datasets. This functionality is configurable in a fine grained way to avoid unnecessary calculation effort. The output format of the statistics is a newly defined XML structure. The big dataset files in the WebRowSet format can be processed by the Java implementation. This innovative implementation is based on an out-of-core architecture and uses a new indexing approach. The loading time is independent from file sizes and the parsing time is comparable to the standard implementation, however, much lower memory usage.

For all people who supported me during writing this thesis.
I also would like to thank Prof. Dr. Brezany and his team, who inspired and motivated me to develop a new technical approach.
Last but not least, this work is also dedicated to my parents, who supported me for 28 years.

Acknowledgments

I pay tribute to Professor Dr. Peter Brezany for the opportunity to work on this thesis and for his guidance during the theoretic phase of this work.

I want to express special thanks for the help during the implementation phase to Alexander Wöhrer. The researchers from the University of Edinburgh gave me a great support on the OGSA-DAI side.

Vienna, Austria — Thomas Lustig
19th November 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the very high pace of computer performance increases in accordance with Moore's Law, the complexity of electronic circuits doubles every two years. Home PCs and even game consoles (e.g. Playstation 3) have massive storage and CPU performance available, that can be used when they are idle. One important technical requirement for accessing these resources are network speeds fast enough for exchanging data in reasonable time. Lots of problems were solved, but it is very important to find standards and software architectures to realize the theoretical idea of a virtual supercomputer.

Fundamental research in Grid computing is the core of several projects. While the approach of a generic Grid solution is still at the beginning, specialized Grid projects have made progress. The GridMiner [37] project of the Institute of Scientific Computing of the University of Vienna, led by Prof. Dr. Brezany, is targeting the e-Science research field of Grid computing for data mining. Doing this, it provides an interface for clients to define workflows and manipulate the collected raw datasets to finally discover hidden knowledge in these datasets. The discovery of hidden knowledge is described by a process called knowledge discovery in databases (KDD). One important subsystem for data access and integration is covered by the OGSA-DAI framework, which exports huge WebRowSet files as an exchange format for a decoupled communication between the Grid and its clients.

The inter-disciplinary topic addressed by this diploma thesis is the optimization of a Grid client component for processing huge datasets under strict system constraints, regarding memory and processing power. Instead of having a permanent on-line connection between the Grid and the client workstation, the result of the requested dataset is exported to an FTP space in a persistent XML format, from where the workstation can download it. This XML format is defined as a standard [48] and known as the WebRowSet format. The workstation loads this huge XML document (>90MB) for later processing. For arranging this, an advanced implementation design was developed based upon the diploma thesis of Michaela Pfeifer [38] where the principle idea of a novel implementation of the WebRowSet interface is treated and a paper of Haihui Zhang [61], that describes the usage of an index based XML parser.

For data preprocessing, several statistic calculations (e.g. average, covariance, etc.) are done on

the raw dataset. The statistics of the requested dataset are calculated in advance on the Grid by a software module, because of the limited CPU performance of the workstation. Because the OGSA-DAI execution engine is used inside GridMiner for data access and integration, the computing of the statistics is done inside the OGSA-DAI framework. The software module computing statistics is called "activity" in the OGSA-DAI terminology. This approach is based on the D3G work by Wöhrer [59]. This thesis extends the approach by an implementation that allows a very fine-grained parameterizing feature of the activity.

Figure 1.1 summarizes the context of this thesis by showing only the involved elements of this diploma thesis. The GridMiner uses the data integration subsystem, which is based on OGSA-DAI framework, to access several data resources (represented by the database symbols). A data preprocessing client application on a workstation defines and controls the workflow inside the OGSA-DAI framework via the Web service interface of OGSA-DAI. The raw dataset (non-preprocessed data) is transferred indirectly (via a storage space, e.g. FTP space) to the data preprocessing workstation client by using the persistent XML exchange format WebRowSet. After preprocessing, the data is sent back to the Grid in the same way by the workstation client.

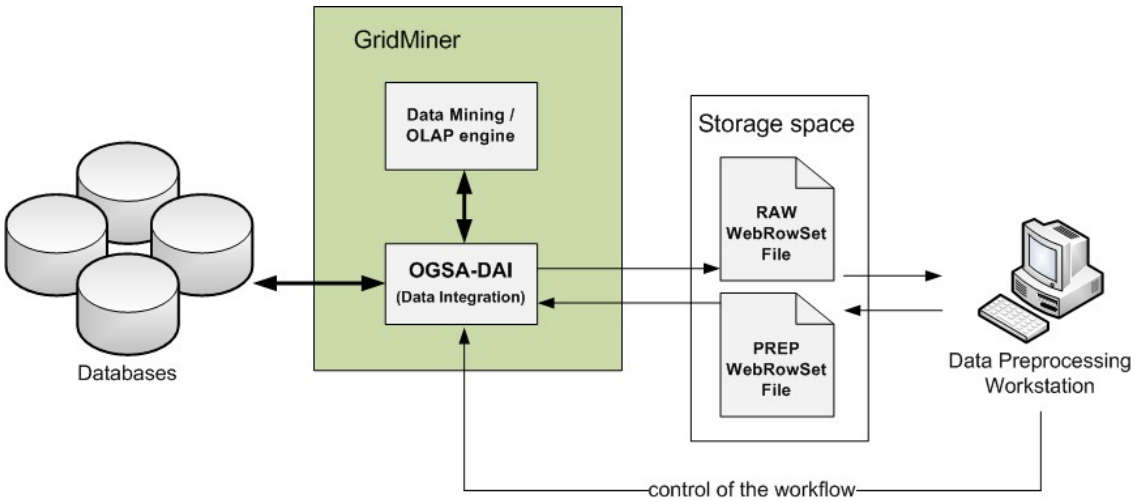

Figure 1.1: Overview of the diploma thesis context

1.1 Motivation

In general, "Knowledge Discovery" means a set of methods, that are used to extract information (e.g. how could the offer be optimized for which kind of customer?) from a huge amount of data. Because it is easy and cheap today to generate data by using hardware, e.g. sensors, or software, like customer relationship management applications, the quantity of the generated data (the raw data) is gigantic. The raw data has to be prepared in a data preprocessing step for the knowledge discovery algorithms and cannot be used directly. The Knowledge Discovery in Databases (KDD) process defines, how the information are transformed to an important knowledge base, on which

decisions and further actions are made.

The motivation for this diploma thesis was to improve the data preprocessing task, which is part of the general KDD process. I have chosen the preprocessing task, because it consumes up to 60% of the whole KDD processing time (value taken from Pyle, 1999 [39]). So it offers a high optimization potential.

The KDD process could be seen as the core process in several e-Science applications. The word "e-Science" stands for "electronic science" or "enhanced science". It describes an integrated infrastructure, that enables the usage of the required resources and tools for research. The main targets are to support the research process itself, by handling collaboration (e.g. providing a digital community platform) and the management of the available computing capacity (e.g. common used Grid architecture). On one hand this leads to the requirement of being able to work in a distributed way independent of the physical placement of research teams and on the other hand to the need for dynamic allocation of technical resources.

Importance of Data Preprocessing and Grid Computing

A popular example of a typical e-Science application is CERN's Large Hadron Collider (LHC) [9]. Its target is the integration of high energy physics projects into a common used research field because of the immense costs (around 3 billion Euros) and the necessary computing power (processing of 15 Pentabytes/year). In order to realize the distribution, it has to be mentioned, that 6 big experiments are arranged. Over 10.000 scientists and engineers in about 100 countries need to be coordinated. The LHC creates approximately several TBytes of data per second. Most of this data is not interesting and already known from other experiments. Data preprocessing has to be done to reduce the data size to a feasible size.

A computing Grid was established - the LHC Computing Grid(LCG) [8]. The core concept of Grid computing is to abstract the physical placement of resources (people, hardware, etc.) and group them into logical sections - the virtual organizations (VO). In terms of the LHC, the experiments define the VO boundaries. So each VO has dynamically assigned resources to fulfill its work. Like LCG shows, another working Grid concept is the break down of hierarchy by taking into account the performance of processing units. This so called multi-TIER model offers maximized efficiency. An important feature of all Grids is to compensate the differences of the connected parties. Thereby "Grid community" members can provide their resources to the rest. Due to the costs of supercomputer hardware, it is necessary to find and provide a solution, how standard PC hardware in standard configuration (no specialized operating system) can be used for calculations.

In case of the KDD process research field, this means optimizing the time and memory consumption in a standard workstation PC by putting the focus on data preprocessing. The concrete enhancement of my approach in the context of the GridMiner environment consists of two parts:

- the possibility of reading and modifying a big dataset file in WebRowSet format
- a shorter processing time by using additional precalculated statistics of the underlying data during the preprocessing process

The proposed research work evaluates the existing file and database indexing methods and implements an appropriate indexing method targeting large WebRowSet files as produced by the de-facto standard middleware for database access on the Grid named OGSA-DAI. It will compare performance, scalability and study existing data preprocessing methods to choose one to be used as proof-of-concept validation of the developed indexing methods. Furthermore, the selected preprocessing method will be implemented in Java and encapsulated as a SOA architecture service, which may be later attached to the GridMiner technology pool.

1.2 Thesis Objectives and Approach

Preprocessing operations can be done "in-core", by manipulating the data inside the primary memory (e.g. main memory) or "out-of-core", by manipulating the data outside the primary memory on a secondary memory (e.g. hard disk). The main objective of this diploma thesis is to enable out-of-core preprocessing of large data sets by advanced data indexing methods. This work will be conducted in the context of the GridMiner project, a joined research project between the Vienna University of Technology and the University of Vienna. The project focuses on extending state-of-the-art Grid technology to provide efficient and scalable knowledge discovery in databases and other large data sets attached to the Grid. This effort will consist of two main parts:

- improve and enrich available GridMiner research and development work
  - advanced Java WebRowSet interface implementation.
  - studying indexing methods.
  - monitoring main memory space utilization and autonomic switching between index mechanisms for performance/scalability.
  - balancing between CPU time and memory usage.
- novel contributions
  - investigation of access patterns of data preprocessing methods.
  - analysis and specification of requirements of preprocessing methods in productive environments.
  - increasing abstraction level of indexing APIs.
  - evaluating the results achieved by designing and implementing a data preprocessing application.
  - and encourage wider adoption of the developed approach.

Approach

The approach for reaching the objectives starts discovering the research environment of the GridMiner project to determine the requirements by analyzing the use cases. After this, the already

existing approaches of Michaela Pfeifer's diploma thesis [38] and Alexander Wöhrer's D3G paper [59] are studied. Then I continue with the evaluation of existing technologies for XML parsing including the needed statistical methods for data preprocessing tasks. Based on the design criteria and requirements, I develop an appropriate architecture of the software components. To accelerate the design process, I use design patterns, which also support the re-usability of the software. The implementation is done with a test first method using JUnit for unit tests. The creation of synthetic test data is needed because of different test data formats. An Apache Tomcat based OGSA-DAI setup is used as the runtime environment for the server components. The tests on the finished components prove the expected results.

1.3 Results Achieved

A new developed OGSA-DAI activity, called "MetaStatisticsNG" activity, allows the calculation of statistical metadata on the server side in a fine grained way (its possible to define a list of statistical functions for each column of a relational table) to save also server resources. I created two types of statistical functions: Standard functions, which operate within a single column, and aggregator functions, which can be used to operate on more than one column. The available statistical standard functions are: Minimum, maximum, average, quartiles (Q1, Q3, and median), standard deviation, and covariance. The available statistical aggregator functions are: Covariance and sumXYdelta. The output of the new activity is a self-defined XML format. By connecting the output to other activities, the XML stream can be saved in a file and delivered to an FTP storage. For reading this XML format, I created a Java class that can be integrated into data mining applications. A demonstration application has been developed that uses these results for a Principle Component Analysis (PCA) and a "values out-of-intervals" calculation.

For handling the huge OGSA-DAI datasets, the novel WebRowSet implementation was developed, that enables the loading of these XML files on standard workstations with a limited main memory size by a new out-of-core architecture. The implementation uses the Java WebRowSet interface. So it can be directly used by applications, which are using the standard implementation. The operations of the new implementation were designed to support data preprocessing applications. Thus, the interface only implements mandatory methods for reading, updating, inserting, and deleting values. The ramp-up speed till getting able to access the first line is much shorter (0,043 seconds with the novel implementation instead of 23 seconds with Sun's implementation) and approximately independent from the file size.

Because of the scalable software architecture by using well defined interfaces, it is easy to enhance it with further improvements.

The implementation of the WebRowSet interface and the statistic activity including JUnit tests can be downloaded from http://www.gridminer.org/dppms.html.

1.4 Organization of the Thesis

Depending on the readers knowledge, it is recommended to start reading this thesis at different chapters. While the needed technical and mathematical background for data preprocessing is explained in Chapter 2, Chapter 3 is dedicated to the software requirements. The software engineering part of the realization phase is treated in Chapter 4. If the reader is only interested in the verification and its results, Chapter 5 should give the answers. The suggestions for future work are presented in Chapter 6.

Chapter 1 introduces the whole thesis and explains the necessity of the novel implementations of WebRowSet and the statistics activity including a brief description of the scientific environment.

Chapter 2 provides the technologies and methods for an easier understanding of the following chapters. It starts with a more detailed survey on computing grids and continues with describing the embedding of this thesis into the GridMiner project. Basic functionality of the OGSA-DAI framework is explained. The statistical background introduces mathematical terms like "deviation", "distribution" and the "Principle Component Analysis" that are used for data preprocessing. Further, different aspects of the physical memory splitting in workstation PCs are illustrated and concepts of the Java virtual machine (JVM) are pointed out.

Chapter 3 focuses on the aspects of the advanced preprocessing workflow, by starting at the use case scenarios and explaining the target of this thesis more detailed. The requirements for the main parts - WebRowSet implementation and the OGSA-DAI activity are analyzed and used for determining the needed design criterias.

Chapter 4 first starts with a brief summary of the used technologies (Tomcat Application server, XML parsing APIs, etc.). This section is followed by a description of the used development environment. After this, two preprocessing methods are described: "Median Absolute Deviation" (MAD) for out-of-intervals and "Principle Component Analysis" (PCA) for dimensional reduction. It includes an explanation how the results of the statistical metadata activity are used. It closes with the illustration of the UML diagrams, taking into account the used software engineering patterns for client side and server side. Finally, the resulting software architecture is presented and important concepts of the statistic calculations in relation to the requirements are shown (e.g. how the calculation of the median is done by combining a binary search algorithm with memory block copying).

Chapter 5 contains the description of the test setup and the evaluation of the results. It pictures the needed testing tools for measuring Java specific performance, the performance of the file system and other components. I focus on using the test results for a comparison with other architectures and the understanding of the reason why they may differ. Also the generation of the synthetic test data created by the modified Agrawal generator of Waikato Environment for Knowledge Analysis (WEKA) [32] is introduced.

Chapter 6 interprets the results and summarizes the conclusions. It also lists the ideas of enhancements and potential leaks of the thesis, that may be relevant for the future, but their consideration would have exceeded the time horizon.

Chapter 2

Background of the Thesis

The following chapter offers background information about the research field of this thesis and the importance of new options based on the results of this work. It is separated into a description of Grid technology in connection with the KDD and another part centering on the specialty of the key-technologies. Furthermore, it offers a detailed description of the use cases which are determining the software requirements. Finally, the software architecture is presented according to these requirements.

2.1 Grid Overview

The term "Grid computing" was introduced by Ian Foster in the book "The Grid: Blueprint for a New Computing Infrastructure" [16] released in 1998 and describes an infrastructure that allows to share different resources (storage space, databases, hardware, and processing power) across geographically separated places.

Foster's definition of Grid computing was refined in the second edition of the book [51]:

The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization.

Grid computing is seen as the successor of meta-computing, where it was tried to connect computing clusters together that consisted of different architectures. A basic abstraction layer was used for balancing the system differences.

After the success of the local interconnection of computers to bigger clusters, it was clear that the advantage of distributed computing has a practical impact on the users. That's why the research of distributed computing was enforced to develop the next level - the Grid.

In the Grid, the prime directive is not only sharing pure CPU power, but instead managing different resource types to the parties (e.g. virtual organizations), as introduced with in the survey of the LHC computing Grid. The resources, have to be allocated transparently and on-demand (in realtime) by the application or the virtual organization that requested them. This features could be compared to the requirements of a multi-tasking PC environment. The ability of handling this could be seen as the border between a simple cluster and a Grid.

By Ian Foster's definition, the Grid has to fulfill the requirements of controlling resources without central supervised management by using open standards and open interfaces to provide non-trival services to the user.

In terms of scientific computing, Grids open new options for providing much more performance for less costs than a single cluster or supercomputer provide. So Grids can be adapted and build for many different use cases to replace existing infrastructure for running more powerful applications. That's why Grid computing is ideal for the Knowledge Discovering process - one research area of the Institute for Scientific Computing in Vienna.

Today, Grid computing is still at the beginning and there are lots of scientific projects running in parallel to develop the needed Grid technologies. It is a quite complex exploration region to develop possibilities for coordinating a shared usage of resources by taking also other aspects like performance, costs, security, dynamic changing virtual organizations, etc. into account. At the moment, lots of proprietary Grid solutions were constructed without the option of working together. Some well known ones are:

- **EGEE (I-III) [50] - Enabling Grids for E-sciencE**
  European Grid project, which has a strong focus on applications for high energy physics. It could be seen as the successor of the LHC Computing Grid (LCG). It is still continued. The middleware gLite was developed for EGEE and is currently used under version 3.1. Until now, EGEE was not able to provide software, that is easy usable for general Grid applications beside from the high energy physics sector.

- **XtreemOS [42] - Grid enabled OS**
  The target is to build a Linux-based operating system that provides functionality of virtual organisations in a native way. The result will be available in open source.

- **D-Grid [21] - integration project of D-Grid Initiative**
  A German project institution with the aim to support Grid projects evaluating different middleware and architectures (e.g. gLite, UNICORE, etc.) for different usage fields.

- **GridMiner [31] - Knowledge discovery Grid**
  An Austrian Grid project that focuses on combining data mining and On-Line Analytical Processing (OLAP) on Grids using Globus Toolkit [3] for the middleware.

- **ADMIRE [13] - Advanced Data Mining and Integration Research for Europe**
  An EU project for developing a high level Grid model for data mining and integration. Universal Service Management Technologies (USMT) is featuring the middleware layer.

The future vision is to use the Internet for building some kind of global Grid, that can be used by everyone.

While security aspects are not really critical for most scientific research projects (here Petabytes of data are transferred in a distributed way), other applications demand higher security architecture.

Although some magazines handle the term "cloud computing" as the next stage of Grid computing, a clear distinction between Grid computing and cloud computing has to be made. Cloud computing means to provide computing resources in "clouds", that are build up and published by service providers. In this type of Grid computing the attached machines are virtualized.

Grids can be classified by their field of application [12]:

- **Data Grids** allow access to distributed databases through a Grid portal.
- **Computing Grids** focus on sharing and providing pure CPU performance to the user. Seti@home is one of the famous computing grids.
- **Resource Grids** share lots of different types of resources.
- **Knowledge Grids** are optimized for operating together on distributed knowledge (e.g. collaboration).
- **Service Grids** are often known in the context of Web services, like Amazon Web Services (AWS).

As introduced, it is important to make the Grids compatible to each other. One possible software architecture is defined by the Open Grid Service Architecture (OGSA) standard [60] with the main focus on openness. A popular implementation of the OGSA standard is provided by the Globus Toolkit framework (actual version 4), that is often used as Grid middleware (e.g. by the GridMiner project). Since GridMiner focuses on KDD applications, it uses an additional framework component for specifying data processing workflows and providing a homogeneous interface to heterogeneous data resources for data access and integration - the OGSA-DAI framework.

The Organization for the Advancement of Structured Information Standards (OASIS) published a collection of specifications for stateful Web services - the Web Services Resource Framework (WSRF) [28]. In the WSRF, the Web service implementation is separated from the Web service state (e.g. session information). The Web service state is stored in an XML data structure, a so called "Resource". WSRF enabled the convergence of Web services, defined by the World Wide Web Consortium (W3C) [53] and the Grid services, defined by OGSA. WSRF compatible Web services are state-full Web services with transactional functionality and can be used as Grid services.

2.2 Context of the Work

The environment of this diploma thesis was mainly influenced by the current two Grid computing projects of the Institute for Scientific Computing - GridMiner and ADMIRE.

2.2.1 GridMiner and ADMIRE

ADMIRE stands for Advanced Data Mining and Integration Research for Europe and benefits from the research experiences made so far with the earlier started GridMiner project [37]. The GridMiner project deals with the parallel implementation of two complementary technologies - data mining and On-Line Analytical Processing (OLAP). It evaluates and designs the basic technology for data mining and OLAP on the Grid by using the OGSA-DAI framework.

GridMiner

The aim of GridMiner is the development of an infrastructure for supporting e-Science applications. The concrete use case is to support the management of patients with Traumatic Brain Injuries (TBI) for determining their next medical treatment. This is done by classifying the patients into five different classes. For the creation of this classification knowledge, a huge amount of cases is used to improve the final model. This data is delivered by different hospitals. A group of hospitals is represented by a Virtual Organization (VO). Also privacy aspects in handling the data have to be considered. Figure 2.1 gives an overview of the roles and use cases.

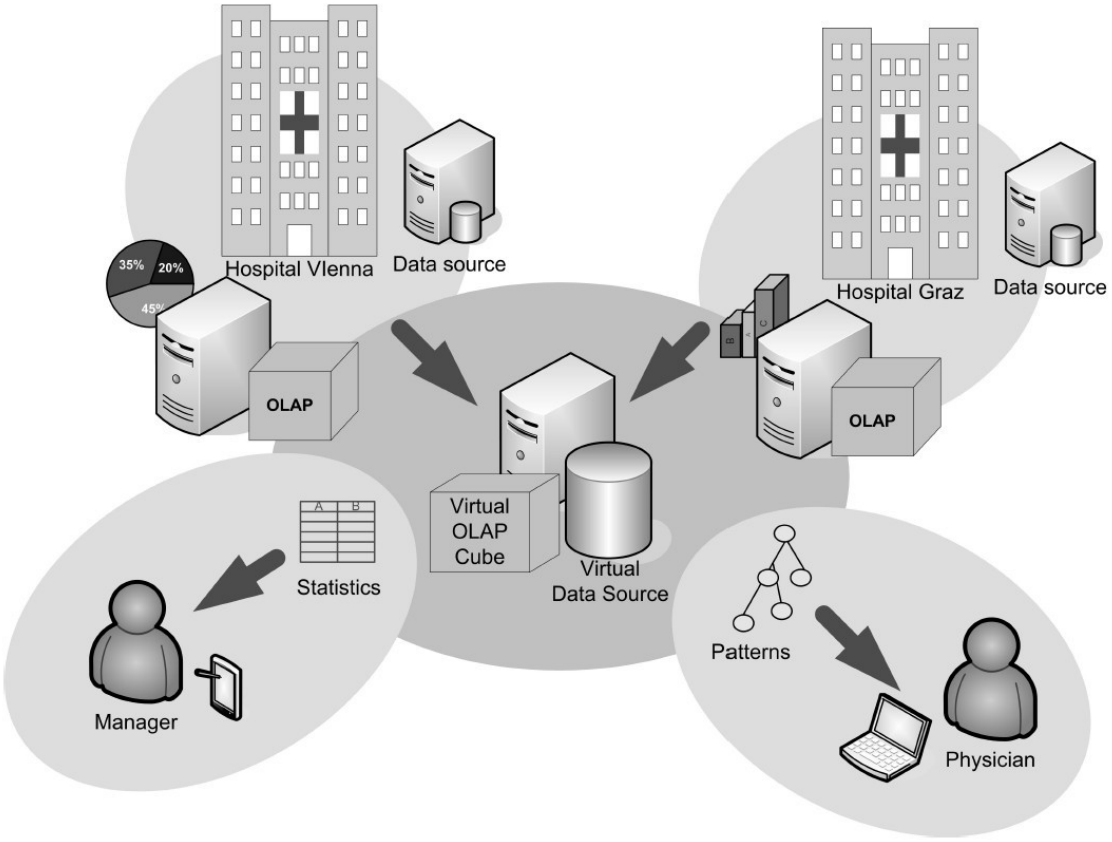

Figure 2.1: GridMiner use case scenario[36]

The GridMiner framework consists of two main components as defined in Brezanys paper [36]:

Technologies and Tools *They are intended to effectively assist application developers to develop Grid-enabled high performance analytics applications. GridMiner includes services for sequential, parallel and distributed data, text mining, On-Line Analytical Processing, data integration, data quality monitoring and improvement based on data statistics and visualization of results. These services are integrated into interactive workflows, which can be steered from desk-top or mobile devices. A tool called workflow composition assistant allows semi-automatic workflow construction based on the Semantic Web technology to increase the productivity of analytics tasks.*

Use Cases *They show how the above technologies and tools work together, and how they can be used in realistic situations. The presented research and development was conducted in cooperation with some of the worldwide leading Grid research and application groups. The set of pilot applications directly pro?ting from the project results includes the medical domain (cancer research, traumatic brain injuries, neurological diseases, and brain informatics) and the ecological domain (environment monitoring and events prediction).*

The target is to provide tools for knowledge discovery from different heterogeneous data sources in a distributed environment.

Application area for evaluation: medicine

- Treatment of TBI patients [7]
- Prediction of the outcome of seriously ill patients [36]
- Focus on data mining and OLAP [11]

ADMIRE

The project aim of ADMIRE is to merge different sources of information and accelerate the access to them. This is done by covering the heterogeneity of service data and processes by an abstract view. The ADMIRE infrastructure consists of several gateways connected together over the Internet. These nodes communicate by using the Infrastructure Service Bus (ISB). Figure 2.2 shows this complex matter in an abstracted picture.

The aim of ADMIRE is to deliver consistent technology with high usability for extracting information and knowledge, provided by an abstract view of data mining and integration (DMI) It develops a high-level language with tools which allow an abstract mapping onto a service-oriented architecture.

Application area for evaluation: large-scale enterprise systems

- Flood modeling and simulation
- CRM (Customer Relationship Management)

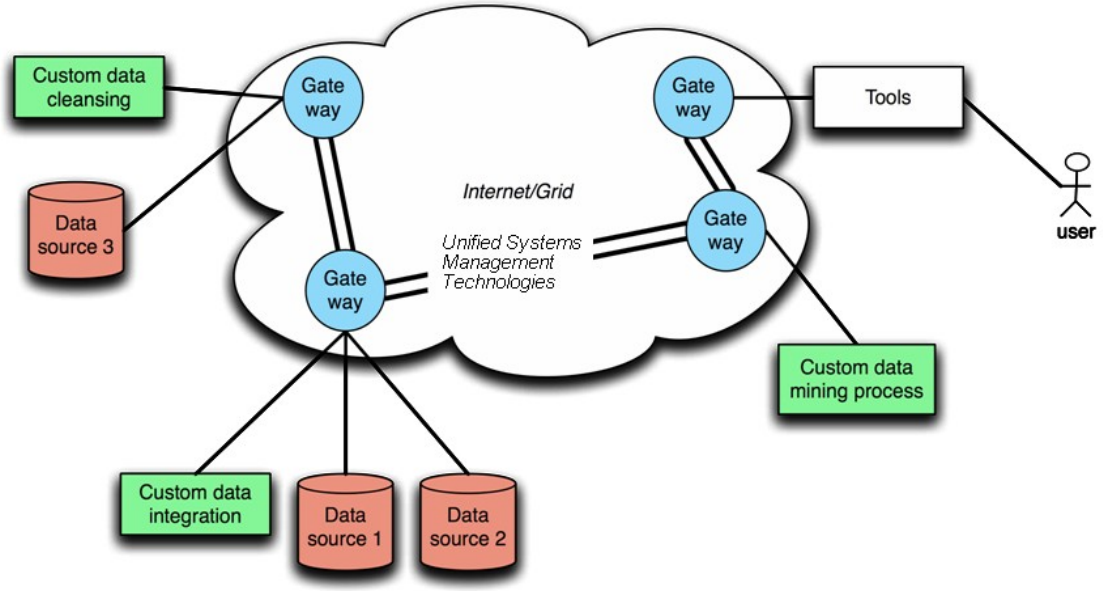

Figure 2.2: ADMIRE architecture [4]

2.2.2 OGSA-DAI and SOA

From the position of e-Science, **SOA - Service Oriented Architecture** supports the integration of software and hardware components. Because of this, a type of architecture is chosen, that focuses on the low level communication between the modules and abstracts them to services, that are provided to the users. This kind of software model is called Service Oriented Architecture(SOA). OGSA-DAI follows the SOA principle.

OGSA-DAI stands for Open Grid Service Architecture Data Access and Integration. It is a Java framework based on the OGSA infrastructure standard to share different data resources like databases, files, etc. across a Grid. This framework is defined in the following way in its documentation [30]:

- *an extensible framework*
- *accessed via web services*
- *that executes data-centric workflows*
- *involving heterogeneous data resources*
- *for the purposes of data access, integration, transformation and delivery within a Grid*
- *and is intended as a toolkit for building higher-level application-specific data services*

Figure 2.3 shows the integration of the novel developed statistics activity *MetaStatisticsNGActivity* ("NG" stands for "Next Generation") inside the GridMiner environment. The preprocessing

subsystem consists of several blocks that match with the preprocessing stage of KDD. The interface to OGSA-DAI is accessed by different preprocessing modules.

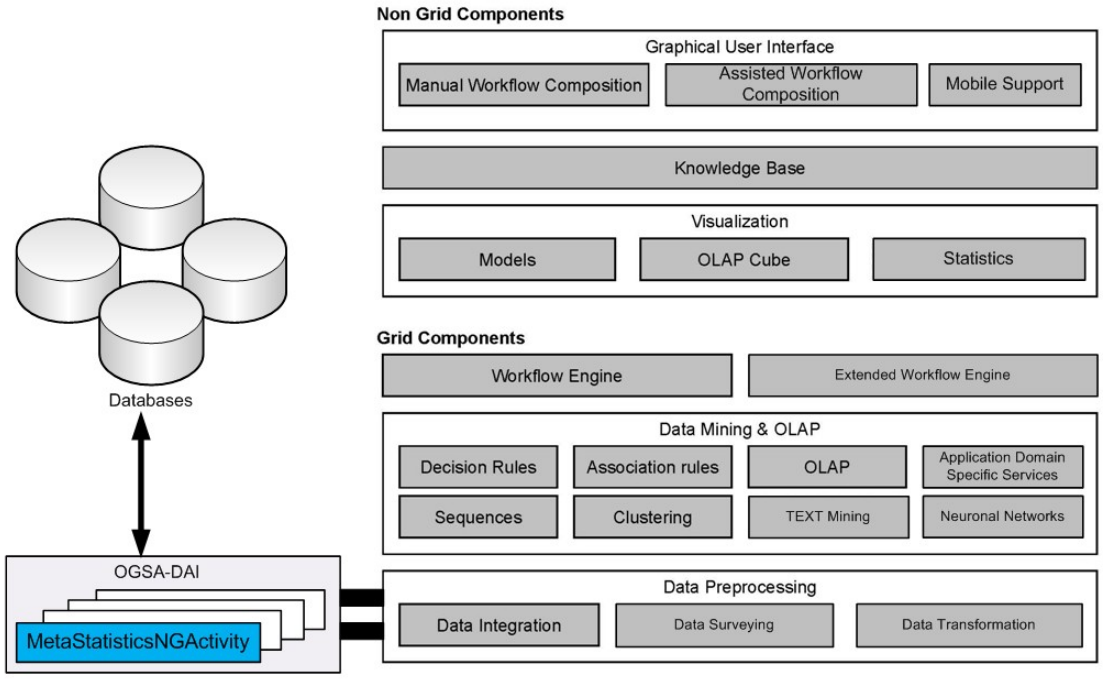

Figure 2.3: Placement of the novel statistics activity in the GridMiner architecture

Figure 2.4 shows the interaction between a user and different services during the preprocessing task by comparing a normal SOAP- and an OGSA-DAI workflow. The standard workflow could be seen on the left side, where the user interacts with single Web services that use SOAP communication for control message flows (thin lines, with low bandwidth usage) and data transfer flows (thick lines, with high bandwidth usage). During the workflow, a huge amount of data is transferred several times between user and the services of the server side (OGSA-DAI, Transform Web service, and FTP server).

When using the features of the OGSA-DAI framework, which could be seen on the right side, the workflow is defined on the client side by connecting the components together, but is executed on the server side by the OGSA-DAI service. This minimizes the communication and data overhead compared to the standard workflow. The final outputs are transferred only at the very end of the process to their final destination (e.g. an FTP storage space).

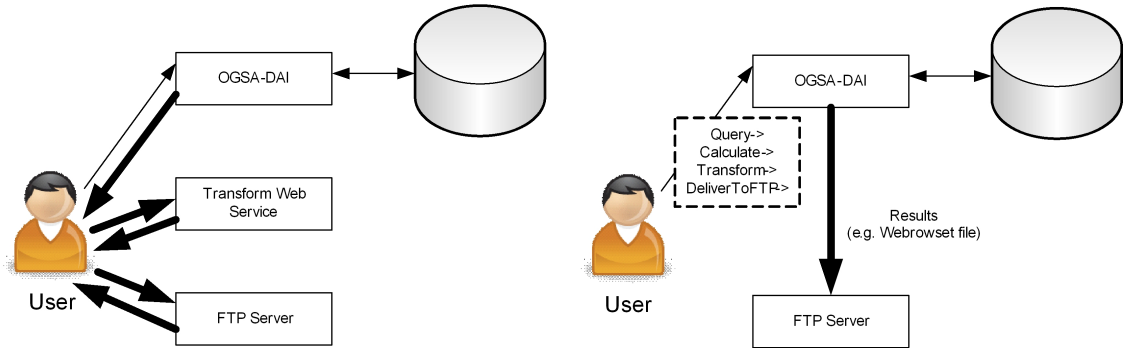

Figure 2.4: Comparison between normal SOAP- and OGSA-DAI workflow [30]

OGSA-DAI is available in two distributions, depending on the underlying Web service layer:

- Globus Toolkit edition - This is used when integrating OGSA-DAI in a Grid that is based on Globus Toolkit, like the GridMiner project.
- Apache Axis 3.0 edition - This version of OGSA-DAI is used in this diploma approach, because it is easy to setup and develop activities. For the developed activities, it does not matter which edition of OGSA-DAI is used. They are portable as long as the version number matches.

The user defines the workflow by connecting function units, so called "activities". This is done by connecting the in- and outputs of the activities to allow the data flow from one activity to the other. Although the output of an activity is normally done by using Java objects, the format (e.g. the class) itself may differ. Therefore transformation activities can be used to convert between different formats. To sum up, OGSA-DAI is responsible for data access, updates, transformations and delivery. A more detailed description of the workflow used in this diploma thesis is described later, but Figure 2.5 shows the potential components.

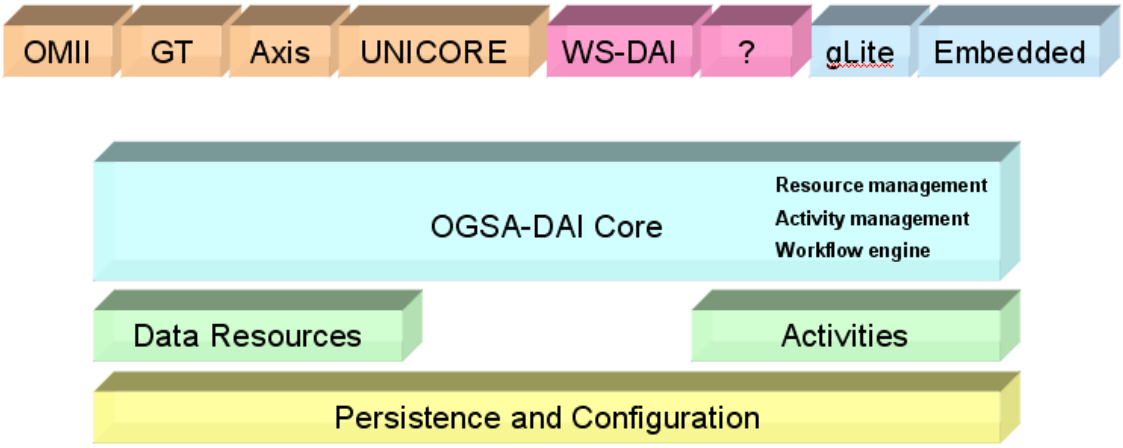

Figure 2.5: OGSA-DAI components overview [30]

The OGSA-DAI core contains management features to control data resources (e.g. databases, files, etc.) and activities. The workflow engine inside the OGSA-DAI core provides methods for definition and execution of workflows that consist of connected activities. The persistence and configuration component offers functions for parameterizing the OGSA-DAI framework. The boxes (e.g. “GT”, “Axis”) on the upper side of Figure 2.5 represent the different possible presentation layer interfaces of OGSA-DAI, e.g. WSRF-compliant services using Globus Toolkit (GT).

Changes in the latest version of OGSA-DAI

There were some major changes made in the architecture of OGSA-DAI between version 2.2 and the latest release 3.0 which have to be taken into account when developing or porting activities. This is important, because of the lack of backward compatibility between OGSA-DAI 3.0 and earlier releases.

Older OGSA-DAI versions < 3.0 used “Perform documents” to define an OGSA-DAI workflow and “Response documents” for returning the actual status or the output of the OGSA-DAI service. The activity framework and the core engine are now using a Java request object and a request status object without the marshalling and unmarshalling of XML documents.

Beside standard Java types as inputs and outputs, OGSA-DAI supports two new data types since version 3.0 - Tuples and MetadataWrapper. Tuple objects represent rows that contain column elements. Tuples are recommended by OGSA-DAI developers as a format for relational data. Using Tuples as a format forces the development of activities that provide a common standard. MetadataWrapper objects are used to cover any other objects as metadata. The interpretation of these wrapped objects is specified by the implementation of the activities.

2.2.3 WebRowSet - The exchange format of OGSA-DAI

OGSA-DAI is often used to operate on data represented in tables of relational

Inside OGSA-DAI, between the activities, the table data is represented by Java objects, called “Tuples”. Tuples provide an efficient way to transfer data between activities inside the OGSA-DAI framework, but they are not capable as a persistence format for data exchange between OGSA-DAI and client applications. Thats the reason, why OGSA-DAI natively supports the persistence WebRowSet XML format.

Specialized activities can be used inside the workflow to convert Tuples to WebRowSet and vice versa, e.g. the “TupleToWebRowSetCharArraysActivity” activity transforms Tuples to WebRowSet and the “WebRowSetCharacterDataToTupleActivity” activity back to Tuples. Because WebRowSet could be seen as the standard persistence format for several data mining applications, it is very important to offer a scalable interface for access operations (e.g. read lines, insert lines, etc.). Due to the memory limitations of the standard implementation, this thesis is about a novel designed implementation focused on scalability.

Figure 2.6 demonstrates a possible workflow of using the OGSA-DAI framework for accessing two data resources (e.g. a database, a flat file, etc.) and the WebRowSet XML format for transferring a data set to the client application. The client application accesses and modifies the data set and creates a WebRowSet file, which can be used by another application or an OGSA-DAI activity.

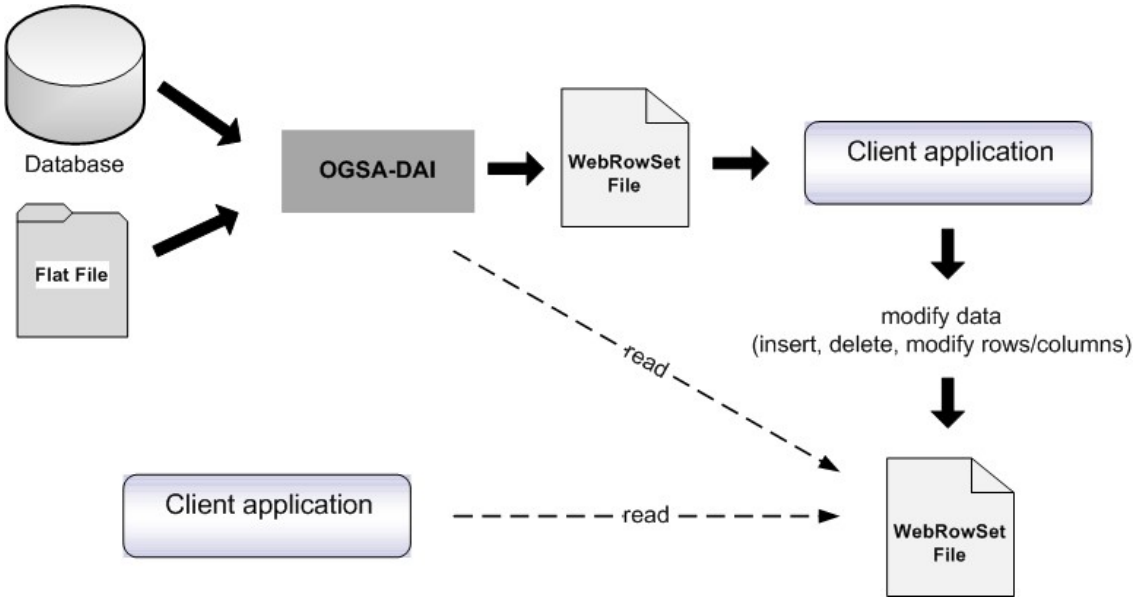

Figure 2.6: WebRowSet processing workflow

2.2.4 Data Preprocessing as part of the KDD process

The KDD process

The main target of this work is to support preparation tasks for data mining applications. Although the results of this work can be used in a generic way of data preprocessing software, the core components are dependent to the usage of the OGSA-DAI framework. For a better understanding of the work, it is relevant to get a clear view of the process of discovering knowledge - formally known as the Knowledge Discovery process for Data (KDD) [58]. The basic steps of the complete KDD process are presented in Figure 2.7 and the task, which contains the new developed software components is highlighted. The process is separated into six big tasks, that represent the different stages of the process chain: data selection, data preprocessing, data transformation&integration, data mining, pattern evaluation and presentation.

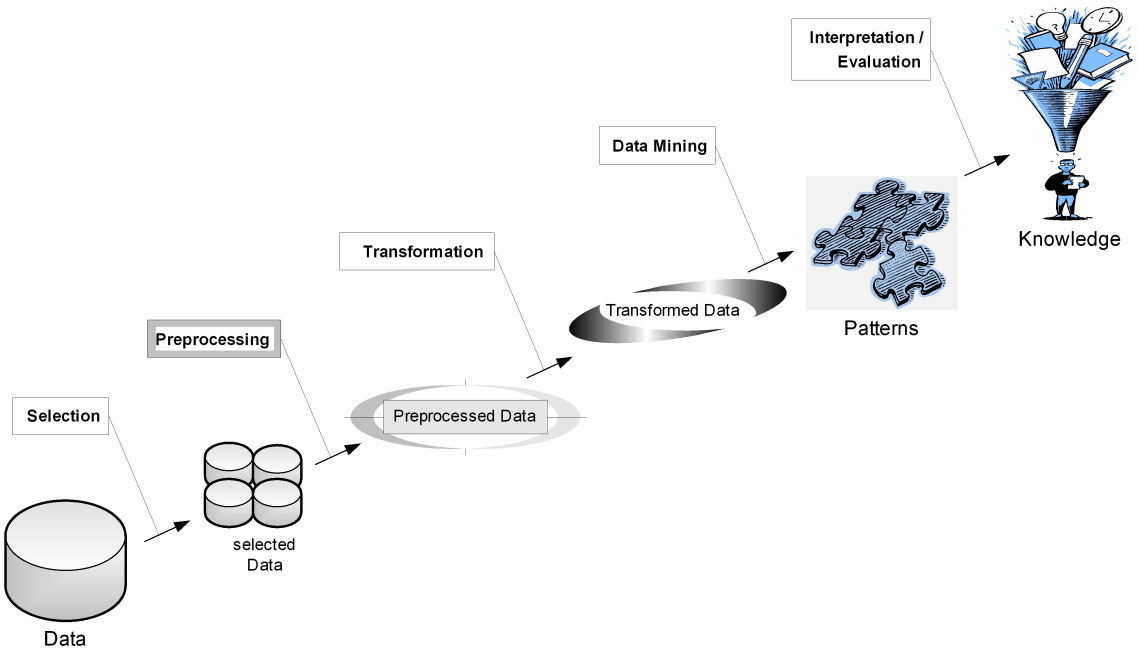

Figure 2.7: Tasks of the Knowledge Discovery Process

At the beginning, data has to be selected by querying the data resource. The step **Data Selection** provides this first subset of data. In productive usage, this query is done on more than one data resource, even different resource types are affected (files, database, etc.).

Normally, the selected data is not complete, neither consistent. This has multiple reasons like different writing formats, simple typos when entering data, etc. The raw data is also called “dirty” because of this features. This dirty data is handled by the important KDD task called **Data Preprocessing** task. It is explained in more detail later on, but it can be noticed, that it provides integrated (when multiple data resources were used in the data selection task) and clean (no missing data, removed noise of data, etc.) data, that can be used in the next level - the “Data Transformation” [34]. Also lossy operations (where “irrelevant” information are removed) are

used for reducing the amount of the data (sampling) or the dimensional complexity (dimensional reduction).

In the **Data Transformation** phase, multiple further operations are applied to the cleaned data. Transformation means, to map the data values to an alternative projection space. This can be e.g. done by normalizing the data to fit them into a range of a float value between 0 and 1 (this is important, because the later applied data mining algorithms are optimized for this variable range). The task of data transformation is often combined with data preprocessing.

Of course, one of the essential steps in the KDD is the extraction of knowledge by finding patterns in the huge amount of data, which is done by the **Data Mining**. Its goal is to search for structures in the provided data. The quality of the results of the used data mining algorithms (classification, clustering, etc.) heavily depends on a proper data preparation of the earlier tasks.

The final tasks are called **Evaluation** and **Presentation**. The results of the data mining step in their raw format cannot be used by the end user. It is important to define which results are relevant for the presentation. The relevance of the discovered patterns is described by parameters like "support"- or "confidence" level, which are assigned at the "Evaluation" phase. The found knowledge has to be transported in an appropriate, understandable way to the end user. Therefore, different visualization techniques, like rule sets, diagrams, etc. are used for presentation and explanation.

Data Preprocessing

Within the data preprocessing stage, there are still sub-tasks that can be or have to be executed on the used dataset [22]. One of the basic subtasks is **Data Cleaning**. Real world data in its raw form (as it was entered into the database) is often noisy, incomplete and inconsistent. Noisy means that the dataset contains values, that are "far away" from their expected deviation. They are also known as outliers and represent error-data. As the word "incomplete" suggests, this kind of data contains cells with lost information - attribute lacks. These lacks could be divided into simple missing values and values with lost information due to aggregation operations. The class "inconsistent" contains not matching data on a semantic level (e.g. joining two tables of economic data, results in a table of more than one capital city of the same country). In this diploma thesis the robust calculation of the deviation is done to identify "outliers" also called - values out-of-intervals.

Data Integration and Linkage is done when datasets from multiple data sources are combined or tables from the same source are joined. This is often the critical point where inconsistencies are caused by different format schemes (e.g. notation order of the firstname, lastname or the number format of country codes). Even the existence of lots of redundant data is critical concerning to the negative effect on the speed of the whole KDD process.

Data mining algorithms are often implemented by expecting a specific input format or range. This is done by using **Data Normalization and Aggregation** methods for transformation. It depends on the class of data if and how transformation has to be done. As an example, neural networks operate on an input data range between 0 and 1. In this case it is necessary to transform the values to a new representation by normalizing them. Beside of normalization, also a transformation

to another number space can be done. Aggregation means the possibility of summarizing values together (e.g. sum up the sales per specific regions).

Beside Data Cleaning, **Data Reduction and Discretization** is the second point, which this work focuses on. The KDD process is used with a practical target background. Therefore it is extremely important to find a balance between quality and speed of the whole process. Knowledge which is not available when it is needed is useless. So the calculation for data mining has a performance requirement, that has to be fulfilled by using existing hardware. Reduction has two faces - lossless (e.g. compressing the data with a zip-algorithm) and lossy (e.g. compressing images using the JPEG algorithm). The difficulty on data reduction and discretization is to reduce the amount of data by a minimal (or no) loss of information. Dimensional reduction (using not all available attributes for data mining) reduces the complexity in a massive way, but also drops information. Sampling is another method for reducing the data by using a subset that has "similar" feature values like the whole dataset (e.g. same standard deviation) and produces a comparable analytical result in the data mining task. The principle component analysis (PCA) was chosen as one demonstration method of dimensional reduction. In this thesis, data cleaning and data reduction use cases are covered in more detail.

2.3 Related Work

There are two emphasis of this thesis - statistical methods and memory management. Both are fundamentally illuminated on the following pages.

2.3.1 Statistic principles

In the knowledge discovery process, statistics, data preprocessing and data mining are not handled separately. It is mandatory to understand how they fit together and how they are used in combination. Therefore a major point is to understand the basic statistics considered in this diploma thesis.

Statistic location parameters

The simplest method of calculating the center of data is the usage of the arithmetic mean, where all values are calculated together and then divided by their frequency.

$$\frac{1}{n}\sum_{i=0}^{n} x_i$$

One advantage is that the arithmetic mean is easy to calculate, but the problem is the great drift concerning outliers. This means, that the arithmetic mean is sensible to outliers. Thus a better solution is to use the Median, which is much more robust. It is defined as the value that separates the underlying population into a higher and a lower half. If the amount of values is odd, the value with the center index is taken. If the amount is even, the arithmetic mean of the values next to the center is calculated.

$$x_{med} = \begin{cases} x_{\frac{n+1}{2}} & \text{n odd} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right) & \text{n even} \end{cases}$$

The amount data values are tending to spread or, on other words, how much they differ from one to each other, is called dispersion of variance. Variance is a very important attribute of the data that is put into the cleaning task of data preprocessing. It gives an idea how noisy data is identified. This statistical dispersion can be measured by a number of different statistical methods like variance, standard deviation and interquartile range. The measurement itself gives a value that is zero, if the data is identical. It increases if the spread also increases.

One basic measurement for dispersion is the variance. If variable X (random) has expected value E(X) (arithmetic mean), the variance Var(X) of X is defined:

$$Var(X) = \sigma^2 = E[(X - \mu)^2]$$

For a better understanding, Figure 2.8 illustrates two distribution curves with the same amount of expectation value but different variance.

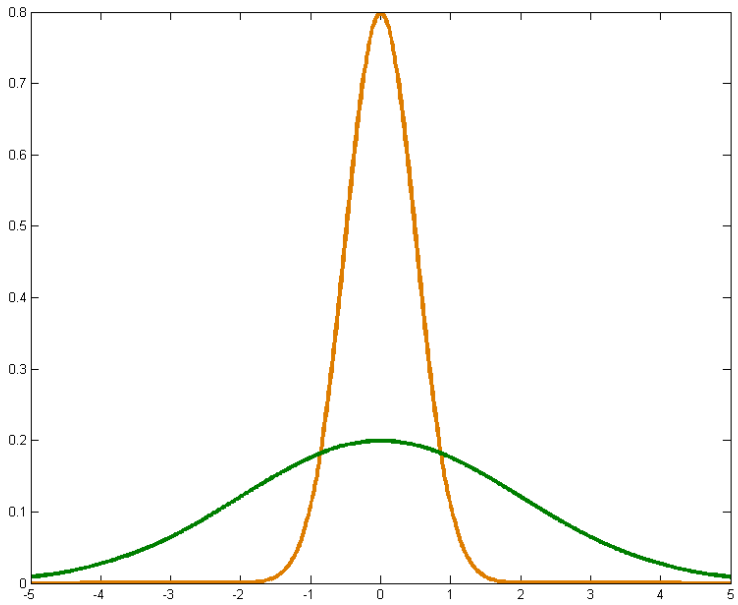

Figure 2.8: Distributions with the same expected value E, different variance [57]

The standard deviation is used to measure the dispersion of a data set. It is labeled with the Greek character small sigma σ and defined by the spread of the single values from the mean. That's why it is also known as the root mean square (RMS) because the summed up differences from the mean are squared and then the root is taken.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(x_i - \overline{x})^2}$$

Figure 2.9 demonstrates this relation in a graphical way.

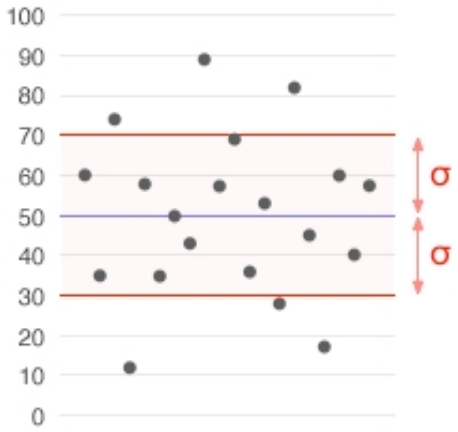

Figure 2.9: Illustration of the standard deviation [56]

Normal distributed values can be completely described by their arithmetic mean μ and the standard deviation σ. Values that are outside two or three times the standard deviation could be treated as outliers. Outliers can be indicators for errors in the data recording step of the KDD. However, also as interesting patterns.

In the case of data preprocessing, it is not very useful to store each value temporarily and then calculate the standard deviation when working on huge datasets. This problem is caused by the term

$$\frac{1}{n}\sum_{i=0}^{n}(x_i - \overline{x})^2$$

which blocks an on-the-fly updating because the arithmetic mean $\overline{x}$ does not stay constant and has to be processed on each incoming line. For the implementation in the statistics activity, a formula was used which allows sequential updating of the standard deviation value.

When applying the following proposition

$$\sum_{i=0}^{n}(x_i - \overline{x})^2 = \left(\sum_{i=1}^{n}x_i^2\right) - n\overline{x}^2 = \left(\sum_{i=1}^{n}x_i^2\right) - \frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)^2$$

a formula can be used for a much more elegant calculation

$$\sigma = \sqrt{\frac{1}{n}\left(\sum_{i=1}^{n}x_i^2\right) - \left(\frac{1}{n}\sum_{i=1}^{n}x_i\right)^2}$$

For visualizing the spread of data, a boxplot is used. For its creation the following values have to be determined.

- upper quartile (Q3)
- lower quartile (Q1)
- median (=Q2)
- interquartile Range (IQR)
- the minimum value (optional)
- the maximum value (optional)

The median defines the center of the data, the box itself is represented by the upper and lower quartile. Connected to this quartiles, the so called whiskers are attached. There are different variants for drawing the whiskers. While it is possible to use the minimum and maximum value as limits, the length of the whiskers is often limited by the maximum of the $1,5 \cdot IQR$ (=interquartile range). It is defined as the difference between the upper quartile (Q3), written as x_{75}, and the lower quartile (Q1), and written as x_{25}.

$$IQR = Q3 - Q1 = x_{75} - x_{25}$$

Values that are between $1,5 \cdot IQR$ and $3 \cdot IQR$ are called "mild" outliers. Values above $3 \cdot IQR$ are called "extreme" outliers. The Figure 2.10 shows this statistical context in a graphical way.

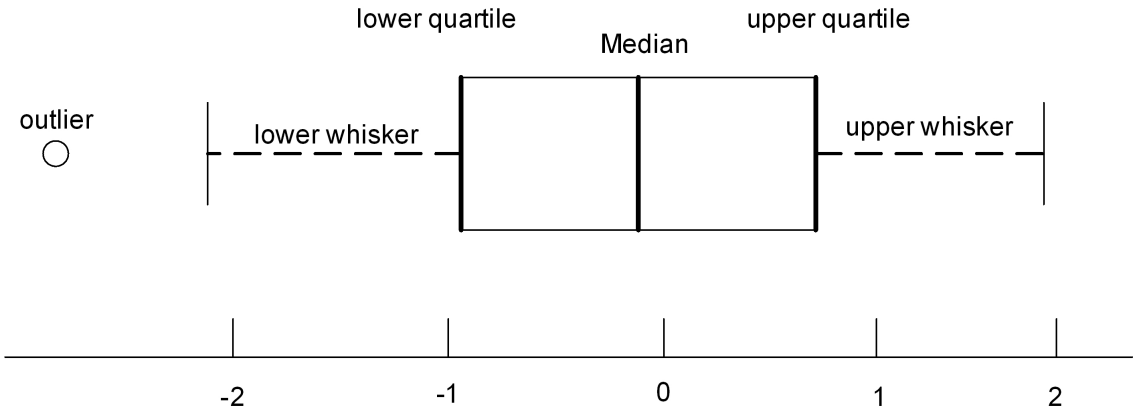

Figure 2.10: Boxplot illustrating the whiskers and quartiles

The so far presented statistical functions are important in the preprocessing phase of the KDD process to extract knowledge about the distribution of the data. Due to operating on a huge dataset with multiple dimensions (lots of features, that have to be taken into account) the reduction of the complexity is important to save computing power and also storage space. One method to do so is called Principle Component Analysis (PCA) which is described in detail in the next section.

PCA - Principal Component Analysis

The PCA is used for reducing high dimensional datasets by exchanging a big amount of weak (less relevant) features by lesser stronger principal components. This is done by a vector transformation of the current variable space into a new one, rotating the coordination system and calculating the impact of every variable to the complete information. After this, a level of the wanted final fidelity (relevance) is defined. Below that, the variable is dropped. As an example - if it is defined that 90% of the original data are sufficient, then it can be that this rule is fulfilled with only 4 of 10 dimensions (also known as “features” or “columns” of a relational table). That would reduce the complexity about 60%. In the resulted vector space, further data preprocessing methos can be done.

For demonstrating this principle in a practical way, an example from a PCA tutorial [41] of two dimensions will be used. The implementation details on the PCA in terms of this diploma thesis could be found in the implementation chapter.

As already mentioned, the following expressions are based on this example data:

Table 2.1: Example data for PCA demonstration taken from tutorial [41]

x	y
2,5	2,4
0,5	0,7
2,2	2,9
1,9	2,2
3,1	3
2,3	2,7
2	1,6
1	1,1
1,5	1,6
1,1	0,9

For a better understanding, these values have been inserted into a diagram, shown in Figure 2.11.

At the beginning of the Principle Component Analysis (PCA), it is needed to subtract the mean from the original data, which results in a transformed table of the data:

In Figure 2.12 it can be seen, that this step of the PCA gives a table of data with an arithmetic mean of zero.

After this step it is necessary to calculate the covariance matrix, which is just a matrix of all possible covariances combinations of the columns.

$$C^{m \times n} = (c_{i,j}, ci, j = (Dim_i, Dim_j)) \tag{2.1}$$

In the case of this example, it results in a 2x2 matrix.

$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(y,x) & \text{cov}(y,y) \end{pmatrix} \tag{2.2}$$

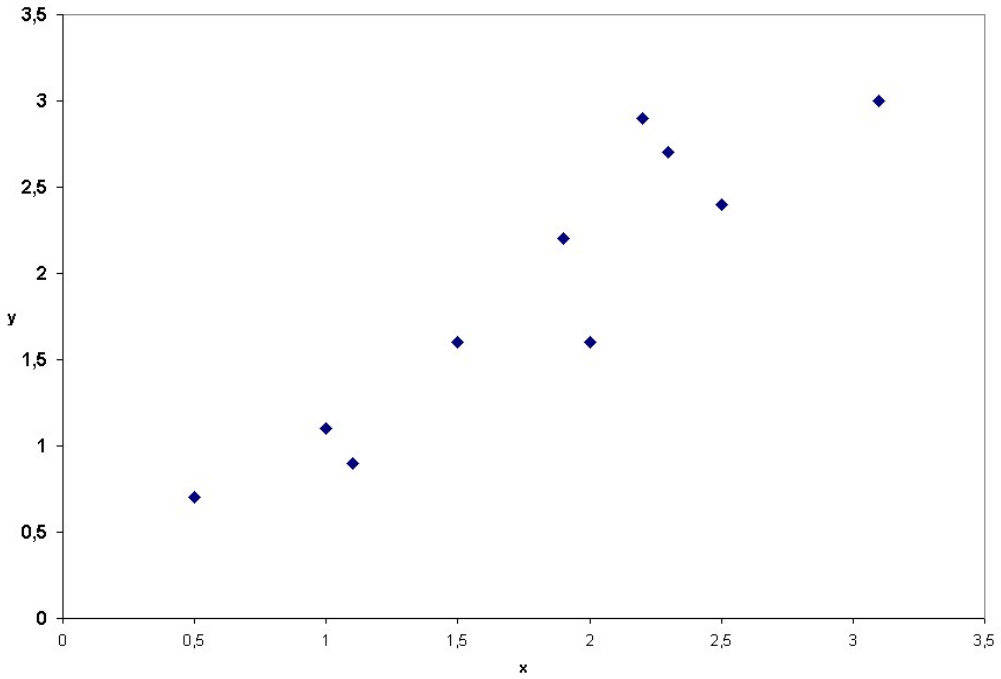

Figure 2.11: Original example data

Table 2.2: Transformed table of data

x	y
0,69	0,49
-1,31	-1,21
0,39	0,99
0,09	0,29
1,29	1,09
0,49	0,79
0,19	-0,31
-0,81	-0,81
-0,31	-0,31
-0,71	-1,01

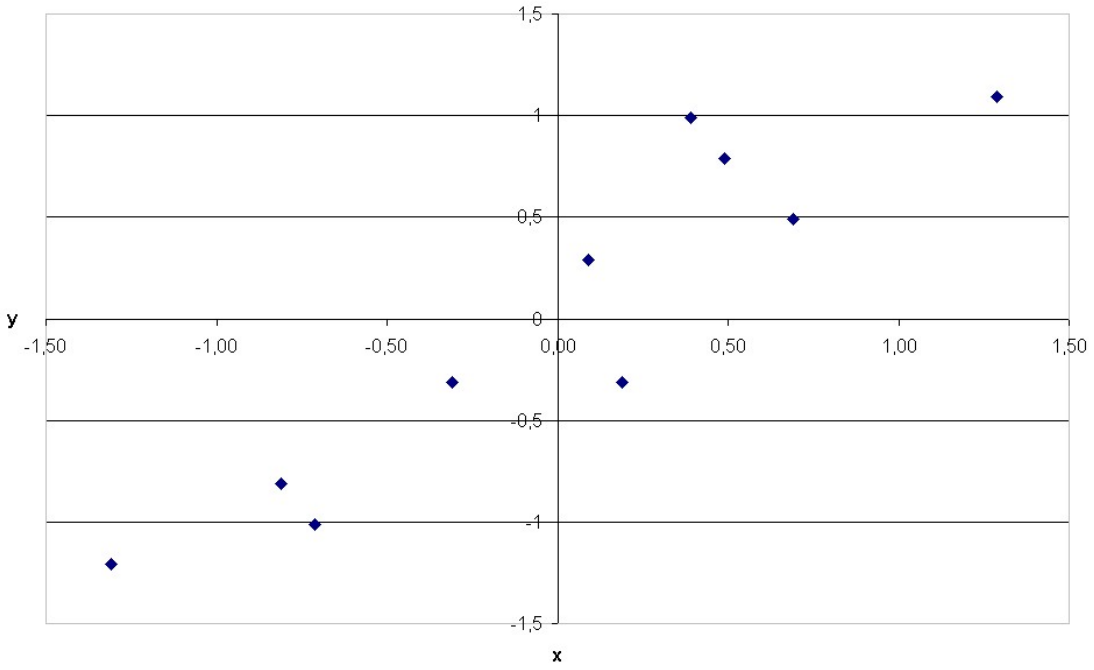

Figure 2.12: Transformed example data

By using the already known formula for the covariance, the following values were calculated:

$$\text{cov} = \begin{pmatrix} 0,616555556 & 0,615444444 \\ 0,615444444 & 0,716555556 \end{pmatrix} \tag{2.3}$$

The next step is to compute the so called eigenvectors by the following definition [49]:
Consider the square matrix A. We say that λ is an eigenvalue of A if there exists a non-zero vector x such that Ax = λx. In this case, x is called an eigenvector (corresponding to λ), and the pair (λ,x) is called an eigenpair for A.

$$eigenvectors = \begin{pmatrix} 0,677873 & -0,735179 \\ 0,735179 & 0,677873 \end{pmatrix} \tag{2.4}$$

The eigenvalues of the covariance matrix are calculated by solving the equation in [49] and can be seen below here:

$$eigenvalues = \begin{pmatrix} 1,28402771 \\ 0,0490833989 \end{pmatrix} \tag{2.5}$$

They also correspond to the expected value. With the knowledge of the eigenvalues, it is possible to sort the eigenvectors concerning their impact on the data, which is first done by the creation of the feature vector. Therefore, it can be seen, that with the first eigenvector (the first column from

matrix 2.4)

$$\begin{pmatrix} 0,677873 \\ 0,735179 \end{pmatrix}$$

it is possible to describe 96% of the data, while the second only provides 4%. This means, that the second dimension (the second vector) could be ignored, if 96% is sufficient.

$$FeatureVector = (eig_1, eig_2, eig_3, eig_4.....eig_n) \tag{2.6}$$

At the end, the final data is calculated by matrix multiplication of the row feature vector with the transformed data values from Table 2.2.

$$FinalData = RowFeatureVector \times RowDataAdjust \tag{2.7}$$

Because 96% of the data can be described only by one dimension, in this example, the two-dimensional feature vector is choosen. In a concrete dimensional reduction, only the first column would be relevant and be taken for further calculations. This final result of the data by a two-dimensional feature vector is in Table 2.3 and drawn into the diagram of Figure 2.13.

Table 2.3: Table of final data

x	y
0,82797008	-0,17511574
-1,77758022	0,14285816
0,99219768	0,38437446
0,27421048	0,13041706
1,67580128	-0,20949934
0,91294918	0,17528196
-0,09910962	-0,34982464
-1,14457212	0,04641786
-0,43804612	0,01776486
-1,22382062	-0,16267464

2.3.2 Memory management

Storage levels

One central point of this thesis is the basic meaning of the memory architecture of todays workstation PCs (lots of server systems are also based upon this structure). It has a great impact where to store data during processing and where to store the results. The total available memory of a PC can be split into four main sections which mainly differ of their access time and storage capacity:

- Primary storage (CPU and main memory)
- Secondary storage (hard disk)

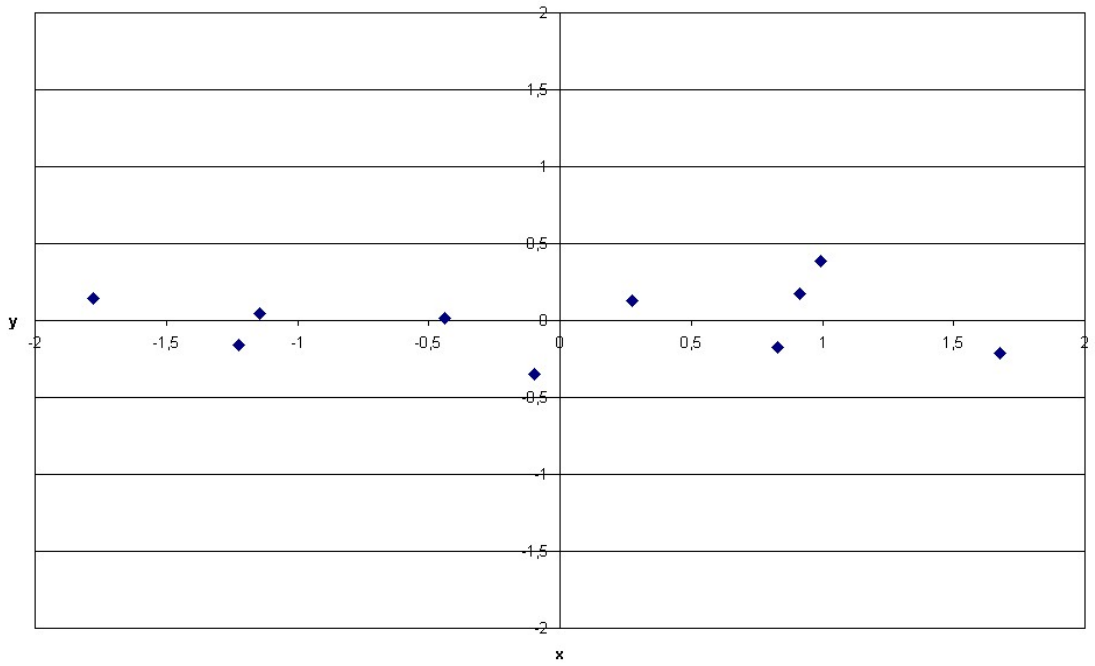

Figure 2.13: Final calculated example data

- Tertiary storage (disk changer)
- Off-line storage (removable media)

Figure 2.14 gives an overview of the different storage types and how they are connected.

The primary storage is mostly located on the mainboard. Here again, a separation between internal CPU memory and external CPU memory is done. The internal memory is located inside the CPU and is divided into a first level (L1) and a second level cache (L2). The first level cache in actual Xeon workstation CPUs is about 32KBytes size and the slower second level cache, that buffers data from and to RAM (main memory) has a size between 2 and 12 MBytes. While the first level cache operates on the clock frequency of the core (e.g. 2 GHz), the speed of the second level cache can vary (e.g. half speed).

The main memory is attached next to the CPU via the so called Front Side Bus (FSB) which runs at about 400 MHz. Because data is transferred in parallel, this 400 MHz clock speed results in a maximum transfer rate of 10 GBytes/second on current available Intel CPUs. Its size is about several hundred Megabytes to several Gigabytes (workstations are often equipped with around 8 Gbytes and more of RAM). Also the operating system has to handle the maximum physical size. That's why often a 64-bit operating system is a must have in high performance computing field. The access time of the systems main memory (around 10 nanoseconds) is slower than the internal memory.

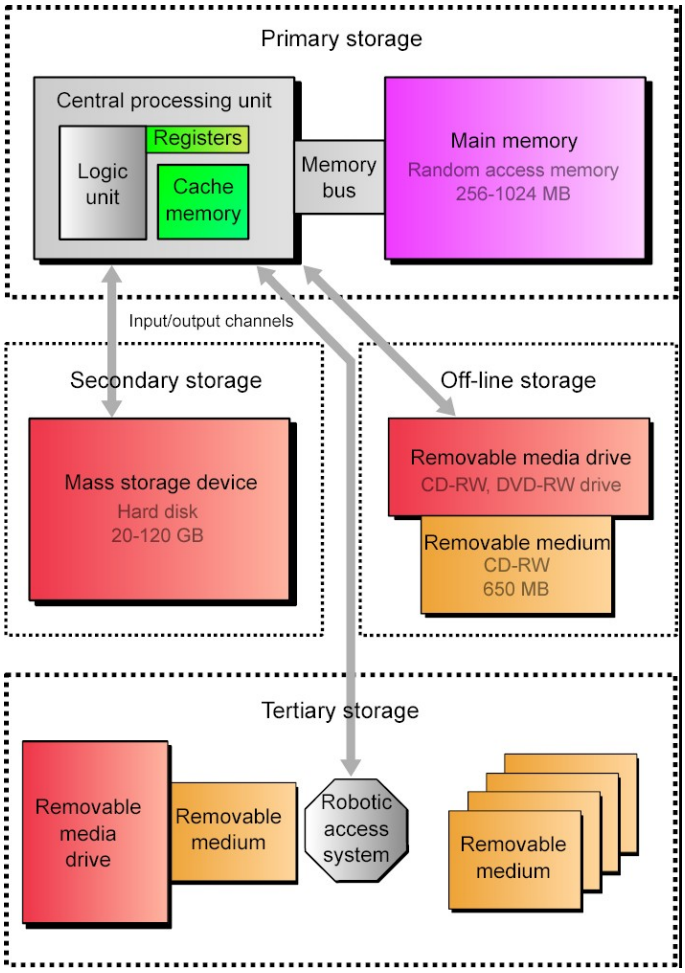

Figure 2.14: Storagelevels of a typical workstation PC [55]

The secondary memory is usually associated with hard drives. They are attached via a workstations internal (e.g. SATA, SAS) or external interface connections (e.g. USB, Firewire) to the CPUs. Their access time is about six orders of magnitude higher (milliseconds) than the access time to the main memory. The transfer speed is around 40 to 90 MBytes/second. The big advantage is the capacity which is between 200 GBytes and 1 TBytes per hard disk. The upcoming Solid State Drives (SSD) are seen as the successor of the current hard drives, because they do not consist of any mechanical parts, which gives them a much shorter access time. At the moment SSDs are very expensive and only available till sizes at 250 GBytes. Another advantage of the secondary memory technologies beside its capacity is that it can be used for persistent storage. The data is stored permanently without the need for electrical power, whereas Primary memory losses its data after turning off the workstation.

Tertiary storage is the storage type with the biggest on-line availability combined with great scalability. The disadvantage of this type is its very high access time. The transfer speed reaches nearly the dimension of secondary memory (in high performance systems between quarter and half speed of secondary memory is reached). Examples of tertiary storage are NAS (Network attached

Storage), disk changers, etc.

The off-line storage can be seen as a tertiary storage, that needs human interaction for memory access. Every removable media, like CDs, DVDs, etc. need additional effort to "mount" them in order to get available for the CPU. The big advantages of off-line media are the persistent storage, the robust design and the low storage costs which makes them the perfect choice for archive use cases.

OS memory management

The Stack is a data structure that is often compared to a stack of books. It supports two kinds of operations - push and pop. The operation "push" puts a data element on the stack, while the operation "pop" returns the data element that is the first on the stack (i.e. the last inserted) by removing it from the stack. That's why the behavior of the stack is also defined as LIFO memory - Last In, First Out. This principle is very useful when using nested syntaxes (e.g. XML, function calls). In microprocessor technology (the Java virtual machine could be seen as an emulation of a specific processor), the stack is used for storing local variables and handling function calls. If a function is called the address of the calling function is put on the stack as a return address for the called function.

The Heap is a memory space that can be used by a program to allocate dynamic memory. The program requests data on a heap and after a successful reservation, it is allowed to use it. If the storage space on the heap is not needed anymore, then it has to be freed. On the stack, the reserved storage has to be freed in the reverse order, that it was reserved. That's why the stack memory handling is also called "automatic memory allocation". Compared to the stack, the time for reserving memory on the heap takes noticeable longer than on the stack. If the program does not free unnecessary memory on the heap, so called memory leaks can be caused if the reference to the reserved space is lost.

Java VM memory handling

To avoid memory leaks in Java, the Java VM has a garbage collector (GC) that does the cleaning of the memory automatically (also to avoid memory fragmentation). This is a big advantage, but the fact that it operates automatically in a separate thread (since Java 1.4) makes it difficult to take control over the object's life circle in special cases (e.g. caching algorithms). The Java garbage collector calls the method "finally", which is available in every object and can be overwritten for own purposes. The finalize method is only called once by the garbage collector. If the object is prevented from being garbage collected (e.g. creating references in the finalize method), then it is not garbage collected again. The "new" operator in Java is used to put a variable on the Heap and returns a reference to it. The variables are accessible across Java threads. If a variable is not referenced from anywhere, then it is allowed to be garbage collected. Another possibility is to assign "null" to the reference variable.

The Java virtual machine is using its own memory management for a better abstraction level, when running on different operating systems. This also gives the possibility of garbage collection of the virtual machine, even if the underlying OS does not support this.

The available memory space of the virtual machine is split into different sections:

- eden space
- survivor space
- tenured generation
- permanent generation
- code Cache

Figure 2.15 illustrates this splitting.

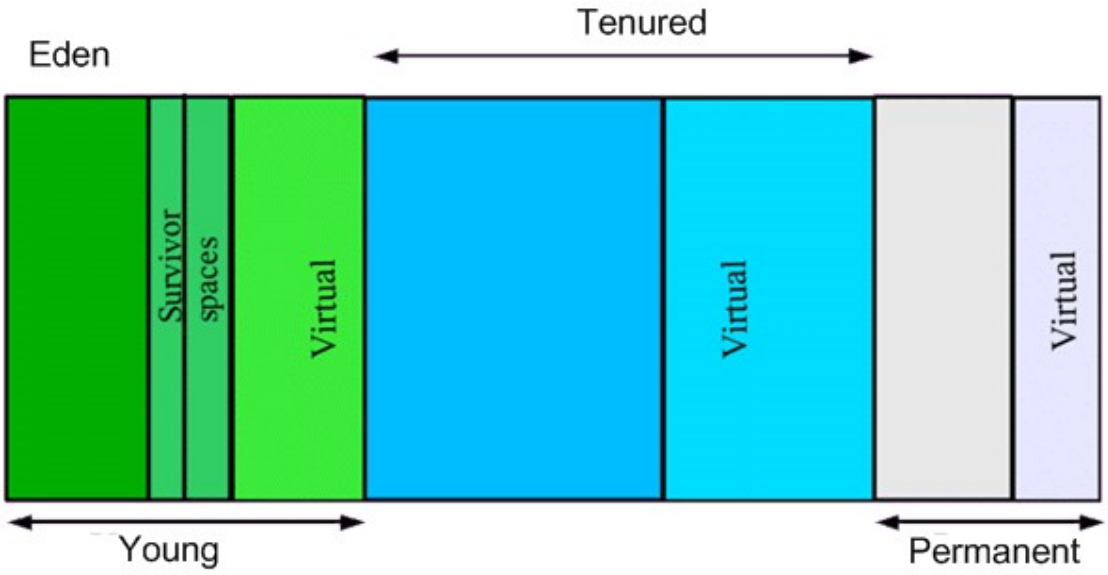

Figure 2.15: Memory architecture of the Java Virtual Machine (JVM) [47]

Sun defines the "Virtual" memory space the following way: *At initialization, a maximum address space is virtually reserved but not allocated to physical memory unless it is needed* [47]. Most of the objects are initially created in the **eden space**. When the garbage collector cleans this first section of memory, the objects that were not removed are transferred to the survivor space. The advantage of doing so is that the garbage collector does not have to check the objects every time. So the eden space is foreseen for short-term living objects.

The **survivor space** consists of the objects, that survived the first memory level. The garbage collector checks the survivor space with a lower frequency than the eden space for saving system resources. That's why the survivor space has the task to store median term living objects for a

fixed defined time. After this, the garbage collector transfers them into the tenured generation memory, if they are still used.

The long term living objects are existing in the **tenured generation** space and survived the two previous levels of garbage collection. The tenured generation space is scanned rarer than the survivor space, because it is not necessary for long term living objects.

The space that is reserved for the virtual machine itself is called **permanent generation** space. Here all used classes, methods and data are stored, that are permanently used during execution. If the virtual machine supports the feature for dynamic data exchange, the permanent generation space is separated again into a rewritable and a read-only area.

The Hotsport JIT (Just-In-Time) compiler has its own memory space called code **cache space**. This is the place that it uses for the compilation of byte code into native code for performance improvements.

Soft references

For the requirement of implementing some caching strategy to the novel WebRowSet implementation, a special type of references was used, to allow objects to automatically being garbage collected, although there is still an existing reference. This type of reference is known as SoftReference.

Suppose that the garbage collector determines at a certain point in time that an object is softly reachable. At that time it may choose to clear atomically all soft references to that object and all soft references to any other softly-reachable objects from which that object is reachable through a chain of strong references. At the same time or at some later time it will enqueue those newly-cleared soft references that are registered with reference queues.

All soft references to softly-reachable objects are guaranteed to have been cleared before the virtual machine throws an OutOfMemoryError. Otherwise no constraints are placed upon the time at which a soft reference will be cleared or the order in which a set of such references to different objects will be cleared. Virtual machine implementations are, however, encouraged to bias against clearing recently-created or recently-used soft references [44].

That's why soft references are ideal for implementing caches. In the novel WebRowSet component, the SoftReferences have been used to implement write-through cache represented by a newly implemented sorted tree map.

Java and evaluation strategy

One important thing when operating on huge structures in memory is to know how these data is passed to a function that is used for manipulating it. In programming languages there are some different paradigms (evaluation strategies), that also depend on the used language. In this section, only the most important ones are discussed.

The first possibility is the ***call-by-value method***, where the parameters are passed as copies from the caller to the callee. All changes the callee is makes on the data are only appearing on its local copy and not on the original data the caller holds.

In contrary as another possibility is the ***call-by-reference*** function call. Here the parameters are passed as references to the callee. Because the callee can operate on the original data, every change can be seen on the level of the caller too.
In Java, only call-by-value is used, but the behavior differs concerning the variable types that are passed to the subroutine. If simple data types like int, boolean, char, double are passed it seams that call-by-value is used and on the other hand, when complex data types (objects) are passed, call-by-reference is used. In fact Java passes complex types by passing a copy of their reference. That's why the subroutine is able to modify the object although call-by-value was done.
This knowledge is very useful, because on main aspect of the preprocessing software components is to use as less memory as possible by avoiding unnecessary copying of data.

Chapter 3

Advanced Data Preprocessing on the Grid

The term “advanced” in the context of preprocessing on the Grid means to create the additional statistical metadata directly on the Grid and provide it to the preprocessing client application. This approach called D3G was first presented in a paper of Alexander Wöhrer, et al. [59] and its further development is provided in a much more fine-grained way in this diploma thesis. The chapter starts with a description of the use case scenarios. After this, the chapter introduces the novel indexing method of WebRowSet files and the generation of metadata statistics. Then, the design criteria and requirements are presented. Finally, the resulted software architecture is described for the modern WebRowSet component part and the metastatistics component part.

3.1 Use-Case Scenarios

As described earlier, OGSA-DAI represents some form of a workflow engine by using a pipes & filters pattern to connect activities together as a chain. The standard workflow is to access a data source (e.g. a database) and use the result as an input for one or more activities. The WebRowSet file is created as the final destination. The novel method is to create useful statistics of the dataset during the dataflow inside the OGSA-DAI framework and provide this metastatistics at delivery state for later data preprocessing.
The newly designed statistics activity is used in a parallel way. That means the Inputstream needs to be copied before it is fed into this activity. The activity provides one input for specifying the wanted statistics column by column. So it is possible to define which statistics are calculated on which columns. The second input is used for the Tuples on which the calculations are made on. The output of the statistics activity gives the calculated metadata in the XML format. The supported statistics are minimal value, maximal value, average, frequency, median and standard deviation. How the WebRowSet interface is used together with statistical metadata can be seen in Figure 3.1. The figure demonstrates the creation of the statistics in parallel based on the same

data resource (the database). The are available as soon as the WebRowSet creation is finished and can be used before processing the WebRowSet file by the client application.

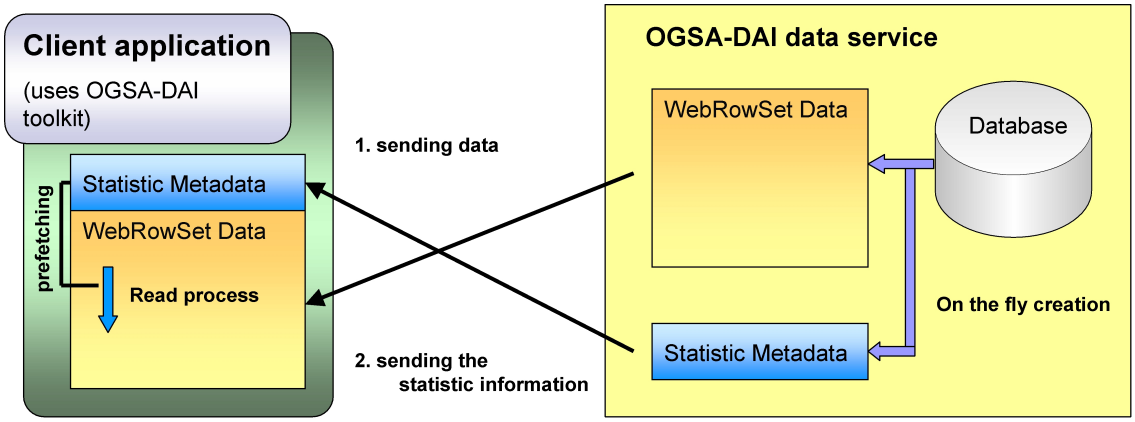

Figure 3.1: The usage of statistical metadata and WebRowSet

After a query on the database was executed successfully, the result set is transformed to a datastream that is used to create a WebRowSet file. During this, the statistical metadata are created by using a copy of this stream in parallel. With this method, the client can first access metadata without loading and processing the whole WebRowSet file. The client can also benefit from the statistical calculations during data preprocessing (e.g. it can directly fill in missing values by using the median).

There are four main use cases that are the basis for determining the requirements of the novel developed components. The WebRowSet XML format is used as a persistent storage format for a disconnected RowSet object. Figure 3.2 shows how an existing WebRowSet file is extended by adding additional rows or deleting existing rows. The workstation runs an application that provides an interface for WebRowSet files.

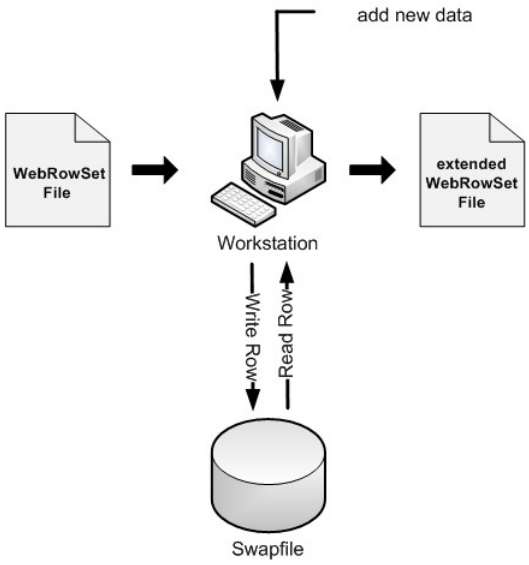

Figure 3.2: Extending an existing WebRowSet file

Because of the memory limitation, a swapspace is used (represented by the "Swapfile" storage) to carve out the changed data. For this operation, the row objects have to be serialized to Strings for storing them in a swapfile and de-serialized from the swapfile back to objects for using them in an application at a later time. If the Client is disconnected from the database (e.g. maybe due to mobility reasons), the only data resource is the WebRowSet file. Therefore it is needed to modify it.

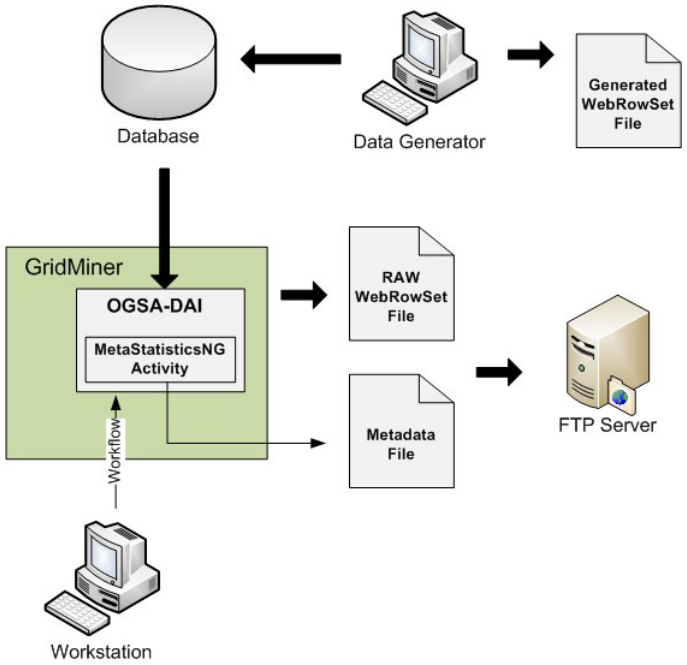

Figure 3.3: Delivering metastatistics and WebRowSet file to an FTP storage

Figure 3.3 illustrates the definition of an OGSA-DAI workflow, which creates a WebRowSet file

and uses the metastatistics activity, called "MetaStatisticsNG activity" ("NG" stands for Next Generation) to produce an XML file that contains the statistical results, that were defined by the workflow. Both files have to be transferred to an FTP storage. The FTP storage decouples the data from the OGSA-DAI workflow. The data are created by a generator instead of using real world data. Using a generator provides a more flexible development of new KDD approaches. In our test environment, the generated data are written synchronously to the database and into a WebRowSet file. This allows to use the same data without OGSA-DAI.

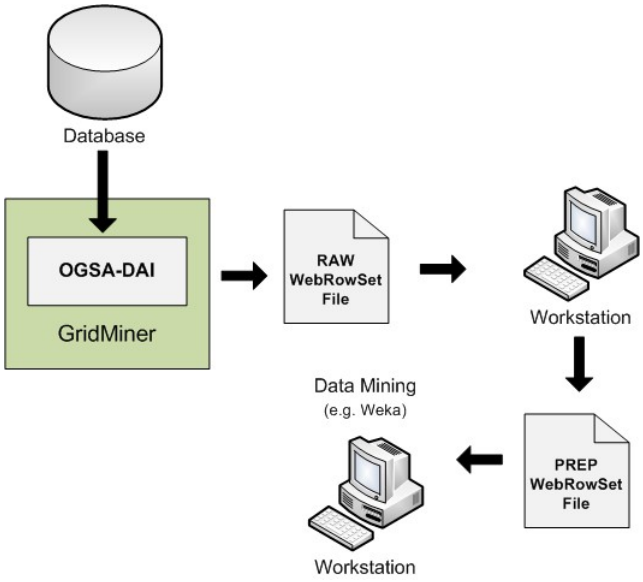

Figure 3.4: Inter-exchange of a modified WebRowSet file

How the WebRowSet file format is used for exchanging data between different applications is demonstrated by Figure 3.4, where the preprocessing client on the workstation is used to process the incoming data from the OGSA-DAI framework and creates a new WebRowSet file, that can be used by another application (e.g. WEKA for data mining). It is mandatory to guarantee a compatible creation of WebRowSet files in the whole chain of this use case process.

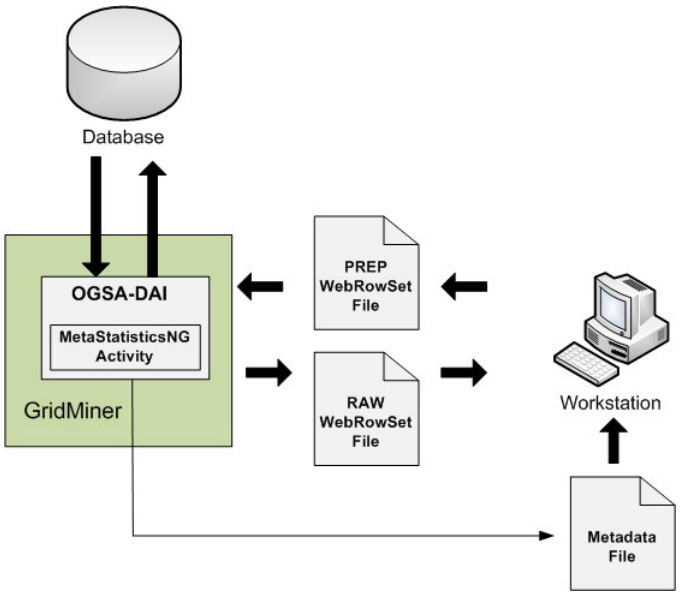

Figure 3.5: Workflow interaction with OGSA-DAI

The main use case is the cooperation between a workstation and the GridMiner, that runs the OGSA-DAI server. Figure 3.5 pictures the interaction of a workstation, that is used for data preprocessing with the OGSA-DAI framework and expresses the workflow in both directions.

3.2 Novel Indexing Method of WebRowSet XML Files

The indexing of XML documents shows similar problems that are found when indexing relational or object oriented databases [19]. Storage space is used to keep track of the data item location. This allows shorter access times. Of course there has to be found a balance between the amount of storage space that is needed and the faster access. The structure of the data items could be indexed as well as the values itself.

XML documents provide a standard structure that can be used for navigation (e.g. element - sub element), but for navigation through other dimensions (e.g. attribute values) a new structure has to be build for a faster access. Otherwise the whole given structure has to be parsed, which can cost a lot of time. Relational databases provide their navigation by joins, where two ore more tables are linked together by their so-called “key fields”. That is why often the building of a data structure is done that allows fast access to key-value pairs for structural navigation. In object oriented databases, the data items are represented by objects that have interconnection with others. Because the object-oriented database language defines these connections and not a query language like SQL, the methods for indexing their structure is quite different. XML indexing should be able to handle a very flexible structure of the data elements. This is usually done by creating binary link relations between two attributes. The most important indexing strategies are:

- **No indexing**
  When data is changing rapidly and the amount of data items is low, it could be an advisable

way to skip indexing. This is because the time of building the index could be above the access time. On every change it is needed to rebuild the index.

- **Complete indexing**
  If a dataset is accessed, that is not often changed, a complete index drops the access time for each query. Also an index concerning the relationships between documents can increase the performance.

- **Partial indexing**
  The idea of partial indexing is to reduce the amount of the needed indexing. This means that indexes are only created on parts of the data, that are frequently accessed by queries, and that are more time consuming than other queries. For choosing an appropriate indexing strategy it is mandatory to find the queries that are likely to be asked. This could be done by starting at a complete indexing with a search and then reducing the unnecessary indexes.

Figure 3.6 displays the indexing methods, that are used on relational table based data structures. The example table in this figure consists of four columns and four rows. Indexes are used to reference the cell data. There are two different index approaches - row-based and column-based. The column-based method uses a separate row index for every column (the figure shows the four different index storages). The advantage is that only the affected column indexes have to be used for executing a query. This reduces the required processing effort and in context with DataBase Management Systems (DBMS) a lower memory consumption (compared to the row-based approach) during the construction of the query result - the result set. The row-based method uses a single row index for the whole table. The advantages are the easier handling of data manipulation on row-level (in the column-based approach all index storages have to be modified if a row is inserted or deleted) and the less required memory consumption for the index data. In context of this work, the based WebRowSet XML file is more comparable to a table, because it does not use attributes in the row and column elements. Therefore it is important to imagine the differences between an approach, where columns are addressed compared to the possibility of accessing the data on a row level. The main benefit of the column-based approach (data reduction during query processing) cannot be used in context with WebRowSet files, because they already are results of queries.

Because of the special XML tree of WebRowSet files, a specialized indexing algorithm was used. The first approach was done by an earlier thesis of Michaela Pfeifer [38], where a column based indexing was chosen. The advantages of a column based indexing is the direct access to each cell, which increases the speed. The disadvantage is the memory consumption. Figure 3.7 demonstrates this correlation. In the case of data mining, the dataset can consist of multiple features, that are organized as columns. If so, the data is called “multi-dimensional”. Some datasets consist of ten to hundreds of features. Then a column-based way produces $indexes\ data = rows \cdot columns$. This explains, why the modern implementation uses a row-based method.

The new implementation of the WebRowSet interface uses the overture from Michaela Pfeifer’s diploma thesis and the paper of the Index Based Parser [61]. One major difference is that both earlier concepts only stored file pointers, while the realization of this thesis uses a snapshot status of the whole SAX parser, by using the Memento pattern [18] to store the private member values.

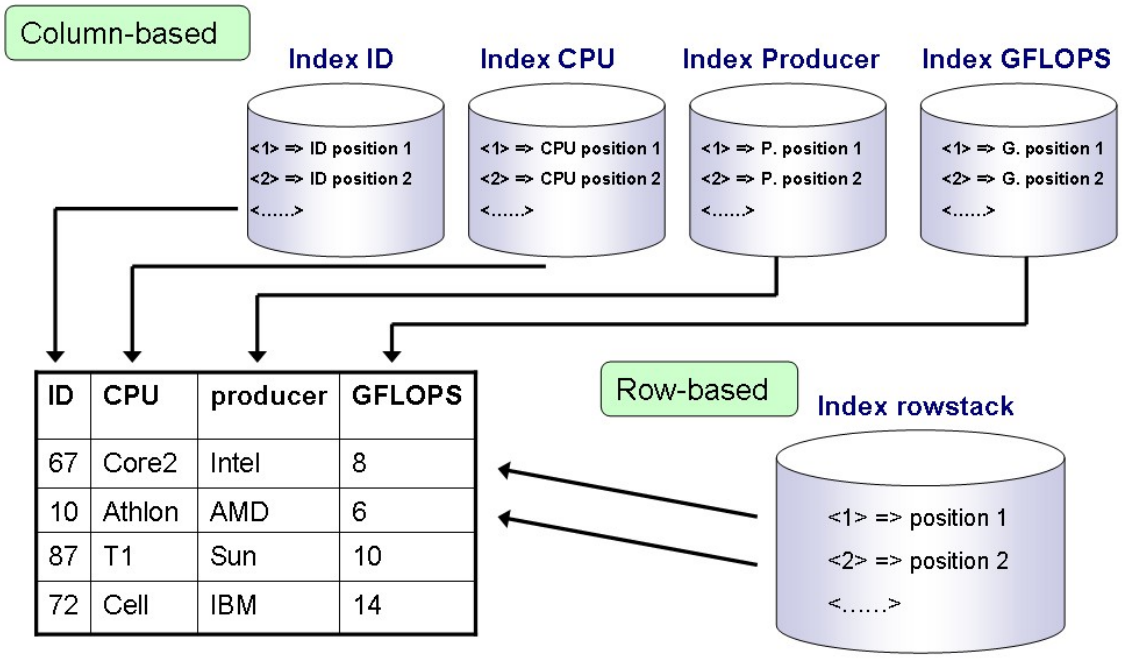

Figure 3.6: Column- and row-based indexing methods

3.3 New Preprocessing Method

The novel part of the preprocessing method is to use precalculated statistics for optimizing the preprocessing task. The preprocessing application runs on a standard workstation and queries a dataset that has to be preprocessed by the Grid. The Grid itself provides access to the data via the OGSA-DAI framework and has much more computing power compared to the workstation. It is more efficient to calculate statistics based on the query on server side and provide them to the client workstation in advance. The client then can use this information (e.g. average value) directly during the data preprocessing without re-reading the whole dataset again. It is also possible that the whole dataset is dropped by the client if there are too much null values.

As it can bee seen in Figure 3.8 (page 41), the used OGSA-DAI pipeline consists of a Tee activity, that splits the Tuplestream from the SQLQuery activity into two identical copies of the stream. One stream is used for the transformation of the query result into a WebRowSet format by connecting it to the TupleToWebRowSetCharArray activity. The other stream is connected to the MetaStatisticsNG activity, where the statistical metadata is created. The output of the TupleToWebRowSetCharArray and the output of the MetaStatisticsNG activity is transferred to the DeliverToFTP activity where the sending of the data is done. In the statistical metadata file, the appropriate WebRowSet file is referenced by generating a Message-Digest Algorithm 5 (MD5) hash code from the SQL command to ensure a correct assignment.
The statistical metadataformat is represented by an XML file in a self defined format. This format has to be known to the preprocessing application, that uses the results of the MetaStatisticsNG activity. After this preprocessing application has downloaded the metadata file, it can decide about also downloading the corresponding WebRowSet file or modifying the query to the database to get

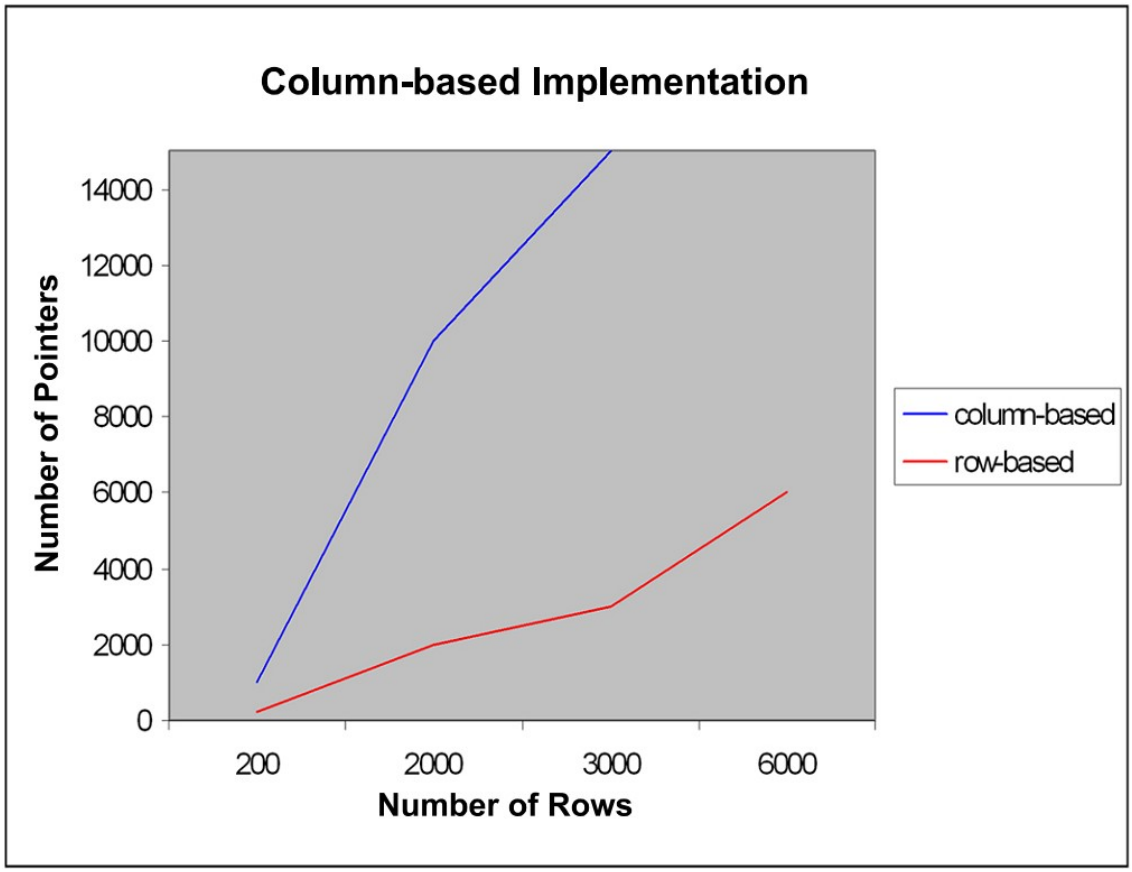

Figure 3.7: Comparison of pointers needed for a column-based and a row-based approach [38]

a new dataset. The other possibility is to use the available statistics for preprocessing operations like out of bound values (using the average value) or dimensional reduction (using the covariance).

The first step was to investigate different preprocessing methods to notice which statistical calculations they are using. In the second step the most relevant statistical functions were identified and filtered by their support of a continuously (iterative) usage. An important criteria of the function was that they are able to calculate them on the fly without further iteration steps. As an example, the calculation of the median requires sorted data. This fact assumes, that all data has to be held in memory, or a sophisticated algorithm has to be used for effective indexing and swapping the sorted values.

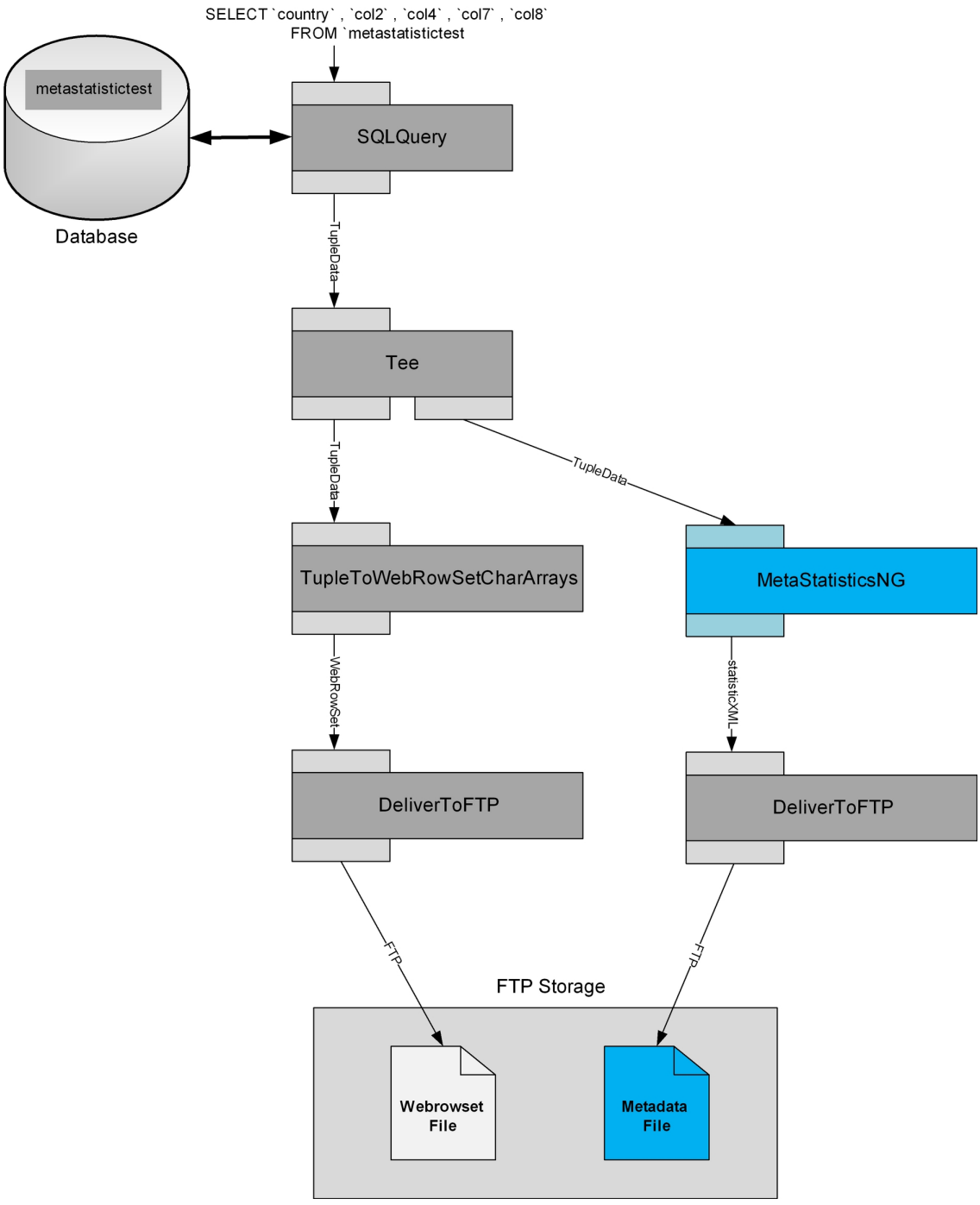

Figure 3.8: Workflow of calculating metadata statistics inside OGSA-DAI

3.4 Design Criteria and Requirements

The earlier discussed use case scenarios create a set of factors, that need to be taken into account. The factors determine the requirements of the software modules. The following sections contain the design concept of the developed software component parts to fulfill the requirements determined by the use cases. The general design criteria were:

- **strict encapsulation** of the data and methods of a class using Java OOP concept. The attributes inside a class are accessed by public methods.
- **logical decomposition** into modules with a well defined interface for connecting them together.
- **high functional cohesion** to collect methods in a dedicated module. The module should be able to process an atomic action or functionality.
- **low coupling** between the modules. Otherwise later changes of a single functionality require modifications in lots of modules.
- **standard based** method for interoperability with the OGSA-DAI framework and WebRowSet files.
- **scalability** for the usage on Grids concerning the used data size and functional range.

The design patterns described in the book "Entwurfsmuster" [18] from the Group of Four (GOF), often provide an elegant software design, create a common understanding of the architecture and support most of the listed design criteria. So I tried to adapt and integrate different design patterns to my architecture. The most relevant ones are described below.

Modern WebRowSet component part

The **functional requirements** of the software are to access and manipulate WebRowSet XML files with a size bigger than 90 MBytes. For the upper layer, a WebRowSet compatible interface should be provided.

The **non-functional requirements** are memory consumption and a scalable source code. The diploma thesis of Michaela Pfeifer [38] described an implementation of accessing huge WebRowSet XML files without the issues of an exceeding memory. My diploma thesis enhances this further approach by a new software architecture that allows backward navigation and caching of difference data (new inserted and modified lines). The parsing and the modification functions need an out-of-core design. The state of the SAX parser needs to be stored and restored in state objects. The design pattern "Memento" represents this method and is illustrated by Figure 3.9. The attributes of class, that represent the state of an object are saved inside a new created memento object. This object is created by instantiating an object from a memento class. The memento class is an inner class of the class, which object states have to be stored and restored.

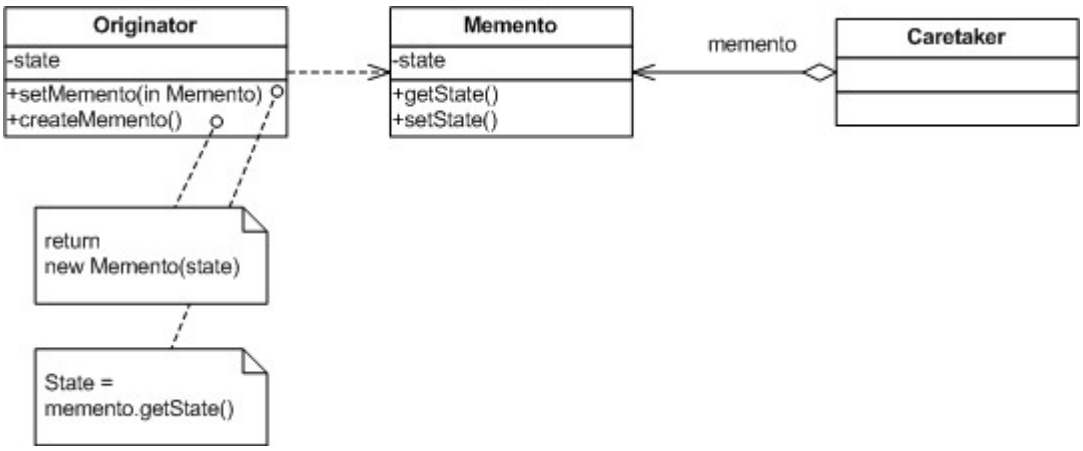

Figure 3.9: The Memento pattern [18, p. 318]

Caching and swapping to the file system is used for storing modifications to reduce the memory amount to a minimum. The amount of memory which is needed, is the size of a single Row (during modification) and the size of holding the index table, containing the memento objects of the SAX parser. Of course this indexes could also be hold by using a swapfile, but the size of these variables are small enough to handle these and avoid parsing a second index file. To enhance the read only WebRowSet file with changes (e.g. inserted rows, changed rows) a decorator pattern for the XML parser has to be used as it can be seen in Figure 3.10. The decorator enhances the function and attribute amount of a class.

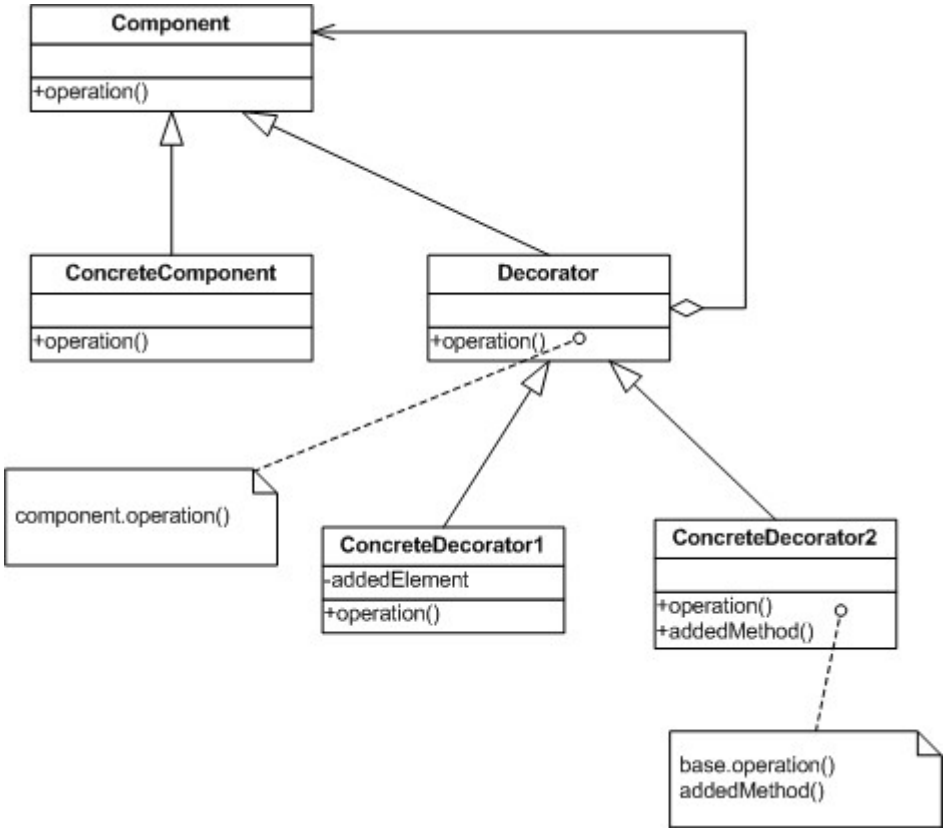

Figure 3.10: The Decorator pattern [18, p. 177]

The Java WebRowSet API defines the order of function calls for modifying the WebRowSet object. The "State Pattern" shown in Figure 3.11 allows a function handling dependent on the state without having huge "switch-case" structures inside the code. It is needed on insert and modification operations of the WebRowSet object. To add a new row, it is required to follow several steps. First, it is needed to call "moveToInsertRow()", then finishing the operation with "insertRow()" after the update function calls. Depending on the state of the insert and update process, the single function calls have a different behavior. That's the reason for using a state machine.

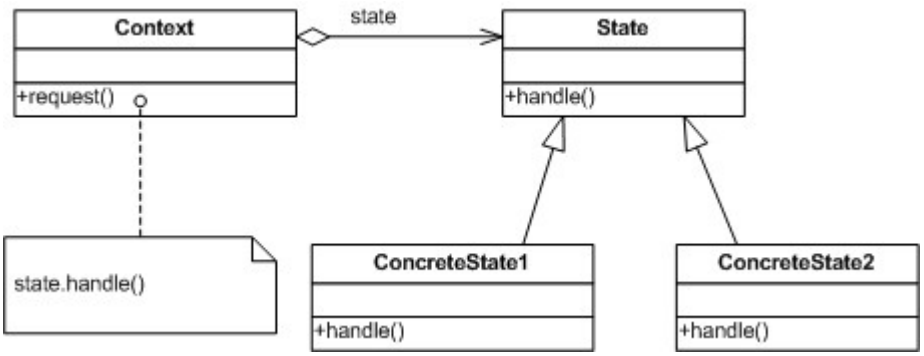

Figure 3.11: State pattern used for a state machine [18, p. 357]

The first objective was to provide a full implementation of the WebRowSet interface. As it can be seen in Sun Java API documentation [44], the WebRowSet interface inherits methods from four big Interfaces - CachedRowSet, Joinable, ResultSet, and RowSet by extending them. This is because the Sun implementation has to fullfil all possible use cases, an off-line and persistent RowSet class has. For the use case data mining and data preprocessing, WebRowSet is used as an exchange format. So the focus was set to read, modify and create WebRowSet compatible files. The WebRowSet interface itself is used for basic operations on a table of rows (e.g. insert, modify, read, etc.). Unlike the standard implementation of Sun, the modification tracking notations inside the WebRowSet XML file and transaction features (e.g. rollback, commit, etc.) were ignored, because of additional effort, which would have exceeded the frame of this work.

Advanced statistical metadata activity

The **functional requirement** of the OGSA-DAI activity is to calculate statistics of the input data and output them as XML files. The description of the column function map has to be represented in an open format.

The **non-functional requirements** are memory consumption, a scalable source code and processing speed. The selection of the statistical function by the client should be possible in a fine-grained way. This minimizes the costs for calculation, otherwise unwanted calculations on columns are done. The calculation should be done by statistic functions that provide a common interface as depicted in Figure 3.12.

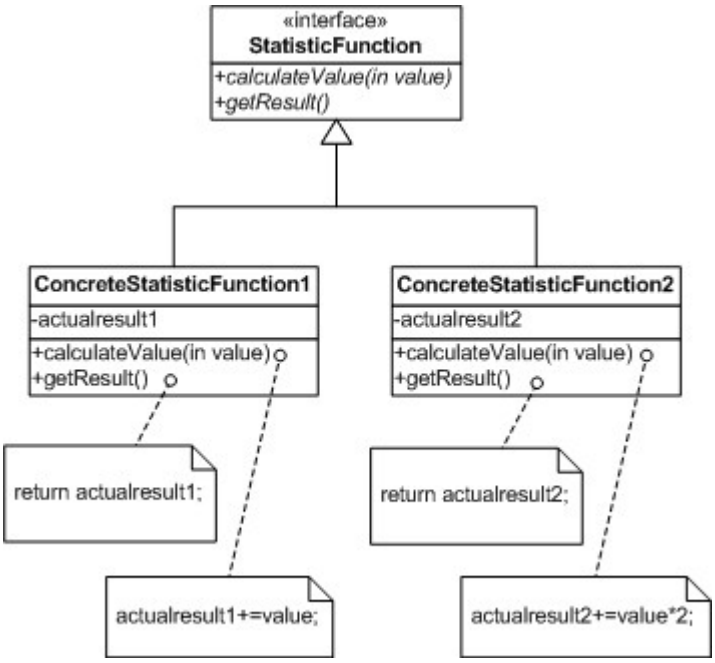

Figure 3.12: Statistical function interface diagram

3.5 Software Architecture

Although the software is developed in an iterative way, it is still necessary to define a frame, representing the architecture to keep on track.

Modern WebRowSet component part

The first level component that is accessing the WebRowSet XML file is a simple SAX parser. Because it is not necessary and also with a view to performance aspects, it is not validating the XML document. The characters are read byte-per-byte and each character is classified for its meaning. This is important, because it has to be taken into account that the WebRowSet file does not mandatory have line breaks to separate the blocks (if so, it would have been possible to read the whole line into a buffer). Normally basic SAX parsers are only used for forward parsing and not for parsing backwards. Therefore the positions of rows, which were found by the parser, have to be stored as parser snapshots (using an implementation of the memento pattern) into a table. If the parser has to be moved to a relative previous row of the WebRowSet file compared to the actual position, its internal state is restored again.

When the SAX parser is instantiated, a so called "Documenthandler" is given to it. This Documenthandler processes the upcoming XML events, generated by the SAX parser. Therefore, the following interface describes the provided basic methods:

Listing 3.1: Interface of the Documenthandler

```
1 public interface DocHandler {
2   public void startElement(String tag, HashMap<String, String> attributes) throws
        Exception;
3   public void endElement(String tag) throws Exception;
4   public void startDocument() throws Exception;
5   public void endDocument() throws Exception;
6   public void text(String str) throws Exception;
7 }
```

Whenever some of these events is caused, the SAX parser calls the appropriate implementation of the DocHandler interface. This concept finally results in the needed WebRowSet module architecture shown in Figure 3.13. The UML diagram in Figure 3.13 consists of the novel implementation component of the WebRowSet interface, the Random Access Parser (RAX), the SAX Parser, and the Memento component. The implementation component delegates low-level operations (e.g. move, delete, etc.) for accessing the WebRowSet to the RAX component. The Random Access Parser component extends the SAX parser and stores the state of the SAX parser (the parsing positions) in multiple Memento objects. The swapfile is used for storing modified data (e.g. new rows, deleted rows, etc.).

Advanced statistical metadata activity

The calculation of statistical metadata was introduced in the paper of Wöhrer, et al. [59]. This earlier version was based on OGSA-DAI 2.2 and was not compatible with the latest 3.0 version. Another thing was that the statistics were calculated every time and with limited scalability

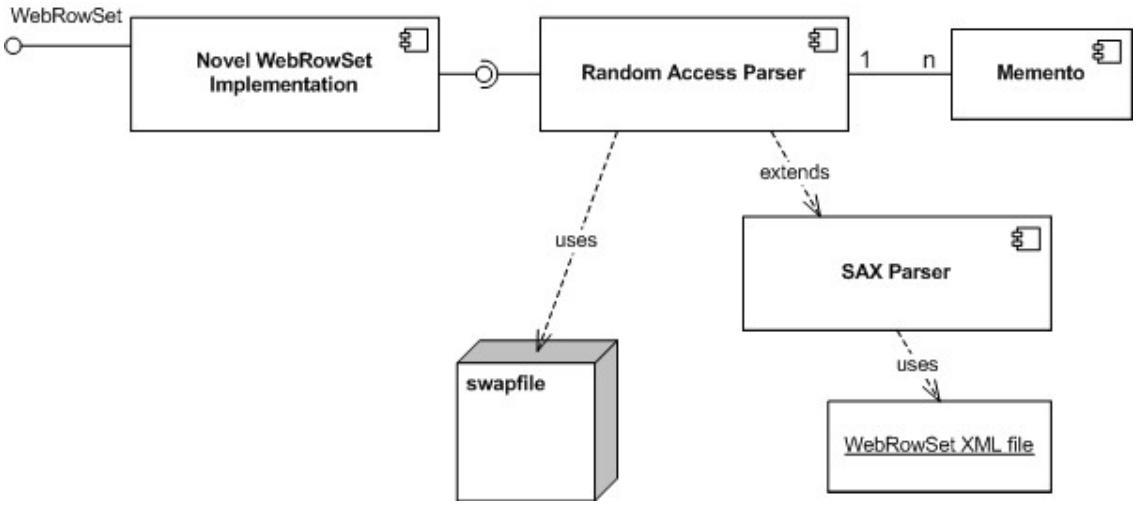

Figure 3.13: Architecture of the novel WebRowSet component

concerning adding additional functions. That's the reason why it was needed to reimplement it. The architecture is presented in Figure 3.14. The client application communicates with the OGSA-DAI framework via the SOAP interface. The OGSA-DAI framework itself is connected to a database via JDBC. The JDBC interface provides connectivity to several databases from different vendors (e.g. MySQL, Oracle, etc.). The metastatistics activity component provides a Tuple input interface for data values, an additional Input interface for configuration data in PMML format, and an XML output interface for the calculated statistics. The metastatistics activity consists of the "MetaStatisticsNGActivity" component which is derived from the MatchedIterativeActivity component to inherit the needed features. The Processor component contains the main routine for handling the Tuple input data and delegating them to the defined statistic functions (standard functions and aggregator functions) through a common interface.

The novel implemented statistic activity is called "MetaStatisticsNGActivity" and allows a very fine-grained way of deciding what statistical calculation should be done on the server-side on which columns of the data.
There are two types of functions that can be used:

- **Standard functions** can be applied and calculated in one dimension on a single column (e.g. maximum value).
- **Aggregation functions** are using the results and values from more than one column (e.g. covariance). Aggregation functions are calculated after the standard functions because of the need of the results.

Both types are represented as Enum types. This is chosen, because it allows to specify a set of Enum objects and assign this to a specific column. On the client side, the application defines the wanted column statistics by adding Enum type function representations to a column, identified by the column name. After that this so called parameter object is transferred to a Predictive Model

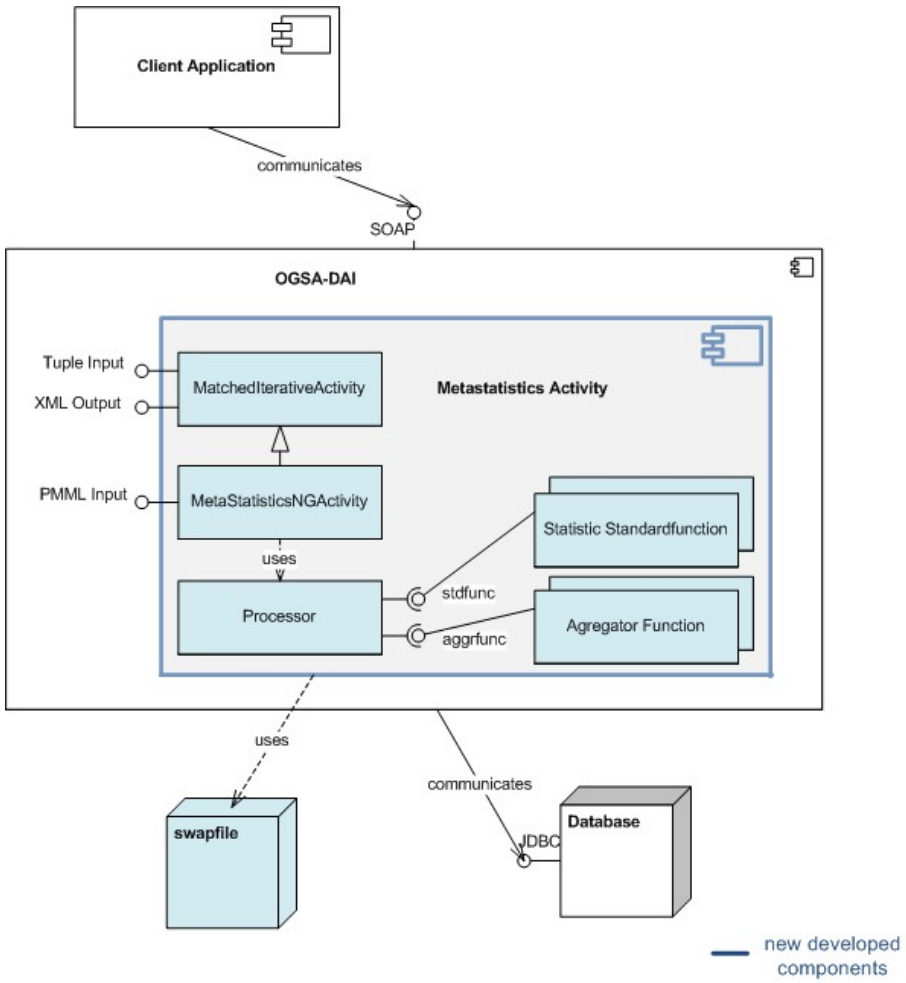

Figure 3.14: Architecture of the metastatistics activity

Markup Language (PMML) object representation, where the standard functions for each column are defined in the "Taxonomy" attribute in the DataField element. The aggregation functions are defined as PMML Extensions because they make use of several columns. To avoid the need of an additional field for the SQL query hash, the attribute "description" in the header section is used. PMML is explained in detail in the supporting technologies section of the next chapter.

Format of the metastatistics XML file

The metastatistics activity outputs its results in a self defined XML format. The root element contains an attribute called "DataSourceID" which is the MD5 hash of the used SQL query that was used to create the corresponding WebRowSet XML file. Therefore this field is important to match the metastatistics and the data. If the query is not passed to the metastatistics activity, the MD5 hash code of the string "default" is used.

As the metastatistics activity itself, it consists of two main elements that represent the two different types of statistics:

- **stdfunctions**
  this element groups the columns, that were selected for statistical calculation and their corresponding set of *standard functions*

- **aggrfunctions**
  this element lists the *aggregator functions* (e.g. covariance) and defines the columns that were used for calculation.

The syntax of the stdfunctions element

As introduced, the columns are listed as elements, equipped with a "name" attribute that specifies the column name. The subelements are function elements, identified by another "name" attribute to see the type of function. The result itself is the function element value.

The syntax of the aggrfunctions element

Because aggregator functions have to be identified too for a later usage (e.g. preprocessing client applications), the element "aggrkey" is used as a subelement to group the aggregator functions that operate on the same columns. The "name" attribute of these "aggrkey"-tags defines the column names, separated by a slash character "/". The functions itself have a syntax like the functions in the standard function section.

Chapter 4

Software Realization

This chapter explains how the software components were realized. It contains a presentation of different used technologies and tools that supported the development process. It closes with UML class diagrams, which give a visual representation of the internal structure.

4.1 Supporting Technologies

The new software is integrated in a runtime environment with several interfaces. The common used interfaces are combined with their supporting technologies and will be introduced.

4.1.1 SOA based software approaches

SOA stands for Service Oriented Architecture and follows the approach of combining low level operations (e.g. database access) to services (e.g. Grid management). The term "service oriented architecture" was used by the Gartner company [40] for the first time. Although there is no general definition of SOA available, often the one of Organization for the Advancement of Structured Information Standards (OASIS) is used:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations [29].

The services represent complete processes inside an organization. Therefore SOA requires a strong inter-working of the IT components for providing their orchestration. For accessing these services outside a SOA based service, various access protocols are available to the clients (e.g. CORBA, SOAP, etc.). Today, one interesting feature of services is the possibility to use them over the Internet. In this case, these services are also known as "Web services". The high-level protocols used by Web services are mainly based on HTTP. It is easier to transfer data with this protocols across NAT networks, but the higher message overhead and additional costs for the processing time have to be considered. The most popular protocols for Web services are SOAP [24] and REST [14].

If the destination of a service is unknown to the client, a discovery service of the service provider can be contacted to get the target information. This service is often realized as yellow pages, where the provided services are grouped by their features.

There are different states on client and server side which form a life cycle; Figure 4.1 shows this in a graphical way:

1. **publish or register** is done by the service provider. It makes the service available and creates a reference in a table.
2. **find** can be used by the client to look-up the service in a directory.
3. **bind** connects the client to the address of the service to forward function calls to the service.
4. **execute** means the call of a service, where function parameters are transferred to the service and the results back to the client.

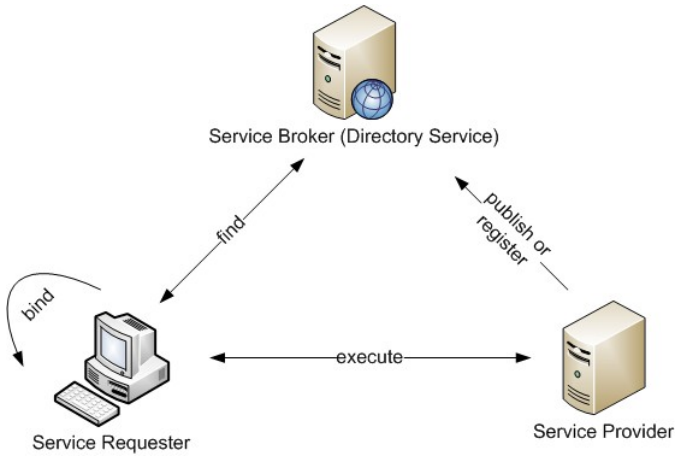

Figure 4.1: Workflow of a Web service based on SOA [52]

4.1.2 Tomcat application server

Apache Tomcat [17] provides an execution environment for Java based Web services. It integrates a Servlet engine, called Catalina and a HTTP server. The HTTP server is often used for development, while in productive environments, the Tomcat connector is taken to bind the Apache HTTP and Apache Tomcat together. Another component is the Jasper compiler, which is used to compile JSP pages into Servlets. Coyote is a HTTP connector which is used to delegate incoming HTTP requests to the application container. The integration of Tomcat's components can be seen in Figure 4.2. The latest current available version of Apache Tomcat (6.0.18) supports the Servlet specification 2.5 [46] and JSP specification 2.1 [45].

Apache Tomcat is a popular open source alternative for executing Java Web service applications. These applications are then used inside Tomcat by enhancing its container with additional framework components to support high-level SOA protocols.

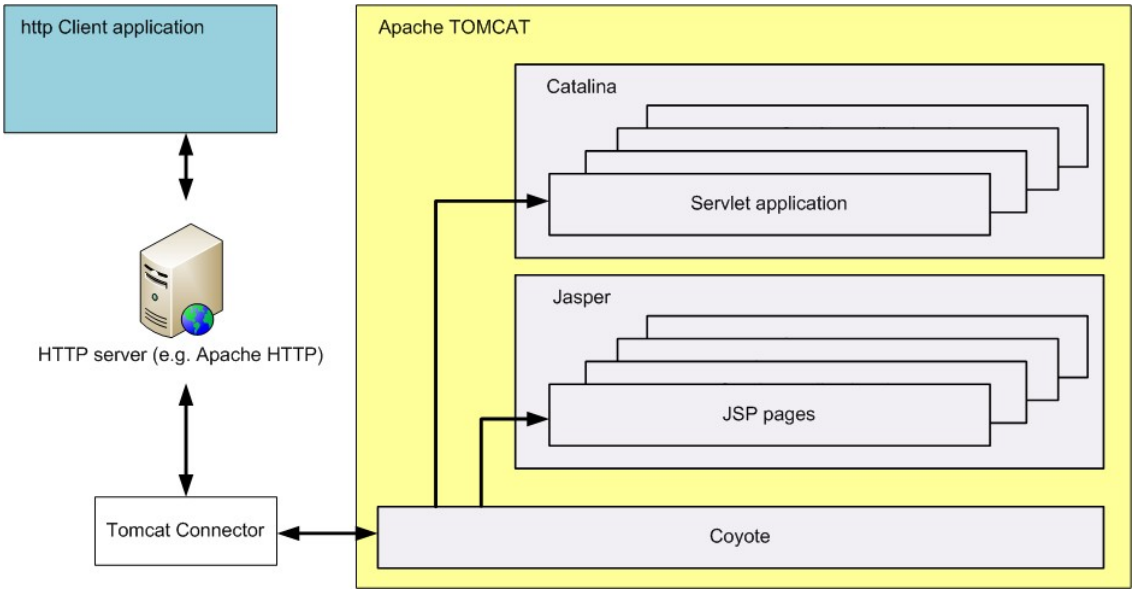

Figure 4.2: Internal architecture of Apache Tomcat

4.1.3 A short introduction to XML

Most of todays exchange formats are defined by XML (extensible markup language). Because of using user defined markers (tags), the data in the document can be structured. This makes it ideal for the use as an exchange format between different systems across the Internet. By using syntax and semantic descriptions like XML schema and DTD (Document Type Definition), the document can be checked concerning its correctness of the specified format restrictions. An XML document can be classified into two levels, which are relevant for XML parsers:

- **Well-formed** - The document passes all XML syntax rules (e.g. every start tag has an end tag). This accordance to the XML standard guarantees that an XML parser is able to parse it. Well formedness can also be seen as the basic format definition.
- **Valid** - If a document is valid, it is also correct concerning the semantic restrictions. These rules are defined by DTD or XML schema. XML schema allows more fine grained rule definition than DTD (e.g. data value of an XML element is of integer type). If a document is valid, then it is also well-formed, but not vice versa. While the validation of an XML document is a very useful option of format checks, it is also a processing time consuming task that is often configurable in different XML parsers and turned-off when not needed.

4.1.4 Different XML parsing APIs

DOM - Document Object Model
The whole document structure is constructed in-core for providing an easy to use way of direct navigation to XML elements.

Advantages: Random access navigation within the XML document data.
Disadvantages: Memory consumption of huge XML documents.

SAX - Simple API for XML
Sequential parsing, event driven.

Advantages: Very small memory consumption.
Disadvantages: No backward navigation, document has to be read again.

STAX - Streaming API for XML
Pulling API for XML documents. The application places a read cursor in the XML document to control the XML parser.

Advantages: Random access navigation within the XML document data.
Disadvantages: Early implementations still have problems with memory consumption of huge XML documents. It was not possible to deactivate the XML validation in the Sun Java implementation of STAX.

JAXB - Java Architecture for XML Binding [35]
JAXB is a framework for Java applications that allows the mapping of XML file data to Java objects. Therefore it provides functions for marshalling and unmarshalling Java objects to XML files without caring about the XML structure and parsing algorithms.

Advantages: Random access navigation within the XML document data, easy to use XML files as Java objects.
Disadvantages: Memory consumption of huge XML documents, because the complete representation is held in memory.

IBP - Index-Based XML Parser model [61]
While searching in the field of available XML parsing technology, I found a very interesting Chinese paper about a parser that was based on indexing positions inside the XML file.

Advantages: Random access navigation within the XML document data with less memory consumption.
Disadvantages: No public available source code.

4.1.5 Description of the WebRowSet format

WebRowset is the defacto standard exchange format between data mining software (Grid services and other database management services).

WebRowSet XML Format

A WebRowSet XML [48] file consists of three sections:

- **properties**
  It contains information about the data source itself - the synchronization provider.
- **metadata**
  In this area, an important description about the data fields is provided, e.g. column name, value type, etc.
- **data**
  Here the data itself is placed, including information about the made changes (inserted, deleted, etc.).

The properties section is a collection of knowledge about the data source. It also provides information, if it is read-only or also updateable (by the concurrency field) and much more other details.

The metadata section is used for resolving the data types to ensure a correct reconstruction of the data into usable Java types. For the standard types, a mapping table in the JDBC documentation is defined - see table A.1. At first, the total amount of columns is defined, followed by the detail descriptions of each column, which are grouped by the "column-definition" tag.

The data section can be seen as the part of the WebRowSet file that contains the payload. Each cell value is stored inside the "columnValue" tag. The columns themselves are then grouped by the "currentRow" element. Usually the WebRowSet file is used as a persistent representation of a CachedRowset. That means, also the changes of rows like inserting and deleting are recorded by using "deleteRow" for removed, "insertRow" for newly added and "modifyRow" for changed rows instead of the name "currentRow".

WebRowSet and Java

The support in Java for the WebRowSet format was introduced with Java version 1.5. The Java WebRowSet is mainly just an interface called WebRowSet, which is derived from the CacheRowSet. Since Java 1.5, Sun provides an example implementation - WebRowSetImpl. WebRowSetImpl implements the whole functionality provided by the interface.

The major aspect is that WebRowSet is derived from CachedRow set. This implies that data is manipulated without having an online connection to the backend database. After the operations were made (e.g. insert, delete, etc.) the off-line RowSet object is synchronized again with the database. Figure 4.3 demonstrates this connectivity between Java ResultSets and RowSets.

Another possibility is to dump the WebRowSet object into a WebRowSet format compatible XML file. This option is often used in the context of data mining applications.

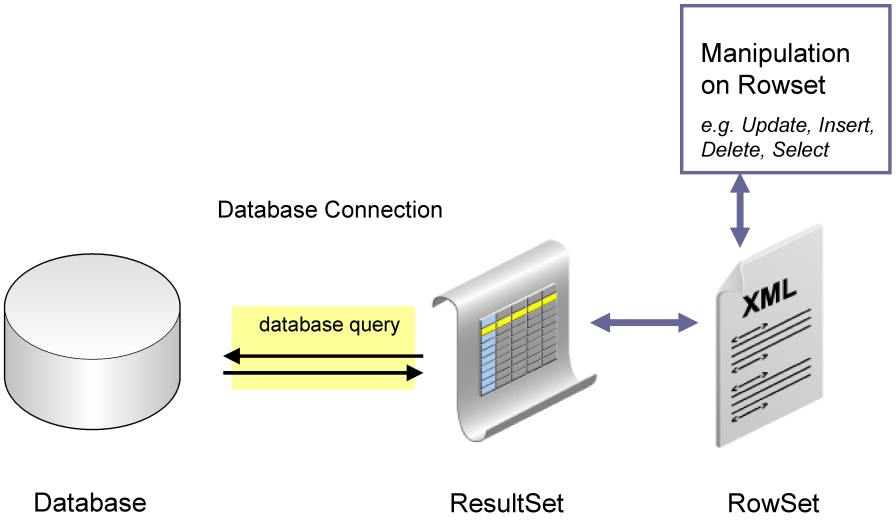

Figure 4.3: Usage of Java RowSet objects

4.1.6 PMML - Predictive Model Markup Language

PMML is a language based on XML and allows the specification of statistical and data mining models. If an application is compliant to the PMML standard, it ensures that it can share the model with other PMML compatible applications. The current released specification version is 3.2 [20].

The format itself consists of the following main parts:

- **Header**
  The Header defines general information about the PMML document like the application description that is responsible for the creation of this PMML document, version, etc.
- **Data Dictionary**
  The Data Dictionary defines the possible model fields by the used data types and value ranges. In this diploma thesis, it is also used for specifying the used statistic functions.
- **Model**
  This component contains information about the used model and the applied algorithms.
- **Mining Schema**
  This schema lists all fields, that are used in the model. So it can also be only a subset of the fields that were defined in the data dictionary. The following information on fields can be used for their description: attribute name, attribute usage Type, outlier treatment method, importance, low value, high value, missing value treatment method, and invalid value treatment method.
- **Data Transformation**
  Because lots of mining models require a specific input format of the values, the transformation component can be used to specify the following PMML types of transformation: DerivedFields, Constant, Normalization, and NormContinuous.

4.2 Development Environment

For development, I used Eclipse 3.2 as an Integrated Development Environment (IDE) and the Java Developer Kit (JDK) version 1.5.0:09 to ensure full Java 1.5 compatibility. For a better navigation over time, I took Subversion as a Code Management System (CMS) in version 1.4.6. Eclipse itself has been used with Subclipse for a useful connection to the Subversion server, which was running locally. The difficulty of creating a development setup is to use versions of the tools, which are compatible. To avoid this time consuming task, I used an all-in-one Eclipse distribution called "EasyEclipse Server Java" [25] version 1.2.2.2. This distribution contains all needed development components (e.g. XML editor, Subclipse, etc.). Optional components can be added afterwards in a tested version concerning their interoperability.

The source code has been developed in an iterative way by a test-first test-driven software development philosophy. This was very useful when working on the WebRowSet implementation and ensured that the already working tests were still working although some new features were implemented.

The tests against the WebRowSet interface were done with the Junittest 4.0 framework and Easymock 2.3 to mock up the different implementation level classes.

4.3 Selected Preprocessing Algorithms

To demonstrate the functionality and how the work of a preprocessing client could look like, two preprocessing methods were chosen using different algorithms for data preparation.

- Out-of-intervals method using Mean Absolute Deviation (MAD) as a robust estimation for the deviation.
- Dimensional reduction method using PCA - principle component analysis

The mathematical background of these algorithms has already been introduced and can be found in the introduction of statistical basics chapter.

4.3.1 Out-of-Intervals

For detecting outliers, it is important to have a robust estimation concerning the expectation value. The first possibility is to use the arithmetic mean. The average value is not a good estimation, because it is very sensible when adding values which are in a great distance from center.

Robust estimation
That's why it is better to choose the MAD as a robust alternative for the average. The calculation of the MAD requires the possibility of calculation the median of a dataset. The calculation of median, when having huge datasets needs some extra effort, because the single data values have to be sorted in an ascending order. This sorting has to be done out of core, because the amount of possible data values is not defined and can exceed the available main memory.

$$MAD = median \left| \sum_{i=0}^{n} (x_i - x_{med}) \right|$$

The method calculation of the quartiles Q1 and Q3 vary, but it is defined that the lower quartile is the border, where 25% of the values are beneath it. The upper quartile covers 75% of the total amount of values. The algorithm for calculation Q1 and Q3, that was also used in the MedianSorter class, is defined the following way:

$$\begin{aligned} grade_{Q1} &:= 0,25 \cdot (n-1) \\ grade_{Q3} &:= 0,75 \cdot (n-1) \\ index &:= \text{integervalue}(grade) \\ weight &:= grade - index \end{aligned}$$

$$\begin{aligned} Q1 &= x_{index} + weight \cdot (x_{index+1} - x_{index}) \\ Q3 &= x_{index} + weight \cdot (x_{index+1} - x_{index}) \end{aligned}$$

4.3.2 Dimensional reduction

Data mining applications are usually confronted with high dimensional data. This is also caused by the fact that the joining of tables (in the integration phase of the KDD) create additional dimensions. Due to the high optimization potential of dimensional reduction, I choose PCA to show the practical usage of the aggregator functions of the statistics activity.

PCA - Principle Component Analysis
For matrix calculation, a Java math toolkit library was used - called JAMA [26], created at the National Institute of Standards and Technology (NIST). It supports many linear algebra functions like:

- Eigensystem solving
- LU decomposition
- Singular value decomposition
- QR decomposition
- Cholesky decomposition

For the PCA on the client side the covariance aggregator function of the MetaStatisticsNG activity is used to defined the wanted values calculated for the Covariance Matrix.

4.4 Implementation and Class Diagram

The two treated main requirements are system memory consumption in terms of WebRowSet implementation and performance saving in terms of the metadata statistics activity. The secondary aspects for focusing on scalability were targeted by using a modern design of software engineering patterns.

As it was seen from early performance measurements, which are represented in the file system benchmarks, it is very inefficient to read byte per byte from the files. The performance is much higher when reading bigger pieces to a buffer and then reading out of the buffer. If the buffer is empty, it is refilled by another reading burst of data. So I encapsulated the RandomAccessFile class functionality into a RandomAccessFile class by deriving the RandomAccessFile class and overriding the effected read methods, which were later used in the developed software modules:

- int read()
- int read(byte[])
- long getFilePointer()
- void seek(long fileposition)

The buffer size is 64 kBytes, which also equals the size of the reading burst. With this method, a WebRowSet file with 10000 data rows can be read in 78 ms instead of 11 s when using the unbuffered version.

4.4.1 Modern WebRowSet component part

The implementation from Sun, named "WebRowSetImpl", works well for the most database applications. It has some major limitations on the size of data. The WebRowSetImpl class is implemented by using some kind of in memory DOM model (this is assumed, due to a big memory use when the file is loaded). For operating on huge datasets, as they are used in data mining, an out-of core implementation is necessary to avoid memory limitations and enable scalability. Figure 4.4 pictures how an existing application can benefit from the novel implementation by using the same WebRowSet interface again.

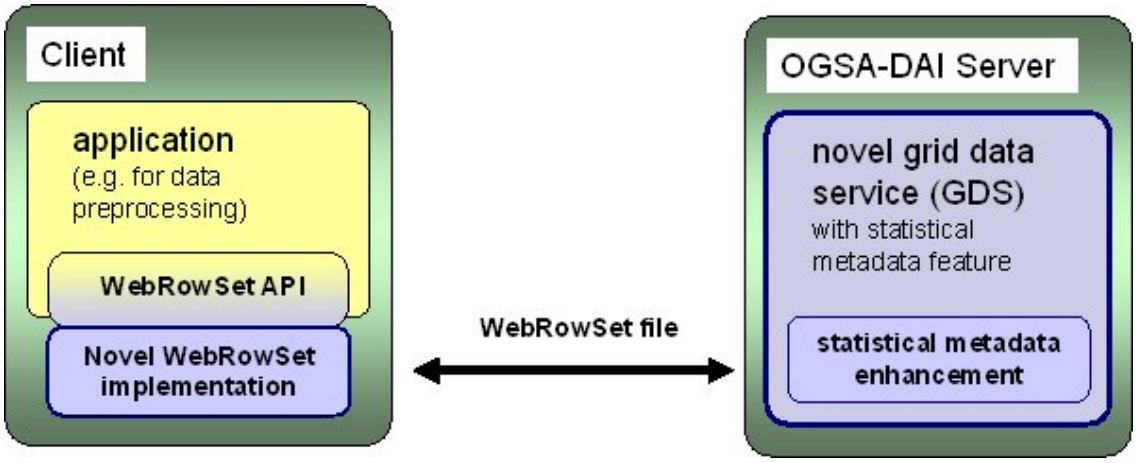

Figure 4.4: Novel WebRowSet API implementation

As introduced, my WebRowSet implementation consists of a trivial SAX parser implementation, that was enhanced with the functionality to store and restore its internal state by using the memento pattern. The state is stored by an object that is instantiated by a Java inner class (a class defined inside a class). The so called created RAX parser (RAX stands for Random Access XML parser) is navigated by forward navigation through the rows. During the movement of the cursor to the target row, SAX events are thrown and treated by the document handler. The document handler then sets or gets the internal state of the SAX parser.

The approach of a simple SAX parser - called quick and dirty parser - was posted by Steven R. Brandt on Javaworld.com and can be found in [6]. I modified his idea of a basic non validating, event driven parser in the low level basis.

This underlying core only supports reading from an already existing WebRowSet XML file in a random access way. To support writing, the RAX parser is surrounded by a delegation pattern that buffers modifying operations (e.g. "delete", "insert", etc.). This buffer is realized by a change table, that overlays the WebRowSet file data.

To enable the insertion of a huge amount of new rows, this new data is swapped to the filesystem by serializing these objects to a RandomAccessFile. The class SoftTreeMap provides the interface for doing this. This class realizes a write-through cache by caching the new row data with Java

SoftReferences and sending them in parallel to the ObjectSwapStorage class, which finally writes them to the file system. I chose this very strict caching strategy of a write through cache because it is easy to implement and reliable.

For change operations on the WebRowSet object, a state machine implementation following the "State Pattern" [18] is used. Figure 4.5 illustrates the class diagram in UML notation. The architecture concept of Figure 3.13 was used for the implementation. The delegation component of the WebRowSet interface is called "NovelWebRowSetImpl". Beside of state handling on WebRowSet change operations, the function calls are forwarded to the "RaxNavigator" class. Inside this class, the core logic of row operations are implemented, which control the "RaxParser" for accessing the data inside the WebRowSet file. The "CacheDecorator" class decorates row modification functions by using the "ObjectSwapStorage" class (this class stores new added rows to the file system) and the "SoftTreeMap" (this class realizes the write-through cache). The low level parsing operations including the memento object creation is done inside the "RaxParserImpl" class.

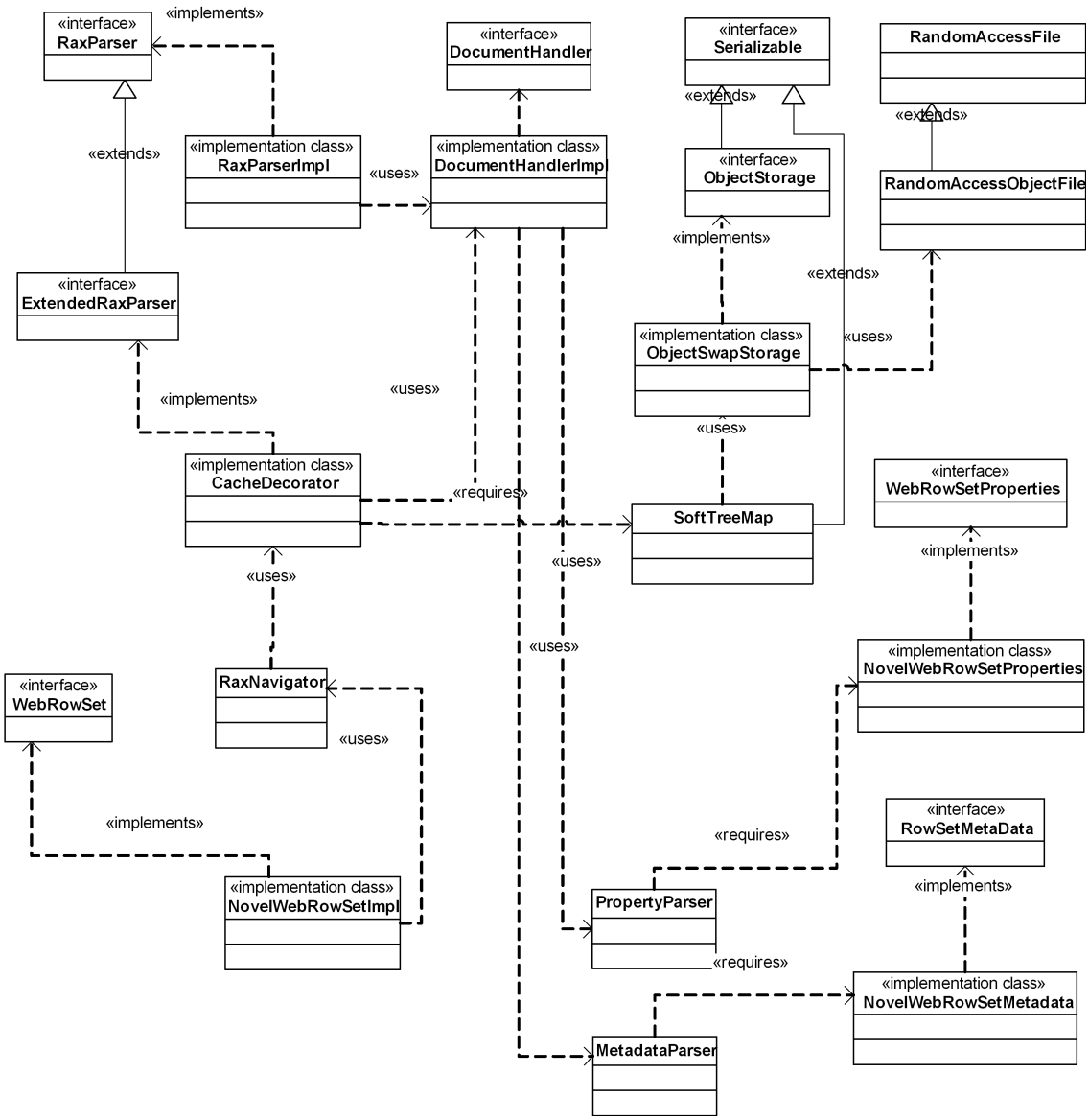

Figure 4.5: UML class diagram of novel WebRowSet implementation

4.4.2 Advanced statistical metadata activity

One of the constituted key technologies is the Apache Tomcat application server as an SOA environment for the novel statistical metadata subsystem. Figure 4.6 demonstrates the integration of the OGSA-DAI framework into Tomcat. In this diploma thesis only the Servlet engine itself is used to run the Apache Axis framework and the OGSA-DAI service framework. The used Tomcat server version is 5.0.30. It is compatible with Servlet specification 2.4. I tried running higher versions of Tomcat and Java runtime and got bad side effects. OGSA-DAI 3.0 only supports version 5.0 of Tomcat (with some hints to version 5.5). It was not possible to use Java 6. So the setup was fixed to Tomcat 5 and Java 1.5.

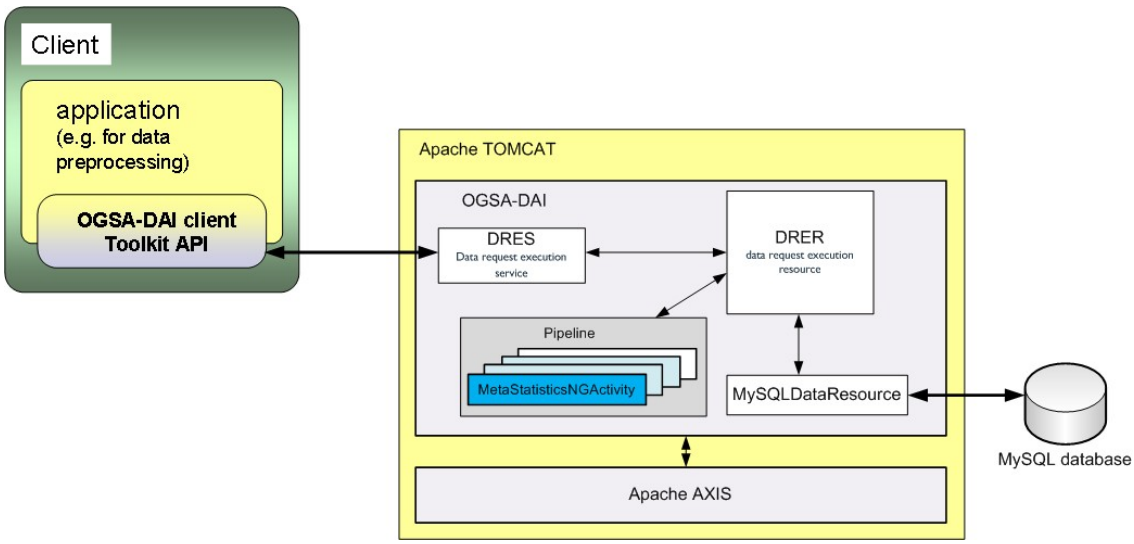

Figure 4.6: Architecture of Apache Tomcat and OGSA-DAI framework

The advanced statistical metadata activity is split into a client side and a server side. Figure 4.7 illustrates the architecture of both sides.

Client side

The client stub has the main objectives to passtrough the parameter data to the server side and providing connection methods for input and output. In the context of the MetaStatisticsNG activity this means to transform the parameter objects for the specification of the column statistics to a PMML (version 3.1 was used) compatible string. This marshalled string is then sent to the server side.

Because PMML in the OGSA-DAI activity is used as an exchange format between client and server part, only the attributes of the DataField element and the Extension elements were used to store the function information.

The parameter object is easy to use by adding a set of functions to the internal column maps. These functions are defined per column by using a set of ENUMs. This set is added to the parameter object using the “addstdfunction” for standard functions and “addaggrfunction” for aggregator functions. Another important function is the “setSQLID” function that allows to define an ID for the later generated PMML object for identifying the generated metastatistics XML file.

Server side

The MetaStatisticsNG activity is derived from the MatchedIterativeActivity to use basic methods of an OGSA-DAI activity. It consists of the main class “StatisticProcessor”, which also has the

central process routine, where the Tuples are decomposed and sent to the matching statistic calculation functions.

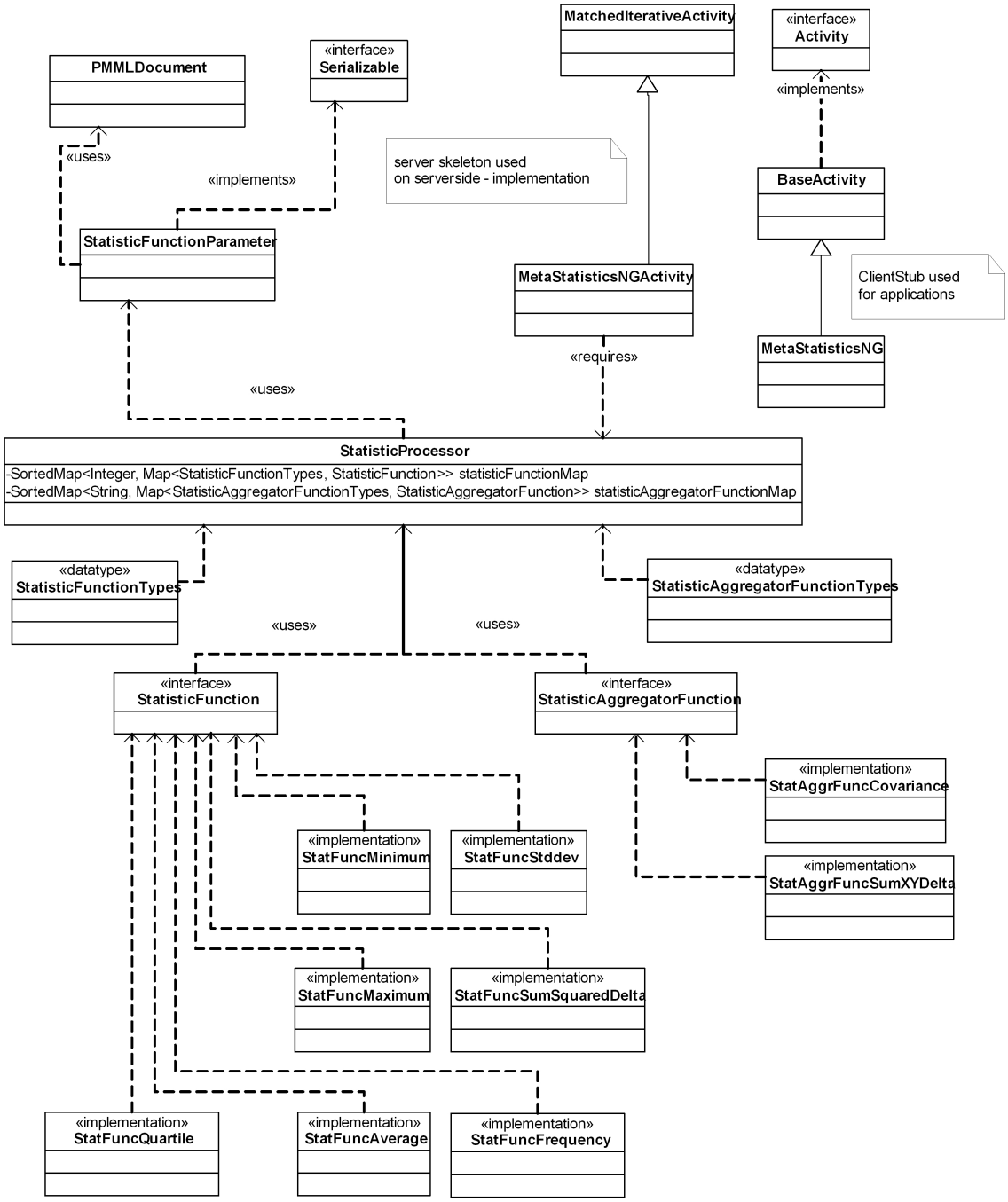

Figure 4.7: UML class diagram of the advanced metadata statistics activity implementation

The two types of statistic functions, standard and aggregator functions, are implemented as classes, which implement a common interface called “StatisticFunction” (for standard functions) or “StatisticAggregatorFunctions” (for aggregator functions). This design is chosen for providing one

standardized interface for all different statistical function objects. When the "process" method iterates through the function objects, it invokes the interface function next() to pass the actual data value. The concrete object instance is resolved by the Java internal OOP polymorphism feature during runtime. With this architecture, it would be possible to instantiate the statistic function objects on the client side and send them to the server, which would improve the possibilities to enhance the MetaStatisticsNG activity with new functions without recompiling and deploying it to the container. At the moment, OGSA-DAI does not support any mechanism (e.g. Java RMI) for marshalling and unmarshalling objects between client and server side. The difficulty is the dynamic class loading. It has to be implemented separately and the serialization format of class files depends on the underlying virtual machine and version. That's the reason why the dynamic class loading feature of statistical functions is not used in this thesis.

The dependency of the function objects is resolved automatically, by a dependency table. If a function object is created, function objects will be instantiated by taking into account the dependent classes.

The "StatFuncFrequency" class is equipped with a pass-through functionality, for counting the data values in three characteristics - total frequency, distinct values and null values count before passing the data to the other objects.

After the statistical standard functions were applied on the columndata, the aggregator classes are used to collect the needed results from the column results.

Out-of-Core implementation of median and quartile algorithm

The difficulty of calculating the median is, that the data values have to be sorted in ascending order. This means that the values are normally held in memory. Referring to the major design criteria, to avoid memory boundaries, a method was implemented without having this limitations by operating out-of-core. The first version of the sorting algorithm was based on a linear search, and then doing bytewise copying. This resulted in a very bad runtime performance for big data value lists.

Then I used the binary-search algorithm that operates based on the "divide and conquer" principle, which has one of the best runtime behaviors $O(\log n)$. After the position has been found, I am using a buffer for copying the following data backward inside the file. The copying of bigger blocks instead of 4 bytes improves the moving performance about factor 6. At the end the new value is sorted-in at the position that has been identified by the binary search. With this method I keep the values sorted in a rewritable random access file. This class is enhanced later to provide the calculation of the quartiles (Q1,Q3) and the median (=Q2) on demand.

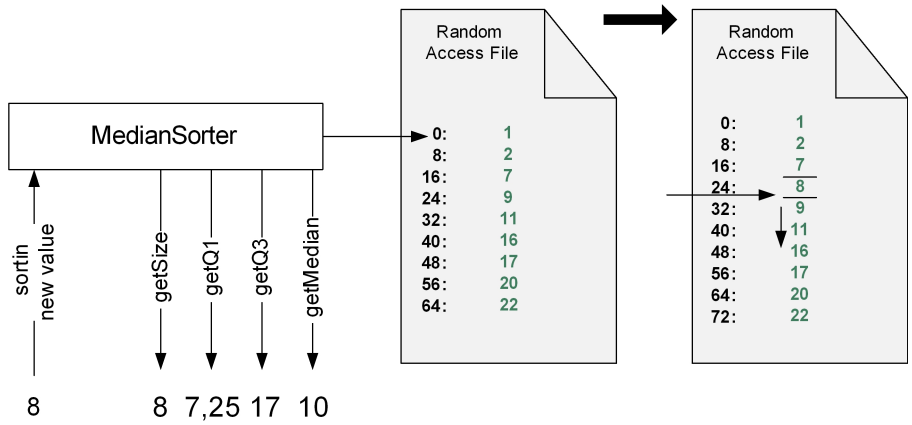

Figure 4.8: MedianSorter quartile calculation process

As it can be seen in Figure 4.8, the MedianSorter component provides an input method for sorting in a new value to the data set, on which the median is calculated. This input method executes the described sorting algorithm. After the algorithm is finished, the output methods can be called. The data set size, lower quartile, upper quartile, and median are calculated on demand by calling the appropriate output method.

Chapter 5

Validation of Software Components

This chapter describes the use of a suitable test environment including test cases. The test details are valuable to adapt the test to other environments. The interpretation of the results is focused on the topic of this work.

5.1 Methods used for Validation

The most important measurement of the novel WebRowSet implementation is the memory consumption. Due to the in-core limitation of Sun (exceeding memory), only the default memory amount of the JVM could be used for operations. On Microsoft Windows XP workstations the default value for the assigned JVM memory is around 64 MBytes. This value can be increased by using XMem flags when calling the JVM. As described earlier, Eclipse was used as the development environment. In order to see if the implemented components work correctly, the actual amount of free memory needs to be controlled. By default, Eclipse assigns about 60 MBytes to the JVM. This value was artificially decreased by reserving a specific amount of empty bytes (e.g. the Java construct *new byte[1024*1024]* reserves 1 MByte Memory). This is important for a fine-grained debugging session.

The integration test was done with the demonstration client using advanced data preprocessing methods for detecting values out-of-intervals and the PCA method for dimensional reduction.

5.1.1 Reusability of the developed software components

The software is designed using interfaces for an easier adoption of the implementation and using widely OOP functionality. Because of the fact that the components are written in Java, the source code provides greater portability than native C code - it runs on every Java 1.5 compatible runtime environment. The implemented design patterns (Memento, Decorator, etc.) guarantee an easy understandable source code.

5.1.2 Performance

For the performance tests, various tools where used and are described in detail in the "Testing tools" chapter. Beside of this, the build-in Java function "public static long nanoTime()" of the "System" class was used for calculating the duration needed for the several execution steps in the software. This function returns the time in nanoseconds of the most precise system timer. Sun recommends this method only for measuring elapsed time and not for system time calculations.

5.2 Data Preprocessing Design

The inter working and implementation of the used software engineering patterns (Memento, Decorator, etc.) can be found in the "Design UML Modeling" chapter.

5.3 Test-Cases

The test cases, that are defined are directly derived from the real world applications for data preprocessing and are grouped by the two big software parts of this diploma thesis:

5.3.1 Test-Cases of the WebRowSet implementation

- *measure the time for accessing the first row*
  This shows how long it takes the application for getting ready to navigate through a WebRowSet file.

- *measure memory usage for accessing the first row*
  The memory usage that is consumed by the JVM demonstrates the character of the internal implementation. In-core implementations show a huge memory consumption for generating the internal representation of the whole XML file structure. The out-of-core implementation only needs space for the indexes (Memento objects), which leads in a lower amount of needed space. Due to the fact that this test only accesses the first WebRowSet line, less than 10 Memento objects are created (e.g. starting point of the properties section, metadata section, etc.).

- *measure time for reading (iterating through) the first 1000 rows*
  This test measures the pure reading time with a completely loaded WebRowSet object. This means that the "readXML" function of the WebRowSet object was called and finished successfully, and the object is ready for reading the rows. The in-core implementation is faster on iterating through the rows due to loading the values from a primary storage (main memory). The out-of-core implementation reads from a slower secondary storage resource (hard disk).

- *measure memory usage for reading (iterating through) the first 1000 rows*
  The memory consumption during reading is very different between the in-core and out-of-core

approach. The memory consumption of the out-of-core consumption only depends on the amount of rows and not on their contents. The in-core implementation holds both in memory - the data values itself and the structure of the data values.

- *measure time for reading (iterating through) a file with 250000 rows*
  To see the practical usability of the novel WebRowSet implementation, it is important to measure the time it takes to iterate through a big amount of rows. In theory, the out-of-core implementation is slower, but it is still fast enough for a usable workflow.

- *measure memory usage for reading (iterating through) a file with 250000 rows*
  The reading of a big number of rows shows the possibility of handling big files without modifying the local JVM settings of the application.

- *measure duration for writing an opened WebRowSet object back to the file system as an XML file*
  The novel WebRowSet implementation caches the modified rows in the main memory as long as space is available. Otherwise it starts swapping the cache to the file system. The duration for writing heavily depends on the number of rows which have been changed. This test should rather demonstrate the compatible creating of a WebRowSet file.

5.3.2 Test-Cases of the statistical metadata activity

- *measure the time for processing 1000 rows of a database 10-dimension table*
  This test measures the reference performance for calculating statistics and allows to measure the CPU load on client side and server side of the statistical metadata activity process. To isolate the basic load of the complete OGSA-DAI framework inside the Apache Tomcat server, the test waits 1 minute till the Apache Tomcat server is loaded completely before starting. After this, the test waits again for 20 seconds to prepare the Java monitoring tool - jConsole.

- *measure memory usage for processing 1000 rows of a database 10-dimension table*
  The reference memory consumption is measured over the complete execution phase of the statistical metadata activity by using jConsole for recording the used heap space over time for the client side and server side.

- *measure the time for processing 10000 rows of a database 10-dimension table*
  To demonstrate the expected performance benefit of the statistical metadata activity on bigger data sets, the used data set size is increased by factor 10.

- *measure memory usage for processing 10000 rows of a database 10-dimension table*
  This test is used to evaluate the memory consumption over a longer time period. The needed memory size should be constant to prove the out-of-core approach.

The test cases for the OGSA-DAI module are demonstrating the splitting between the client and the server part concerning memory consumption. In my test configuration, the performance benefit can be derived from the different CPU load values of the OGSA-DAI server part and the client part.

5.4 Testing Tools

For getting a complete image, the tests were done in different ways. The reason for choosing the specific tools was the fact, that they are used and proved by other projects. The advantage of an open source version is the transparency of the implementation. The major stages of the standard software development process where covered (e.g. unit tests during development, integration tests at the end, etc.).

5.4.1 JUnit

JUnit [23] is a Java framework to run automatically unit tests on Java components. These components are mostly classes and methods. The test for a class is written before the implementation of the class. This sequence makes it easy to see the implementation progress (how much test cases work), the behavior of a class and correctness of the interface definition. If the source code of a component got changed, a run through the test cases verifies, that the code is still working as expected. That's why this software development method is also known as test-driven development.

5.4.2 Mock objects - EasyMock

Mock objects are used inside the JUnit test. They act as "dummy" objects without any logic inside. During the run of a test case, it is defined, which and how often methods will be called and what these methods should return. This could be done without any implementation - just by the interface definition and the instructions concerning the function calls. This means that the behavior of a mock object is always defined by the tester. This opportunity is very useful if an underlying layer is not implemented at testing time of the upper layer objects. In this thesis the Java mock framework called "EasyMock" [33] was used.

5.4.3 jConsole

Since Java 1.5, parts of the Java management extension (JMX) where included in the API. Part of the JDK is a program called "jConsole", that is a GUI for monitoring JMX components. If a Java 1.5 JVM is started with the parameter "-Dcom.sun.management.jmxremote" it enables its remote management facilities. Then jConsole can be used to observe the current Java runtime in detail (e.g. memory consumption and garbage collection). The main screen of jConsole could be seen on Figure 5.1 showing a summary report. In the center it displays a detailed report of the current memory consumption. The listed sections can be inspected in detail and observed over time by selecting one of the main slides (e.g. Memory, Threads, etc.).

5.4.4 Self developed software components

The creation of the graphs during the first implementation phase was done by using JFreeChart, which is a Java framework, that is focused on visualizing data by using different chart types

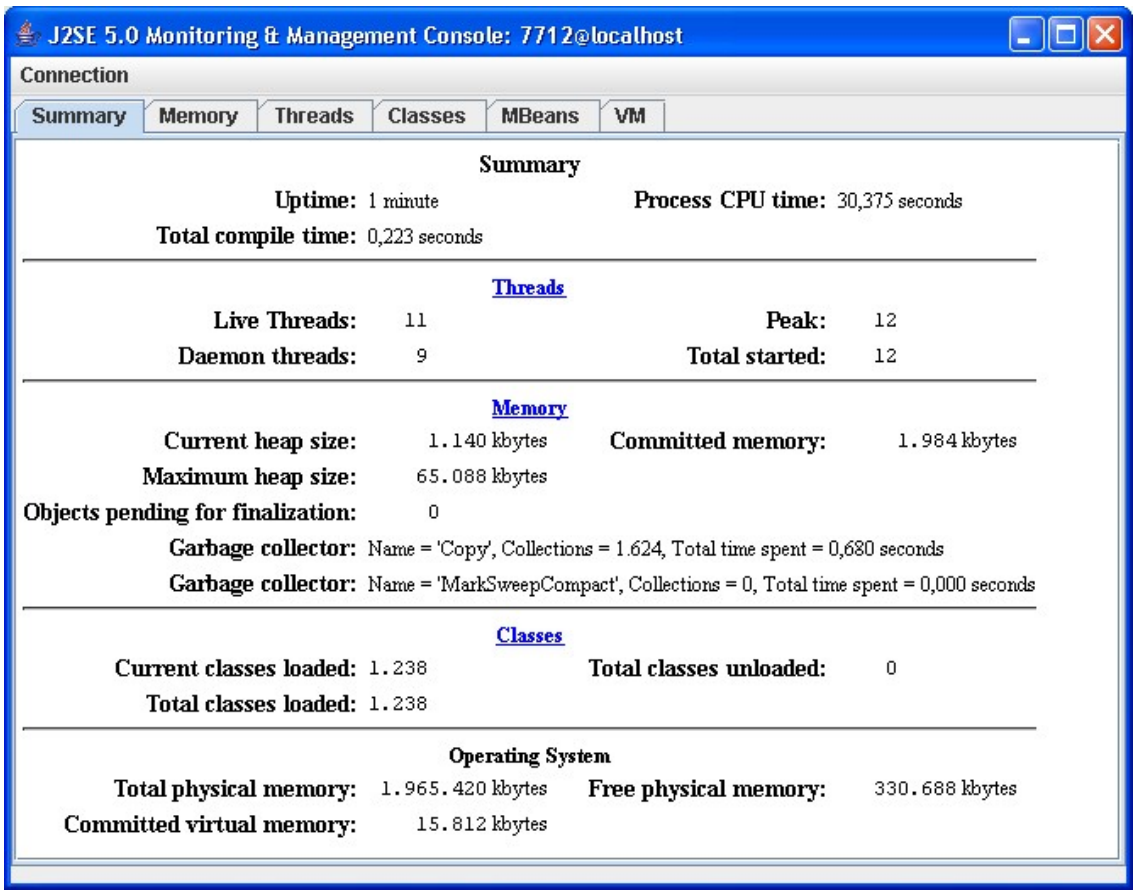

Figure 5.1: Main screen of JConsole Windows XP

(bars, pies, etc.). To allow visualization within a network, I implemented a small server part that calculates the current JVM memory usage statistics and sends this information to the client side via UDP packets. On the client side, this information is collected and a correlating graph is drawn by using this memory statistic time series. The advantage is that the graph construction does not influence the JVM which is executing the Novel WebRowSet implementation. Only the functions for getting the memory usage and sending the UDP packets can be seen as an influence factor of the performance measurements. Other inter connection methods like Soap and Java RMI are too heavy weight because of the marshalling and unmarshalling overhead.

5.5 Test Environment

The two main measurements of the novel WebRowSet implementation where memory consumption and read/write performance. Measuring the memory consumption was done by using functions provided by the Java virtual machine. It is important to take into account that Java virtual machine implementations are different on other operating systems and also across the manufacturers of virtual machines. Although there is a certification process for getting the “Sun Java compatible” logo for a JVM, Sun does not define in detail the behavior of the garbage collector and memory

management. For my test, I used the Sun Java JVM 1.5.09b for the Microsoft Windows XP 32-bit edition.

The hardware consists of a Hewlett Packard notebook model nx6325 equipped with up-to-date hardware. The main thing which has to be taken into account is, that also the OSGA-DAI server part including the SQL and Tomcat server is running on the same host. The use case of this diploma thesis is, that the server applications run on the Grid itself, providing much more performance, by using server CPUs (e.g. Intel Xeon or Sparc) that are optimized for huge data transfers.

Because of the used software architecture, which is mainly based on an out-of-core implementation (e.g. where sorting data values in a file), the read/write performance of the storage has a major impact on the performance results. This impact is caused by using secondary storage type (hard disk) instead of primary storage type (main memory) which expands scalability but decreases performance. The impact depends on two major factors:

- I/O performance of the storage device (transfer rate and access time)
- used file system (Fat32, NTFS, Ext2, JFS, etc.)

The major operations that are normally done in memory (e.g. sorting of values) have to be done on the file system. The tests where done on an NTFS formatted partition on a built-in hard disk of type Hitachi Travelstar 5K100 with a cluster size of 4096 Bytes.

The measuring of these aggregated components was done by using a file system benchmark tool. It measures the performance for reading and writing to and from the file system using different block sizes. The file system benchmark tool used was IOzone [27] version 3.311 (Windows XP binary). The results of the benchmark are presented in Figure 5.2 and 5.3. The description can be found in Section 5.6 presenting the results achieved.

Furthermore, it has to be considered, that the server components (DBMS, Java Runtime) were not tuned for the optimum performance. In a real environment, this requires specialized knowledge of involved administrators and engineers.

5.5.1 Hardware

The hardware consists of a Hewlett Packard notebook nx6325. The relevant hardware specifications can be seen in Table 5.1. It is important to notice that the productive hardware is much more performant than my testing hardware, also used for development. Nevertheless, the benchmark results can be used to compare my test hardware with the productive hardware-based performance.

Table 5.1: Parameters of hardware that was used for testing

Hardware Module	Description
CPU	AMD Turion X2 mobile CPU (dual core, 64 bit support)
Memory	2 GBytes DDR2, 1,87 GBytes available for OS

5.5.2 Software

Table 5.2 shows the software runtime environment including the version information. Most of the software is free for use and some of it is available in open source too, which is very useful for debugging. In the high performance computing sector, the usage of 64-bit software is standard, because of the increased operation performance in handling scientific primitives (e.g. Java double and long primitives are 64 bit values). The most advantage is the facility of accessing more than 4 GBytes of Ram. I did not have a 64 bit version of Windows XP available, so I only used a 32 bit software environment.

Table 5.2: Used software environment including their versions

Software Module	Name	Vendor	Version
Operating System	Windows XP Professional	Microsoft	5.1 SP2 - 32 bit x86
Java Application Server	Apache Tomcat	Apache Foundation	5.0.30
Java Virtual Machine	JDK	Sun Microsystems	1.5.0 15-b04
Database Server	MySQL	Mysql	5.0
Data Integration Framework	OSGI	Open Grid Alliance	3.0
Soap Framework	Axis	Apache Foundation	1.4
FTP Server	Filezilla	Tim Kosse	0.9.27beta

5.5.3 Test data

For the first tests, some WebRowSet files from the GridMiner project were used. Later on I searched for statistical tools that provide the generation of synthetic test data. There are very few tools available and the free tools were not able to create a WebRowSet compatible file. That is the reason, why I adapted the “Agrawal” [2] data generator that is delivered with the WEKA [32] Data Mining Tool Developer version 3.5.8 . WEKA is available in open source so i could adapt the existing generator to the needs of this diploma thesis. The data is generated in parallel for two outputs - WebRowSet XML and database. The reason for this is the chance to compare the behavior of the OGSA-DAI activities with the bare generated WebRowSet file from the generator. This procedural method allows to detect a bug in the generation of WebRowSet XML files inside the TupleToWebRowSetCharArray activity of OGSA-DAI. It produces a format, that cannot be handled by the standard WebRowSet implementation of Sun.

The WebRowSet output is important for off-line tests without any database connection. The MySQL JDBC driver “mysql-connector” was used for writing the generated test data into the test database. This filled test database was used later on for the OGSA-DAI workflow activities as described earlier.

The adapted generator for data was used to create tables with the table layout shown in Table 5.3.

The test data listed in Table 5.4 were used for analyzing the novel software components.

Table 5.3: Features of the test data table layout

Columnindex	Columntype	Value range	Nullable
1	double	20000-150000; continuous	no
2	double	20-61; discrete	yes
3	double	0-75000; continuous	no
4	double	0-5; discrete	no
5	double	1-21; discrete	no
6	double	0-9; discrete	no
7	double	-449991 - 9; continuous	no
8	varchar(20)	20 discrete strings	yes
9	double	0-500000; continuous	no
10	double	0,1; discrete	no

Table 5.4: Table of used test files

number of rows	file size	data source	file name url
1000	435 kBytes	DataGenerator	testdata_1k.xml
100000	41, 7 MBytes	DataGenerator	testdata_100k.xml
23	17, 2 kBytes	GridMiner	23_rows_webrowset.xml
250000	92, 0 MBytes	GridMiner	250k_rows.xml
0	6, 7 kBytes	GridMiner	propertiesTest.xml
23	17, 2 kBytes	GridMiner	metadataTest.xml

5.5.4 IOzone - a file system benchmark tool

IOzone is licensed as Freeware and available in Open Source. It is used as a standard tool for file system benchmarks and also used by other papers [1]. For out-of-core architectures and its great impact on the developed software components, it is important to know the total file system performance of the used workstation to compare the output values. I chose a maximum file size of 200 MBytes, which represents the standard WebRowSet XML file sizes that were used for testing.

Figure 5.2 shows the read performance, while Figure 5.3 shows the write performance. The values itself demonstrate, that the read performance is about 2-8 times higher than the write performance and prove that reading bigger blocks results in a much higher transfer rate. They also illustrate the notebook hard disk performance, which is about half the speed of a workstation device. This is fundamental in future comparisons, when operating in an high performance computing environment (the I/O file system throughput will be much higher).

The test results present the expected poorer performance of the notebook hard disk (although it is connected via a fast SATA interface, combined with non optimized NTFS file system).

5.5.5 SPECjvm 2008 - an additional third party benchmark tool

The Standard Performance Evaluation Corporation (SPEC) [10] provides benchmarking tests for several application scenarios. The SPECjvm 2008 from SPEC was used to get a reference base to compare the Java performance of the used setup (hardware and operating system configuration),

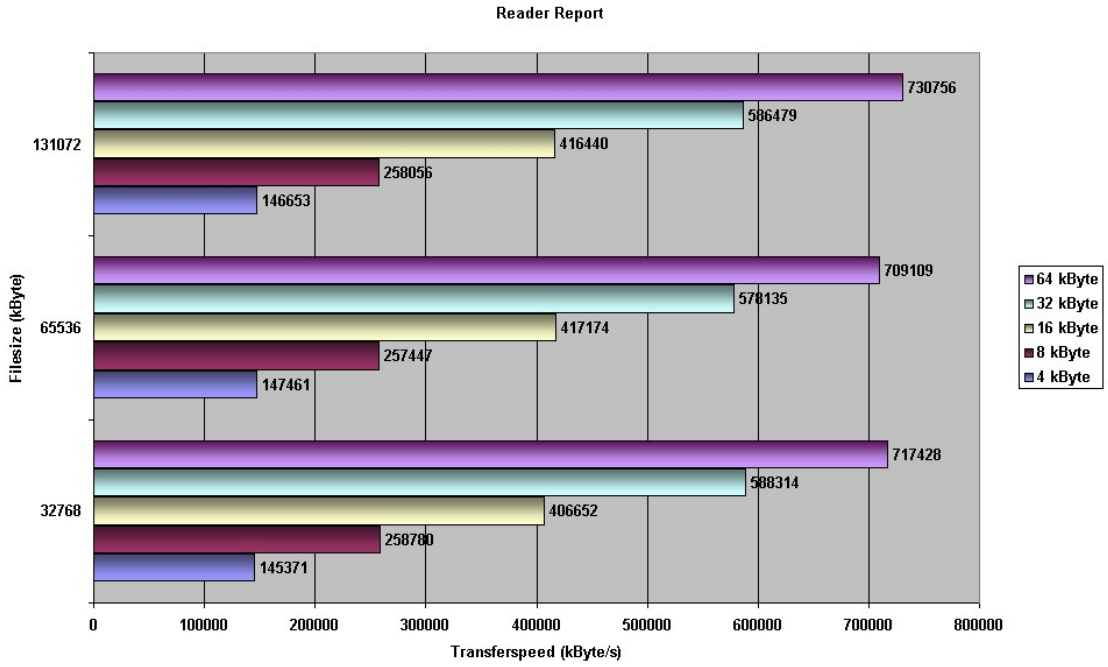

Figure 5.2: Reader performance, measured with the IOzone benchmark

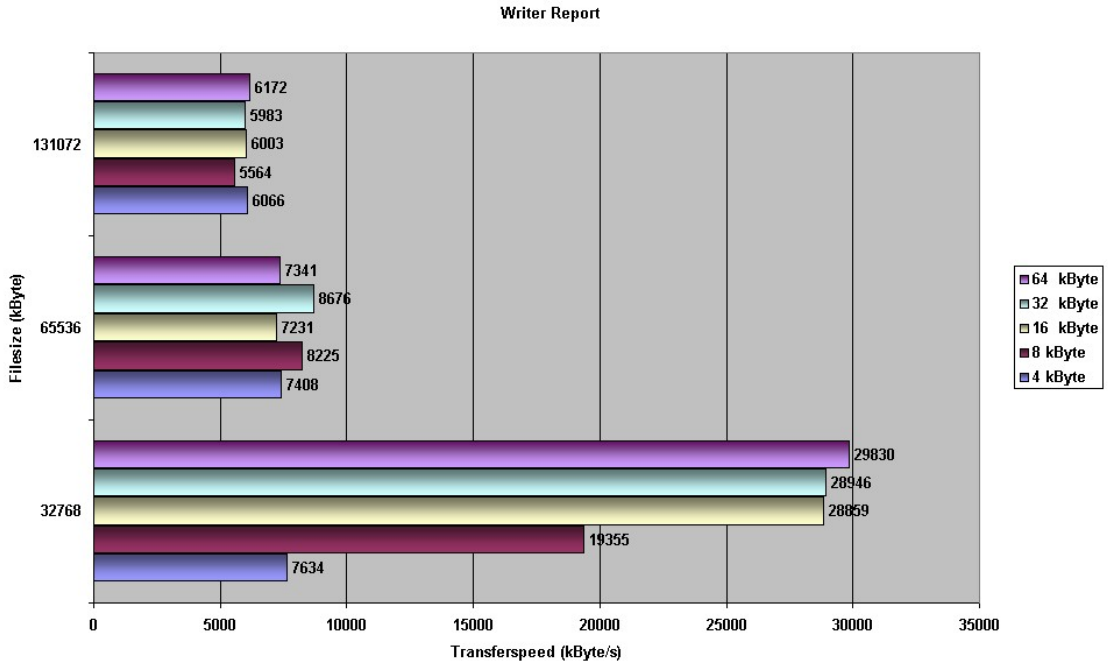

Figure 5.3: Writer performance, measured with the IOzone benchmark

when testing the software on different machines. Scores are calculated by the different tests. They can be uploaded to the SPEC website to compare the own setup with the setup of others.

Figure 5.4 shows the SPECjvm score diagram of the tested workstation. The scores are listed for each Java test, like the performance of the JIT compiler, Java cryptographic functions, scientific calculations (with single and double precision), etc. They have only a relative meaning for comparison usage.

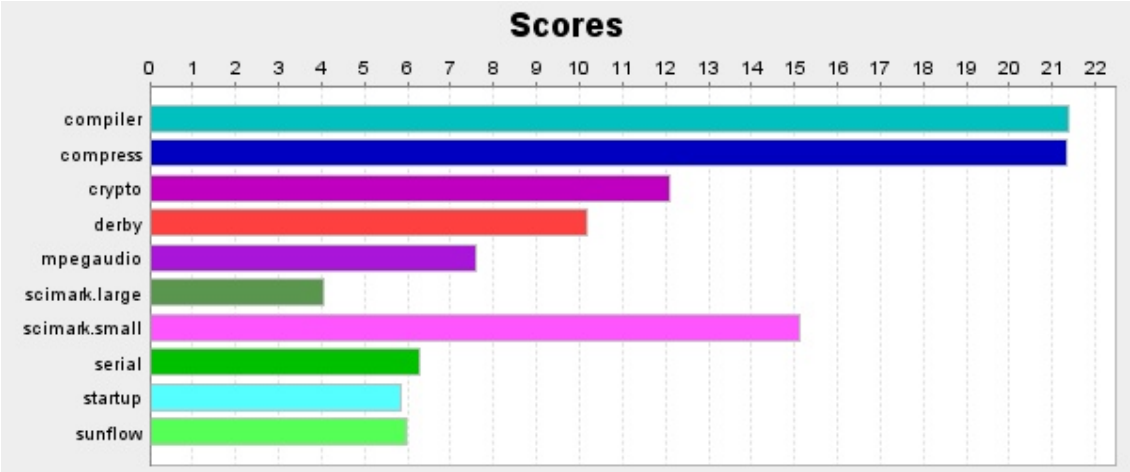

Figure 5.4: Result overview of the SPECjvm2008 on the test machine

5.5.6 Test environment configurations

To reproduce the test setup, the following major configuration changes were made:

- In general, I was aware not to run unnecessary programs when doing performance tests. Every power-management feature (e.g. CPU speed throttling, spin down hard disk) was disabled and the whole system was running in full speed. Also recommended is the unload of unneeded drivers.
- The JVM 1.5 for Window (32-bit) binary was used as it is, with a maximum heap size of 64 MBytes. This is important to simulate a standard configuration, user usually have on their workstation.
- Tomcat was started with the VM argument "-XX:MaxPermSize=256m" for providing enough space for the Tomcat itself.
- The parameter "-Dcom.sun.management.jmxremote" was added directly to the "startup.bat" file in the Tomcat binary directory below the JVM option section. This allows remote monitoring with jConsole, to see the memory and class usage over time.
- MySQL was used with the InnoDB storage engine. This is the default setting in MySQL version >5.0 .

- For the execution of the SPECjvm2008 benchmark test, Windows XP was started with the default minimum runtime configuration using the "msconfig" tool to choose the "Diagnose Boot". In this runtime mode, only the minimum necessary drivers and applications are loaded. This was done to get a representative test result.
- The SPECjvm2008 was started by executing the command **"java -Xmx400m -jar SPECjvm2008.jar"** on the command shell as recommended by the Standard Performance Evaluation Corporation. Otherwise the heapsize would not have been big enough.
- IOzone was configured for simulating the reading of a file with the size of the expected WebRowSet file using different block sizes. So it was started with the command line

  "iozone -Rab PATH_TO_XLS_RESULTFILE -i 0 -i 1 -+u -f PATH_TO_TESTFILE -q 64k -n 32M -g 204800 -z"

  - -R Generate Excel report
  - -a Auto mode
  - -b Filename Create Excel worksheet file
  - -i Test to run (0=write/rewrite, 1=read/re-read, 2=random-read/write 3=Read-backwards, 4=Re-write-record, 5=stride-read, 6=fwrite/re-fwrite 7=fread/Refread, 8=random_mix, 9=pwrite/Repwrite, 10=pread/Re-pread 11=pwritev/Repwritev, 12=preadv/Repreadv)
  - -u Enable CPU utilization output (Experimental)
  - -f filename to use
  - -q Set maximum record size (in Kbytes) for auto mode (or #m or #g)
  - -n Set minimum file size (in Kbytes) for auto mode (or #m or #g)
  - -g Set maximum file size (in Kbytes) for auto mode (or #m or #g)
  - -z Used in conjunction with -a to test all possible record sizes

5.6 Results

The results of the previous tests are now shown and interpreted in this section. The conclusions are described in detail at the beginning of the next chapter.

5.6.1 WebRowSet implementation

The ramp up time for accessing the first row is very fast. Compared to the in-core architecture of the Sun WebRowSet implementation, the novel approach is quite independent from the size of the WebRowSet file concerning the access time of the first row. That is caused by the direct access without loading the whole document. As expected, the random access time between a fully loaded WebRowSet file with Sun's implementation and a WebRowSet file with the novel implementation,

is much higher, because a limited buffer of 64 kBytes is used instead of loading the whole file into memory (in-core approach).

Figure 5.5 shows the heap memory consumption over time with the standard implementation of Sun, while Figure 5.6 does the same with the novel developed one. In both tests, a WebRowSet file of 10000 data lines was used, that has been created by the modified WEKA generator for artificial data. This WebRowSet file also contains "null" values for the second and eight column. During the loading process in the standard implementation, where the data structure is constructed in memory, around 5.2 MBytes are used on heap, while the novel implementation stays around 1.8 MBytes. As already said, the elapsed time of constructing the data structure in memory has also be taken into account when comparing these two approaches. The standard implementation of Java 1.5 needed over 22 seconds till getting ready.

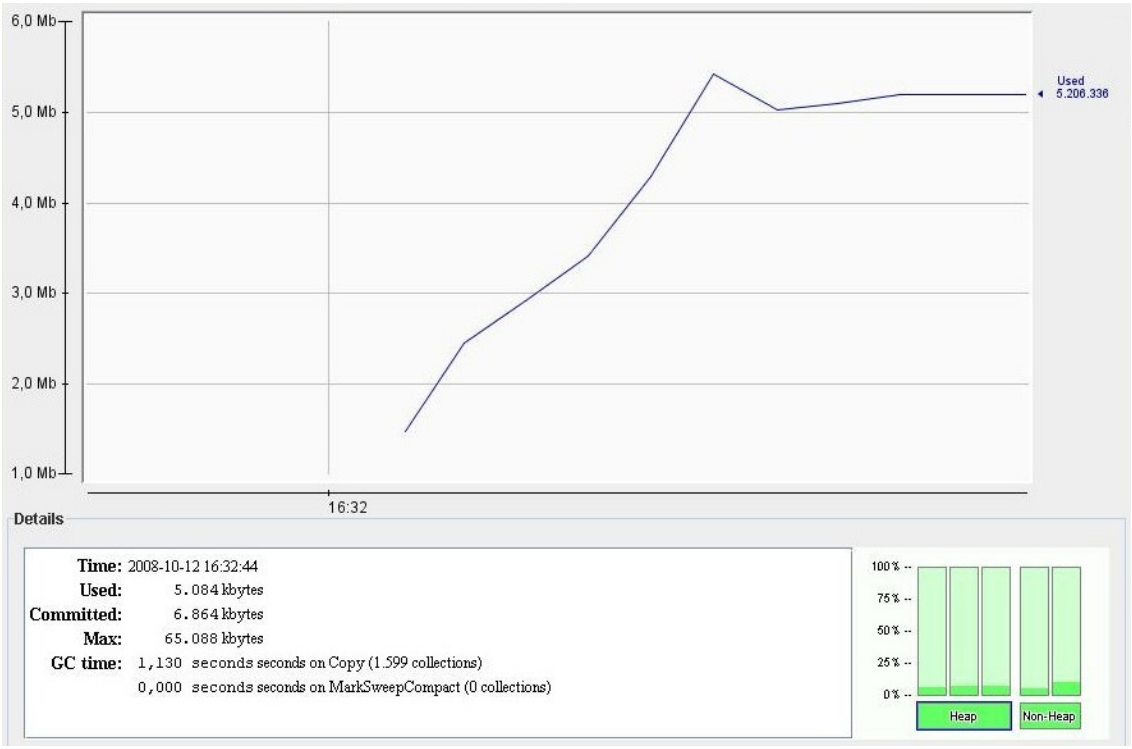

Figure 5.5: Heap space consumption on 10k rows WebRowSet with the original implementation

For the test of reading in a WebRowSet file, that contains 250000 Rows of artificial data (generated for the GridMiner project), the standard JVM heap size of 64 MBytes was increased to 256 MBytes. This was done by using the JVM parameter "-Xmx256m". The file itself contains 11 columns, that hold float and integer data including "null" values. The chosen file size is 92 MBytes and cannot be loaded with the in-core implementation of the WebRowSet interface.

Figure 5.7 shows the first run with the standard implementation. It can be seen how continuously the heap is filled by building up the memory structure. It took 9.6 minutes before it was possible to navigate within the XML file (elapsed time of the readXML function). During this phase, over 150 MBytes were used for the data structure. After this was done, the navigation through this

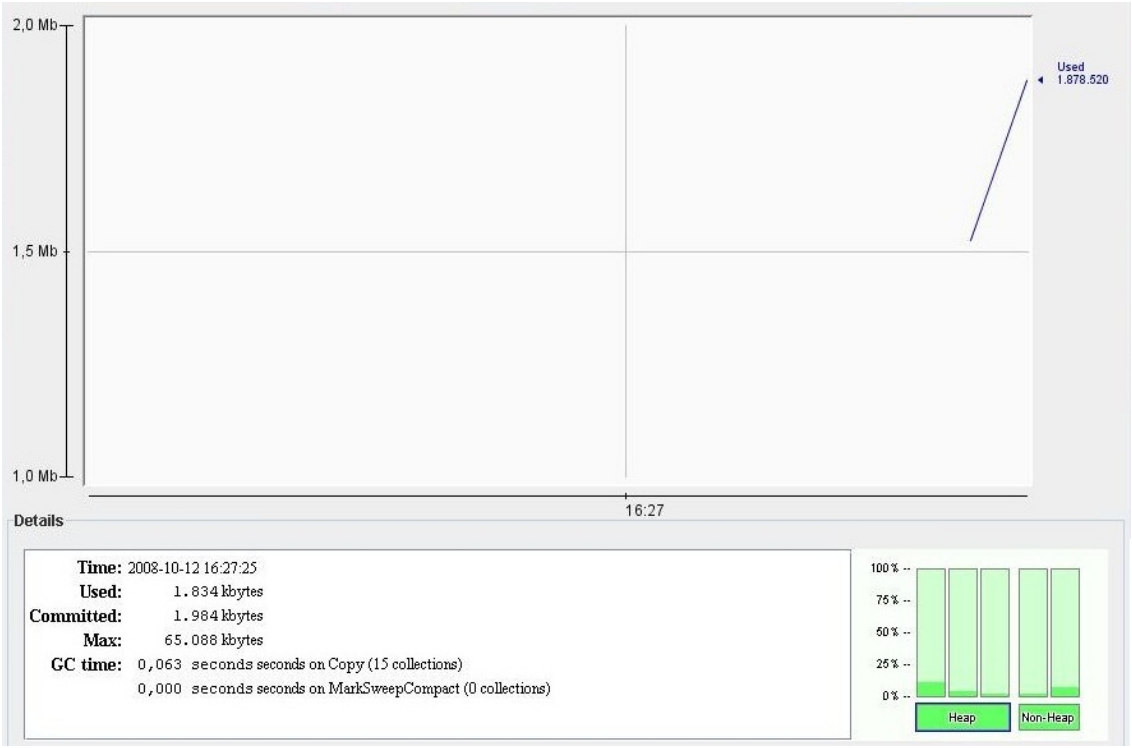

Figure 5.6: Heap space consumption on 10k rows WebRowSet with the novel implementation

data started. It took about 9 seconds for iteration and used again over 70 MBytes additional heap space (this is what the peak at the end shows).

The test with the modern implementation demonstrated in Figure 5.8 shows, that the memory consumption stays below 2 MBytes and is not increasing constantly over time. The decreasing parts of the graph are caused by the garbage collector, which removes unneeded objects from heap. The time for being ready (till it is possible to access the first row) is around 17 ms. The iteration through the rows was finished in 22 seconds.

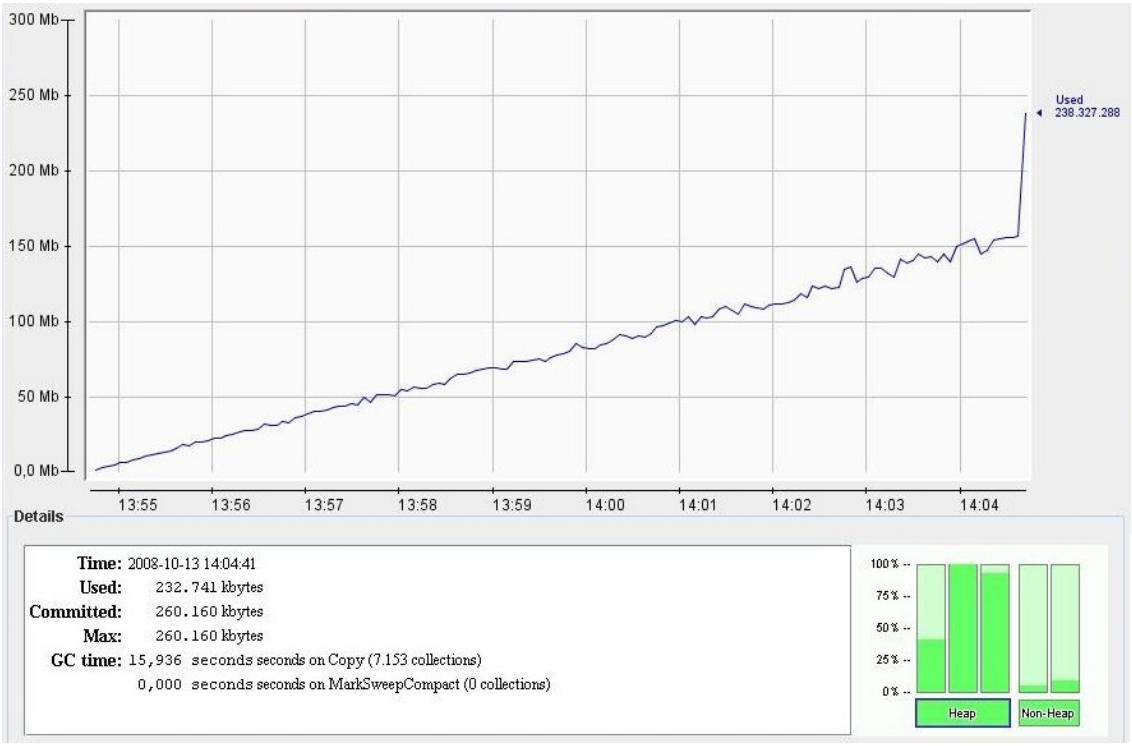

Figure 5.7: Heap space consumption on 250k rows WebRowSet with the original implementation

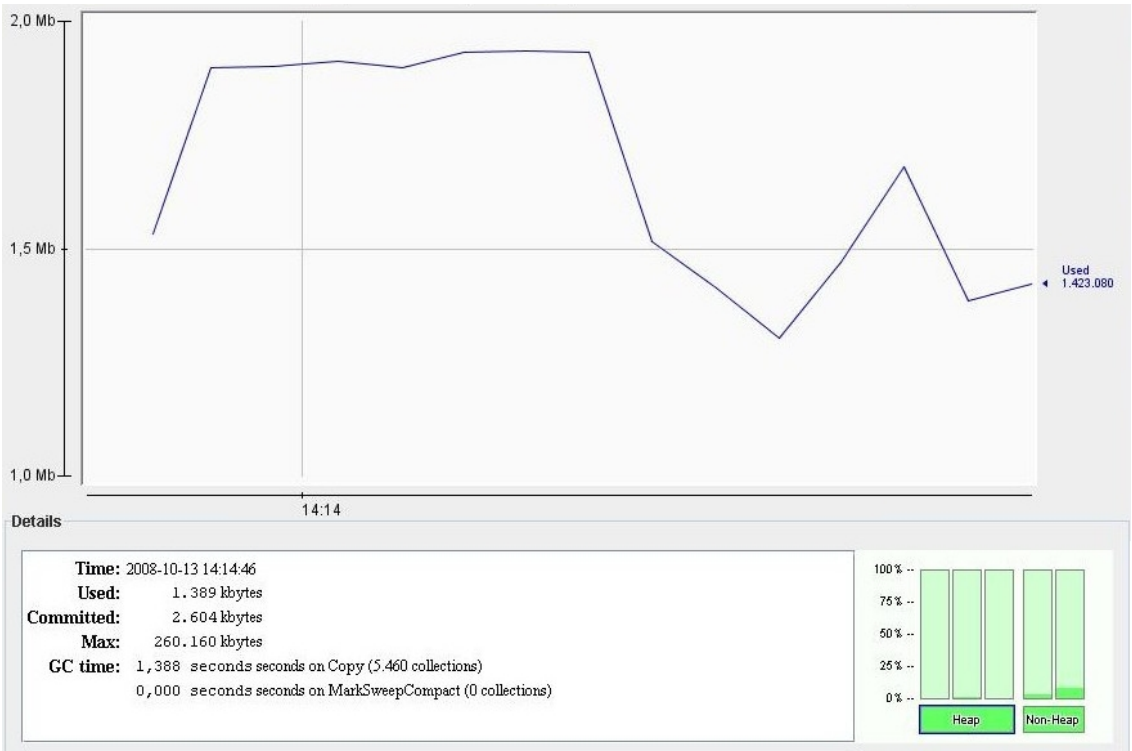

Figure 5.8: Heap space consumption on 250k rows WebRowSet with the novel implementation

5.6.2 Statistical metadata activity

To demonstrate the memory consumption over time of the client and server side, I started the Apache Tomcat with the enabled JMX feature (described in the "test configurations" section). Doing this, I was able to monitor the memory usage of the client- and server-side synchronously.

The results of the performance measurements of the novel statistics activity are illustrated in Figure 5.9. The client- and server-side diagrams were copied into the same figure to compare the time base. To avoid influences of the Apache Tomcat server during startup time, I waited about one minute till OGSA-DAI is loaded and initialized completely. This is represented in Figure 5.9 by the different starting points of the client- and server-side graph at the beginning.

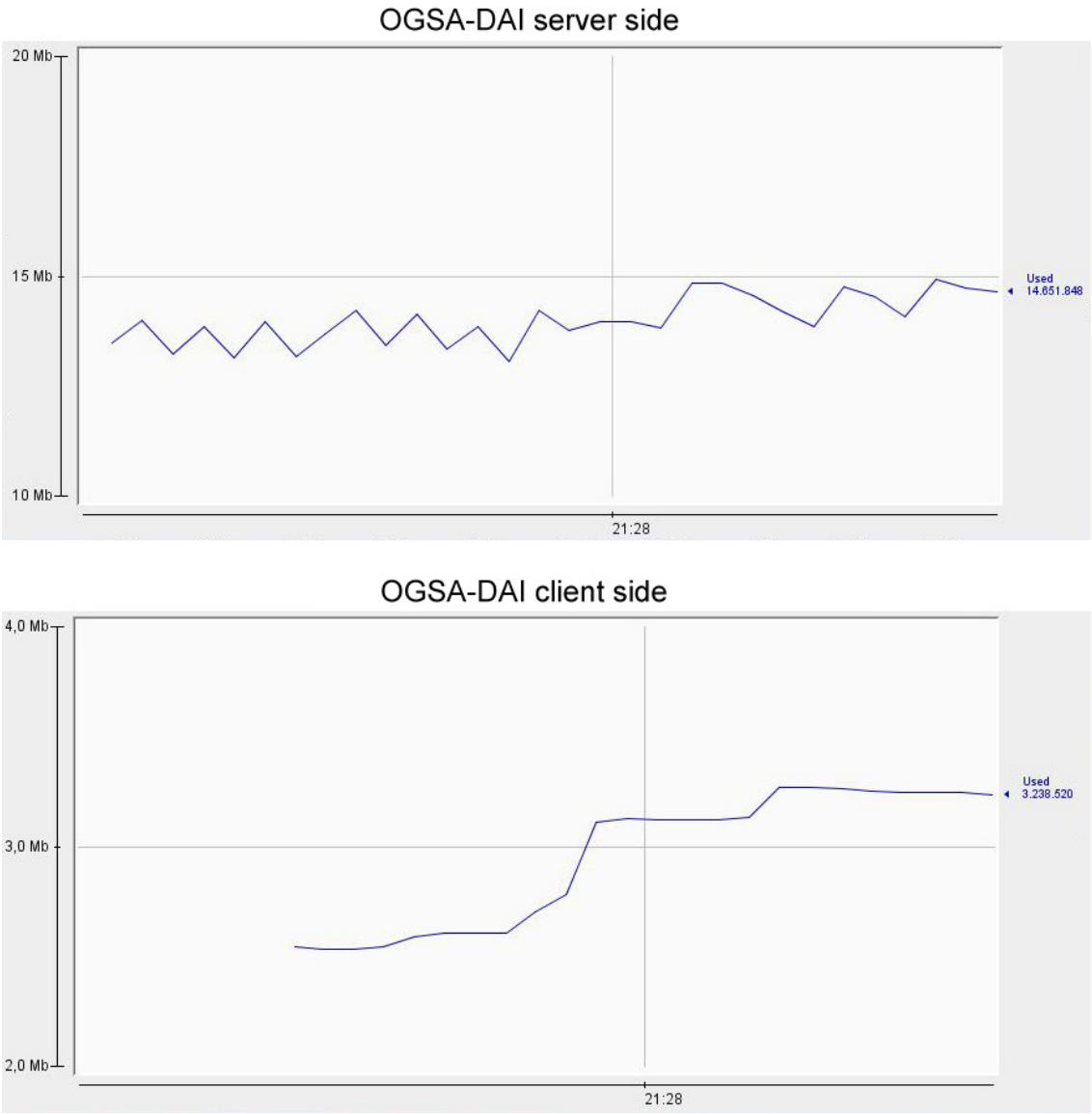

Figure 5.9: Heap space consumption of MetaStatisticsNG activity and client on 1k rows

The last test was repeated by using a data resource with 10000 rows instead of 1000 rows. Figure 5.10 shows that during server side processing, the amount of reserved heap space stays constant. This indicates a good scalability feature. The falling edges on the client side is caused by the garbage collection. In my test setup, client- and server side are running on the same machine due to of easier handling of the environment. Because it is running on a dual processor system, each JVM task can be run on a dedicated CPU (assigned by the operating system). The performance improvement can be shown indirect by the higher CPU usage on the Java process that runs the Apache Tomcat server. During the creation of the statistics, the client-side process has 2-4% CPU load, while the server-side has around 50% CPU load.

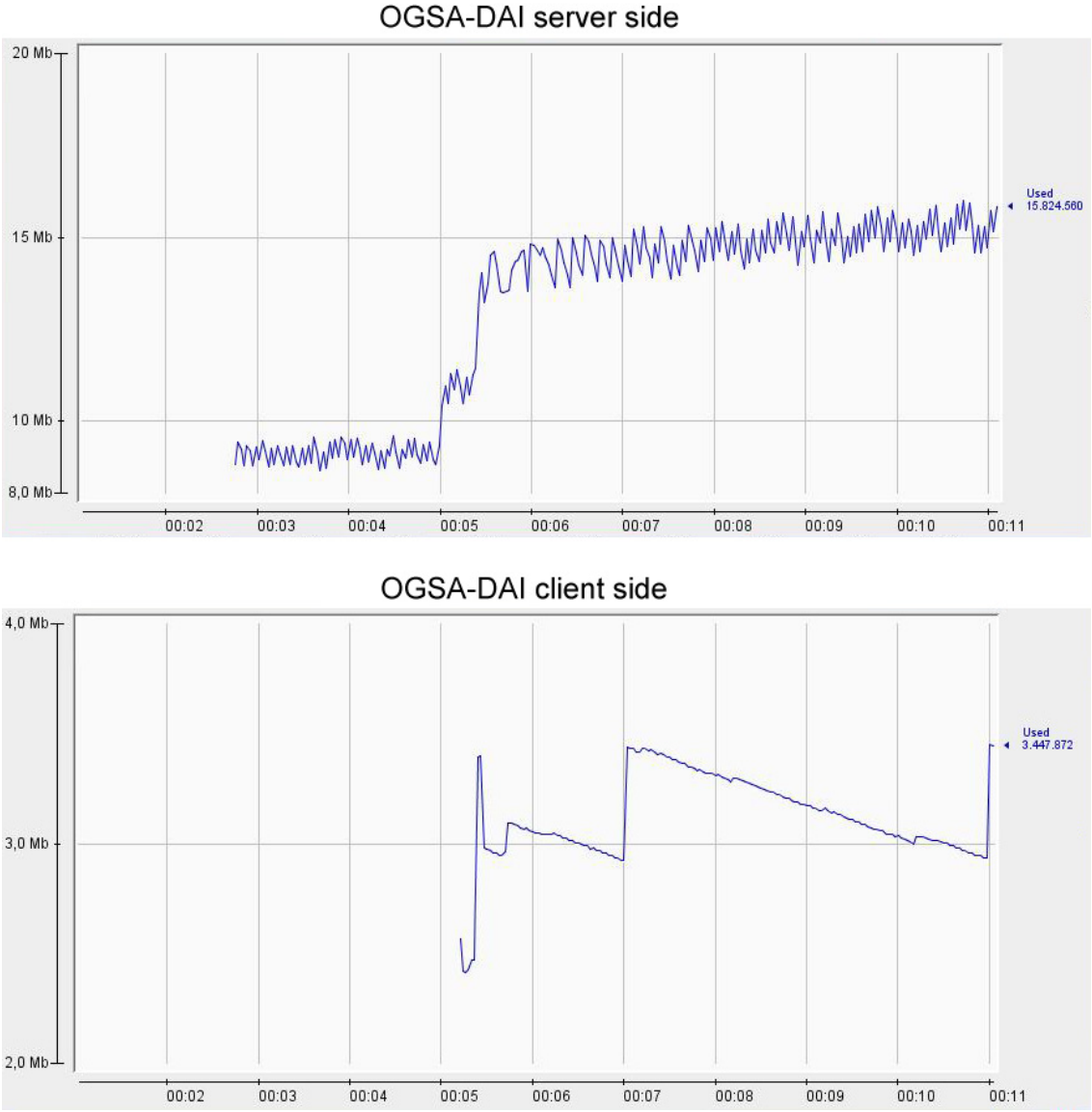

Figure 5.10: Heap space consumption of MetaStatisticsNG activity and client on 10k rows

Chapter 6

Conclusions and Future Work

The handling of huge datasets with limited system resources is an exciting challenge and requires the combination of several research areas. The major point is to find a balance between performance and memory consumption. In terms of data preprocessing the contribution to the WebRowSet exchange format shows that it is mandatory to combine the novel parsing design with the statistical metadata. It can be followed that the novel parsing approach provides a significant improvement concerning memory consumption and ramp-up speed. The out-of-core design of the implemented software components offers a better scalability. At the end of this chapter, suggestions of further enhancements of the developed software components are presented.

6.1 Lessons Learned

I did not find any implementation of a XML parsing API that allows random access (forward and backward navigation inside the XML file) by a sequential parsing method as expected. Most of the libraries that claim to be able to handle huge XML files are DOM based models, enhanced with intelligent compression algorithms. These libraries do not increase the maximum manageable file size significantly. It would have been interesting to be able to compare the implementation of the Index Based Parser (IBP) [61] API with this work. This was not possible due to restrictions in the usage of the source code.

One unexpected insight was that the requirements for data preprocessing on huge data were not well explored. That is why it was interesting for me to get in touch with this inter-disciplinary research area and the aspects which have to be taken into account when working with huge datasets. Some preprocessing methods like clustering or classification cannot be executed sequentially (in only one single iteration). They need sorted data or several loops through the data. This has negative effect on both - memory consumption (e.g. for sorting) and runtime performance (if a high number of iterations have to be done). One very interesting source were the discussions with people from the HEPHY (High Energy Physics) Institute of the Austrian Academy of Sciences [15], because they are experienced in treating huge datasets (> TByte).

Also the calculation of statistical values have to be done in an iterative (sequential) way. It was exciting to learn new statistical approaches of getting useful statistical attributes without lots of iterations (e.g. the mathematical term of the "displacement law" for the calculation of the variance).

The first guess of switching between the column-based and row-based indexing performed very poorly. During column-based indexing by using the memento pattern, for parsing 100 rows with 10 columns each, around 80000 memento objects were created. Although garbage collection is fast, the creation of these huge amount of objects resulted in a time consumption of 4-9 seconds per 100 rows. When using only row-based indexing, parsing the same constellation takes 16 ms. This change allows to load a WebRowSet file with 250000 rows in 24 seconds. This is a very usable value. It has to be added, that a file with only 10 columns is not a comparable layout, because real world data for data mining usually contains much more dimensions. If so, the reduction of the dimensions (e.g. via PCA) could give a good support for preprocessing.

Concerning the development environment, I recommend using test-driven development tools like EasyMock and JUnit. Lots of errors could be detected much faster, if it is possible to test modules in a fine-grained way. This means clamping the object under test between mock objects to simulate the interface. The testing with this approach is much faster and no integration step is needed. The unit tests with JUnit forced me to think about the expected behavior of the interface first before starting coding.

The usage of design patterns increased the code quality of the software engineering part. The applied engineering concepts also guarantee a usable level of scalability of the design for future enhancements. Java Generics, available since Java 1.5, have been used in the implementation. This feature avoids explicit type conversion, but still provides type safety. Using Java Generics increases the readability of the source code and also results in an increased speed of the compiled code. Another tuning method that was used is the usage of Java Enum types combined with switch cases. With this combination the code gets a clear structure and using switch cases is faster than a cascade of "if" and "else" statements.

6.2 Conclusions

The research presented in this thesis showed, that the OGSA-DAI framework produces WebRowSet files that are not completely compatible to Sun's specification of the WebRowSet format. So the output WebRowSet files could not be loaded by the standard Java implementation WebRowSetImpl.

The novel implementation of the WebRowSet format was developed with the focus on the use cases for data preprocessing and not for 100% functionality to the interface. The constructors of the WebRowSet interface expect only two allowed object types - Inputstream and Reader. The difficulty about these classes are, that they do not provide a flexible random access. Reader and Inputstream allow marking positions (by using the Java integer type) for rewinding the stream to them. Because of the variable range of integer, only a filesize of 4 Gbyte would be possible with this access method. That's why a new constructor was used, taking a RandomAccessFile as parameter.

For the software design it could be seen, that buffering I/O operations helps a lot to get better performance. Although the first revision of the statistics activity and the WebRowSet implementation were without any buffering, read buffers for the SAX parser and read/write buffers for the Median calculations were implemented. The approach of a row cache inside the WebRowSet implementation failed, because it is not possible to determine the point of time for the garbage collection. For a more advanced cache architecture the re-implementation of the garbage collector is necessary.

To summarize, the parsing speed of the novel out-of-core implementation is lower than the Sun implementation and depends heavily on the performance of the file system (secondary storage). For the practical use case scenarios it provides a better performance when accessing huge XML files. As the tests showed, the ramp-up time (the time the RowSet cursor needs to reach the first element of the dataset) is significantly lower. The memory consumption is only affected by the cached rows and also here the swapping allows to write back new inserted or changed rows to the file system. With these features, operating on huge WebRowSet files is possible in a comfortable way.

The out-of-core approach followed by the design of the metastatistics activity, allows a space efficient calculation on the server side. This efficiency is provided by the implementation itself and by a fine-grained definition of statistical operations.

6.3 Suggestions for Future Work

Some investigation has to be done if a complete implementation of the WebRowSet interface is needed, because the interface itself defines limitations on random access (the initialization of the constructor using a Reader class does not allow backward navigation). Although the caching of I/O increased the reading performance of the underlying SAX parser, the complex navigation of accessing the single rows should be re-factored by getting a more lightweight call-tree. Another enhancement would be a more sophisticated caching model. At the moment a write-through cache mechanism is used. This only improves the reading performance. Additional performance increase can be reached by using write caches. This means, some extra effort has to be spent on guaranteeing cache coherence.

For OGSA-DAI service it would be beneficial, if Java's Remote Method Invocation (RMI) [54] feature could be used to transfer calculation objects to the server side using serialization of method objects. This would allow the user to use new functionality without re-factoring and deploying the whole activity. By defining an enhanced interface, this solution would be very flexible for dynamical enhancing a deployed activity, that could not be exchanged on the fly.

The calculation of the statistical metadata could be more parallelized by using a multi-threaded architecture, which increases the scalability on massive multi-processor system architectures. I did a short try by simply putting the calculation objects into Java threads and building a dispatcher thread that passes the new data value to the affected objects. The needed synchronization produced a high overhead, which decreased the expected performance gain. I suggest to reduce the dependency of the statistical functions and find a balance between independent, but heavier

statistical implementation classes and slim implementations with a higher dependency to other components. It has to be taken into account that the mapping of Java threads to OS threads depends on the implementation of the JVM. It is more efficient using direct mapping instead of using so-called “green threads”, which are only simulated by the JVM itself.

There is still space for improvements in the section of the median calculation. The U.S. National Institute of Standards and Technology recommends the search algorithm “Interpolation search” [5] for a better performance on huge lists. Another theory is the more sophisticated concept of holding parts of the most likely values in memory by using a windowing approach.

The data generator for artificial test data could be enhanced by allowing a more flexible configuration of the columns by storing the parameters into a configuration file. It was very hard to find any open source generator for synthetic data. So the currently used source generator only provides basic functionality, resulting in the difference between synthetic and real word data. One central difficulty is the random generator, which does only generate pseudo-random data (referring to Java API 1.3 documentation [43]). Other Random Java libraries could be evaluated to create higher quality random values. Shrinking the difference between synthetic and real data is a general requirement of scientific models.

For future developments, I recommend to work in a pure 64-bit environment. This allows to address more than 4 GBytes of memory. Also mathematical operations on ’double’ and ’long’ 8-Byte Java types benefit from a 64-bit architecture, because they do not have to be split into 4-Byte operations. 64-bit desktop operating systems like Windows Vista and Linux are available now and got into productive state. Also every desktop CPU (Intel Core 2, AMD Phenom) supports 64-bit instructions and 64-bit drivers are well supported by the hardware vendors. This allows a cost effective workstation setup.

List of Abbreviations

ADMIRE	Advanced Data Mining and Integration Research for Europe
API	Application Programming Interface
AWS	Amazon Web Services
AXIS	Apache eXtensible Interaction System
CERN	Conseil Européen pour la Recherche Nucléaire
CMS	Code Management System
CPU	Central Processing Unit
DBMS	DataBase Management Systems
DMI	Data Mining and Integration
DOM	Document Object Model
DRER	Data Request Execution Resource
DRES	Data Request Execution Service
DTD	Document Type Definition
EGEE	Enabling Grids for E-sciencE
FSB	Front Side Bus
GC	Garbage Collector
GOF	Group Of Four
HTTP	Hyper Text Transfer Protocol
IBP	Index Based Parser
IQR	Inter Quartile Range
IT	Information Technology
JAMA	Java Matrix Package
JAXB	Java Architecture for XML Binding
JDBC	Java DataBase Connectivity
JDK	Java Developer Kit
JIT	Just In Time
JVM	Java Virtual Machine
KDD	Knowlede Discovery in Databases
LCG	LHC Computing Grid
LHC	Large Hadron Collider
LIFO	Last In First Out
MAD	Median Absolute Deviation
MD5	Message-Digest Algorithm 5

NAS ..... Network Attached Storage
NAT ..... Network Address Transformation
NG ..... Next Generation
NIST ..... National Institute of Standards and Technology
OASIS ..... Organization for the Advancement of Structured Information Standards
OGSA ..... Open Grid Service Architecture
OGSA-DAI ..... Open Grid Service Architecture Data Access and Integration
OLAP ..... Online Analytical Processing
OOP ..... Object Oriented Programming
OS ..... Operating System
PCA ..... Principle Component Analysis
PMML ..... Predictive Model Markup Language
RAM ..... Random Access Memory
REST ..... REpresentational State Transfer
RMI ..... Remote Method Invocation
SAS ..... Serial Attached SCSI
SATA ..... Serial Advanced Technology Attachment
SAX ..... Simple Api for XML
SOA ..... Service Oriented Architecture
SOAP ..... Simple Object Access Protocol
SPEC ..... Standard Performance Evaluation Corporation
SQL ..... Structured Query Language
SSD ..... Solid State Drives
STAX ..... Streaming API for XML
TBI ..... Traumatic Brain Injuries
USB ..... Universal Serial Bus
VO ..... Virtual Organization
WEKA ..... Waikato Environment for Knowledge Analysis
WSRF ..... Web Services Resource Framework
XML ..... eXtensible Markup Language

Bibliography

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[2] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993. Special issue on Learning and Discovery in Knowledge-Based Databases.

[3] Globus Alliance. Globus Toolkit homepage. [http://www.globus.org/toolkit; accessed 31-October-2008].

[4] Malcolm Atkinson, Peter Brezany, Liangxiu Han, Ally Hume, Dave Snelling, and Jano van Hemert. ADMIRE White Paper: Motivation, Strategy and Impact, 2008. [http://www.admire-project.eu; accessed 26-October-2008].

[5] Paul E. Black. Interpolation search, 2007. [http://www.nist.gov/dads/HTML/interpolationSearch.html; Dictionary of Algorithms and Data Structures; accessed 16-November-2008].

[6] Steven R. Brandt. Java tip 128: Create a quick-and-dirty XML parser, 2002. [http://www.javaworld.com/javatips/jw-javatip128.html?page=1; accessed 26-October-2008]; Article including example of a non validating XML SAX parser].

[7] Peter Brezany, A. Min Tjoa, Helmut Wanek, and Alexander Wöhrer. Novel mediator architectures for grid information systems. In *In Future Generation Computer Systems*, 2005.

[8] CERN. LCG - Computing Grid, 2008. [http://lcg.web.cern.ch/lcg/overview.htm; accessed 26-October-2008].

[9] CERN. LHC - Large Hadron Collider, 2008. [http://lhc.web.cern.ch/lhc/; accessed 18-November-2008].

[10] Standard Performance Evaluation Corporation. Specjvm2008, 2008. [http://www.spec.org/; accessed 02-November-2008].

[11] I. Elsayed and P. Brezany. On-line analytical mining of association rules on the grid. In *Master Thesis, University of Vienna*, 2005.

[12] Ralf Enderle. GridComputing, 2008. [http://www-vs.informatik.uni-ulm.de/DE/dept/staff/schmidt/pubs/doc/techreps/VS-R07-2007.pdf; accessed 26-October-2008].

[13] Dr Rob Baxter EPCC. Admire Online, 2008. [http://www.admire-project.eu/; accessed 26-October-2008].

[14] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, IRVINE, 2000.

[15] HEPHY Institute for High Energy Physic. Homepage. [http://www.hephy.at/; accessed 14-November-2008].

[16] Ian Foster and Carl Kesselman. The Globus toolkit. pages 259–278, 1999.

[17] Apache Software Foundation. Apache Tomcat. [http://tomcat.apache.org/; accessed 14-November-2008].

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software.* 1996.

[19] Mark Graves and Charles F. Goldfarb. *Designing XML Databases.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[20] Data Mining Group. PMML 3.2 specification. [http://www.dmg.org/v3-2/GeneralStructure.html; accessed 4-November-2008].

[21] Wolfgang Gentzsch (Hg.) Heike Neuroth, Martina Kerzel. *Die D-Grid Initiative.* Universitätsverlag Göttingen, Göttingen, Germany, 2007.

[22] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[23] Erich Gamma Kent Beck. The JUnit Cookbook, 2008. [http://junit.sourceforge.net/doc/cookbook/cookbook.htm; accessed 26-October-2008].

[24] Noah Mendelsohn(IBM) Jean-Jacques Moreau(Canon) Henrik Frystyk Nielsen(Microsoft) Anish Karmarkar(Oracle) Yves Lafon(W3C) Martin Gudgin(Microsoft), Marc Hadley(Sun Microsystems). SOAP specification v1.2, 2007. [http://www.w3.org/TR/soap12-part1/; accessed 26-October-2008].

[25] nexB. EasyEclipse. [http://www.easyeclipse.org/site/distributions/server-java.html; accessed 4-November-2008].

[26] NIST. Jama : A Java matrix package. [http://math.nist.gov/javanumerics/jama/; accessed 4-November-2008].

[27] William D. Norcott. and Don Capps. Iozone - a filesystem benchmark tool, 2008.

[28] OASIS. WSRF - Web Services Resource Framework. [http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf; accessed 14-November-2008].

[29] OASIS. OASIS SOA reference model, 2008. [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm; accessed 26-October-2008].

[30] The University of Edinburgh. OGSA-DAI 3.0 Documentation: OGSA-DAI components overview, 2007. [http://www.ogsadai.org.uk/documentation/ogsadai3.0/ogsadai3.0-axis/; accessed 10-September-2008].

[31] Institute of Scientific Computing. GridMiner online, 2008. [http://www.gridminer.org/; accessed 26-October-2008].

[32] University of Waikato. Weka - machine learning. [http://www.cs.waikato.ac.nz/~ml/weka/; accessed 16-November-2008].

[33] Tammo Freese OFFIS. EasyMock for JUnit, 2008. [http://www.easymock.org/EasyMock2_4_Documentation.html; accessed 26-October-2008].

[34] David L. Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer, Berlin Heidelberg, Germany, 2008.

[35] Ed Ort and Bhakti Mehta. Java architecture for XML binding(JAXB), 2003. [http://java.sun.com/developer/technicalArticles/WebServices/jaxb/; accessed 26-October-2008].

[36] A Min Tjoa Peter Brezany, Ivan Janciak. GridMiner: An advanced support for e-Science analytics, 2007. ; Book chapter in Data Mining Techniques in Grid Computing Environments.

[37] Ivan Janciak Peter Brezany and A Min Tjoa. GridMiner: A fundamental infrastructure for building intelligent grid systems. pages 150–156. ACM,IEEE,WIC, 2005.

[38] Michaela Pfeifer. Data preprocessing for knowledge discovery in grid databases, 2007. ; Master Thesis, University of Vienna.

[39] Dorian Pyle. *Data preparation for data mining*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[40] Roy W. Schulte and Yefim V. Natis. SSA research note spa-401-068, service oriented architectures, part 1. Technical report, the Gartner Group, 1996.

[41] Lindsay I. Smith. A tutorial on principal components analysis, 2002.

[42] Information Society. Extremeos - building and promoting a linux-based operating system to support virtual organizations for next generation grids, 2008.

[43] Sun. Java-api 1.3. [http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html; accessed 31-October-2008].

[44] Sun. Java-api 1.5. [http://java.sun.com/j2se/1.5.0/docs/api/; accessed 26-October-2008].

[45] Sun. Java Server Pages 2.1 Specification. [http://jcp.org/aboutJava/communityprocess/final/jsr245/index.html; accessed 14-November-2008].

[46] Sun. Java Servlet 2.5 Specification. [http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index.html; accessed 14-November-2008].

[47] Sun. Tuning garbage collection with the 5.0 java[tm] virtual machine. [http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html; accessed 26-October-2008].

[48] Jonathan Bruce (Sun). WebRowSet XML schema. [http://java.sun.com/xml/ns/jdbc/webrowset.xsd; accessed 26-October-2008].

[49] Anne Papakonstantinou Tamara Anthony Carter, Richard A. Tapia. Linear algebra; eigenvalues and eigenvectors. [http://ceee.rice.edu/Books/LA/eigen/; accessed 16-November-2008].

[50] Emerging Technologies and Infrastructures. Enabling Grids for e-Science, 2008. [http://public.eu-egee.org/; accessed 26-October-2008].

[51] Douglas Thain and Miron Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[52] W3C. Web Services. [http://www.w3.org/2002/ws/; accessed 7-November-2008].

[53] W3C. World wide web consortium homepage. [http://www.w3.org; accessed 26-October-2008].

[54] Wikipedia. RMI - remote method invocation. [http://en.wikipedia.org/wiki/Java_remote_method_invocation; accessed 31-October-2008].

[55] Wikipedia. Computer data storage — wikipedia, the free encyclopedia, 2008. [http://en.wikipedia.org/w/index.php?title=Computer_data_storage&oldid=237861429; accessed 12-September-2008].

[56] Wikipedia. Standard deviation — wikipedia, the free encyclopedia, 2008. [http://en.wikipedia.org/w/index.php?title=Standard_deviation&oldid=237074529; accessed 26-October-2008].

[57] Wikipedia. Varianz — wikipedia, die freie enzyklopädie, 2008. [http://de.wikipedia.org/w/index.php?title=Varianz&oldid=50634916; accessed 26-October-2008].

[58] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005.

[59] Alexander Woehrer, Peter Brezany, Lenka Novakov, and A Min Tjoa. D3G: Novel approaches to data statistics, understanding and preprocessing on the grid. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, pages 313–320, Washington, DC, USA, 2006. IEEE Computer Society.

[60] OGSA working group. The OGSA Architecture document. [http://www.ogf.org/documents/GFD.80.pdf; accessed 26-October-2008].

[61] Haihui Zhang, Xingshe Zhou, Yang Gang, and Xiaojun Wu. IBP: An index-based XML parser model. In Hai Jin, Daniel A. Reed, and Wenbin Jiang, editors, *NPC*, volume 3779 of *Lecture Notes in Computer Science*, pages 65–71. Springer, 2005.

Appendix A

Appendix A

A.1 WebRowSet XML Format

This section shows examples of WebRowSet XML file sections, described earlier in this thesis.

A.1.1 Properties section

This properties section shows the use SQL query and other information about the data source.

Listing A.1: Example of a WebRowSet properties section

```
<properties>
    <command>SELECT * FROM testdatabase WHERE id=10</command>
        <concurrency>1008</concurrency>
        <datasource>
            <null/>
        </datasource>
        <escape-processing>true</escape-processing>
        <fetch-direction>1000</fetch-direction>
        <fetch-size>0</fetch-size>
        <isolation-level>2</isolation-level>
        <key-columns>
    </key-columns>
        <map>
    </map>
        <max-field-size>0</max-field-size>
        <max-rows>0</max-rows>
        <query-timeout>0</query-timeout>
        <read-only>true</read-only>
        <rowset-type>ResultSet.TYPE_SCROLL_INSENSITIVE</rowset-type>
        <show-deleted>false</show-deleted>
        <table-name>
            <null/>
        </table-name>
        <url>
            <null/>
        </url>
        <sync-provider>
            <sync-provider-name>com.sun.rowset.providers.RIOptimisticProvider</sync-
                provider-name>
```

```
29              <sync-provider-vendor>Sun Microsystems Inc.</sync-provider-vendor>
30              <sync-provider-version>1.0</sync-provider-version>
31              <sync-provider-grade>2</sync-provider-grade>
32              <data-source-lock>1</data-source-lock>
33          </sync-provider>
34    </properties>
```

A.1.2 Metadata section

This metadata section defines two columns. One column is a numeric type, and the other stores string values.

Listing A.2: Example of a WebRowSet metadata section defining two columns

```
1  <metadata>
2      <column-count>2</column-count>
3          <column-definition>
4              <column-index>1</column-index>
5              <auto-increment>false</auto-increment>
6              <case-sensitive>false</case-sensitive>
7              <currency>true</currency>
8              <nullable>1</nullable>
9              <signed>true</signed>
10             <searchable>true</searchable>
11             <column-display-size>22</column-display-size>
12             <column-label>A</column-label>
13             <column-name>A</column-name>
14             <schema-name/>
15             <column-precision>0</column-precision>
16             <column-scale>0</column-scale>
17             <table-name/>
18             <catalog-name/>
19             <column-type>2</column-type>
20             <column-type-name>NUMBER</column-type-name>
21         </column-definition>
22         <column-definition>
23             <column-index>2</column-index>
24             <auto-increment>false</auto-increment>
25             <case-sensitive>true</case-sensitive>
26             <currency>false</currency>
27             <nullable>1</nullable>
28             <signed>true</signed>
29             <searchable>true</searchable>
30             <column-display-size>3</column-display-size>
31             <column-label>G</column-label>
32             <column-name>G</column-name>
33             <schema-name/>
34             <column-precision>3</column-precision>
35             <column-scale>0</column-scale>
36             <table-name/>
37             <catalog-name/>
38             <column-type>12</column-type>
39             <column-type-name>VARCHAR2</column-type-name>
40         </column-definition>
41 </metadata>
```

A.1.3 Data section

In the data section, the data values are stored as the content of XML elements. The "currentRow" element groups the "columnValue" elements to rows. As already mentioned, the recording of changes on WebRowSet data is done by using different tagnames like "insertRow". This feature is not used in the novel implementation.

Listing A.3: Example of a WebRowSet data section with 2 rows of 11 columns

```
<currentRow>
            <columnValue>17.983947</columnValue>
            <columnValue>0.371105</columnValue>
            <columnValue>5.95291</columnValue>
            <columnValue>0.453017</columnValue>
            <columnValue>3.712088</columnValue>
            <columnValue>0.999908</columnValue>
            <columnValue>5</columnValue>
            <columnValue>0</columnValue>
            <columnValue>8</columnValue>
            <columnValue>1</columnValue>
            <columnValue>0</columnValue>
        </currentRow>
        <currentRow>
            <columnValue>15.528123</columnValue>
            <columnValue>4.508194</columnValue>
            <columnValue>5.459609</columnValue>
            <columnValue>0.13358</columnValue>
            <columnValue>2.441908</columnValue>
            <columnValue>0.999359</columnValue>
            <columnValue>4</columnValue>
            <columnValue>0</columnValue>
            <columnValue>1</columnValue>
            <columnValue>1</columnValue>
            <columnValue>0</columnValue>
        </currentRow>
```

A.2 Metastatistics Activity

The novel Metastatistics Activity uses the PMML format as a standard for exchanging statistic model information between the client sub and the server implementation of the OGSA-DAI activity. Although it looks like an XML file, it is transferred as a serialized string inside a stream and not as a file.

A.2.1 PMML example

This PMML example show the definition of the standard statistic functions minimum, maximum, standard deviation, frequency and quartile. Furthermore four aggregator functions are defined to calculate the covariance matrix for column 1 and 2. It also contains an MD5 identifier in the description attribute of the header section.

Listing A.4: Created PMML

```
1  <PMML version="3.0" xmlns="http://www.dmg.org/PMML-3_1">
2    <Header copyright="www.gridminer.org" description="e28c8b0ed3e461d90c7a766d71070b9">
3      <Application name="Gridminer - Advanced Data Preprocessing diploma thesis" version="
           1.0"/>
4    </Header>
5    <DataDictionary>
6      <DataField name="column 2" displayName="column 2" dataType="string" taxonomy="[MIN,
           MAX, STDDEV, SUMSQUAREDDELTA, FREQ, QUARTILE]" optype="continuous"/>
7      <DataField name="column 1" displayName="column 1" dataType="string" taxonomy="[MIN,
           MAX, STDDEV, SUMSQUAREDDELTA, FREQ, QUARTILE]" optype="continuous"/>
8    </DataDictionary>
9    <Extension extender="AggreatorFunctioncolumn 2/column 2" name="column 2/column 2"
         value="[SUMXYDELTA, COVARIANCEXY]"/>
10   <Extension extender="AggreatorFunctioncolumn 1/column 2" name="column 1/column 2"
         value="[SUMXYDELTA, COVARIANCEXY]"/>
11   <Extension extender="AggreatorFunctioncolumn 2/column 1" name="column 2/column 1"
         value="[SUMXYDELTA, COVARIANCEXY]"/>
12   <Extension extender="AggreatorFunctioncolumn 1/column 1" name="column 1/column 1"
         value="[SUMXYDELTA, COVARIANCEXY]"/>
13 </PMML>
```

A.2.2 Metastatistics file

The results of the Metastatistics activity are stored in a self defined XML structure. This structure groups the standard and aggregator function results by dedicated elements. It also contains the MD5 algorithm hash code of the used SQL query that was used to create the dataset to provide a mapping between the metastatistic file and the corresponding WebRowSet file.

Listing A.5: Created metastatistics XML file containing the calculated results

```
<?xml version="1.0"?>
<metastatistics DataSourceID="e28c8b0ed3e461d90c7a766d71070b9">
  <stdfunctions>
    <column name="column 1">
      <function name="MIN">20000.0</function>
      <function name="MAX">150000.0</function>
      <function name="AVERAGE">86734.25350480416</function>
      <function name="STDDEV">1.4015262028244455E9</function>
      <function name="SUMSQUAREDDELTA">1.3915904469372654E12</function>
      <function name="FREQUENCY_total">1000</function>
      <function name="FREQUENCY_missing">0</function>
      <function name="FREQUENCY_distinct">976</function>
      <function name="QUARTILE_Q1">52793.318054</function>
      <function name="QUARTILE_Q3">119556.2386715</function>
      <function name="QUARTILE_Median">88687.42499</function>
    </column>
    <column name="column 2">
      <function name="MIN">0.0</function>
      <function name="MAX">75000.0</function>
      <function name="AVERAGE">17813.210404961006</function>
      <function name="STDDEV">6.132784550482426E8</function>
      <function name="SUMSQUAREDDELTA">6.098014970632999E11</function>
      <function name="FREQUENCY_total">1000</function>
      <function name="FREQUENCY_missing">0</function>
      <function name="FREQUENCY_distinct">384</function>
      <function name="QUARTILE_Q1">0.0</function>
      <function name="QUARTILE_Q3">36870.74118725</function>
      <function name="QUARTILE_Median">0.0</function>
    </column>
  </stdfunctions>
  <aggrfunctions>
    <aggrkey name="column 1/column 1">
      <function name="SUMXYDELTA">1.3915904469372654E12</function>
      <function name="COVARIANCEXY">1.4015262028244455E9</function>
    </aggrkey>
    <aggrkey name="column 1/column 2">
      <function name="SUMXYDELTA">-6.960866142676609E11</function>
      <function name="COVARIANCEXY">-7.004466187493039E8</function>
    </aggrkey>
    <aggrkey name="column 2/column 1">
      <function name="SUMXYDELTA">-6.960866142676609E11</function>
      <function name="COVARIANCEXY">-7.004466187493037E8</function>
    </aggrkey>
    <aggrkey name="column 2/column 2">
      <function name="SUMXYDELTA">6.098014970632999E11</function>
      <function name="COVARIANCEXY">6.132784550482426E8</function>
    </aggrkey>
  </aggrfunctions>
</metastatistics>
```

A.2.3 JDBC mapping

The next mapping table shows how SQL types are related to the Java types. This information is used when interpreting the metadata section of the WebRowSet file.

Table A.1: Mapping table of SQL and Java for standard types

SQL types	Java types
VARCHAR, VARCHAR2	java.lang.String
CHAR	java.lang.String
DATE	java.sql.Date
DATE	java.sql.Timestamp
NUMBER	java.lang.Double , java.lang.Integer , java.lang.Long
TIME	java.sql.Time
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
BOOLEAN	java.lang.Boolean
INTEGER	java.lang.Integer
TIMESTAMP	java.sql.Timestamp
LONG	java.lang.Long
BLOB	Blob
CLOB	Clob

A.2.4 How to add a new statistic function to the activity

As described in my work, I distinguish between standard functions, that operate on a single column, and aggregator functions, that operate across several columns. The way implementing them is quite the same and consists of the following main steps:

- add a new Enum type
- implement the function interface
- create an entry in the “loadStatXFunc” function to define which constructor should be called
- add an entry to the “createXFunctionDependencyMap” function, to define the dependencies
- optional, add special function handling into the
  “writeXml” and “writeXmlDebug” method of the “MStatXml” class
- for testing, add a JUnit test into “MetaStatisticsNGActivityTest”

There are three functions which need to be implemented by the interface definition: “next(Object n)”, “getResult()” and “getName()”. The function “next(Object n)” is called by the StatisticProcessor, when it iterates through the Tuples of the input. The explicit cast of the Object inside the function has to be done to handle the correct Java type (e.g. Double). Inside the next-method, I am calculating the values in a sequential way to optimize the memory consumption and to avoid another iteration through all values. At the end of this method, the result is available and can be read by calling “getResult()”. The “getName()” method is used to resolve the statistical function

name after instantiating an object. Standard functions have a local variable called “mapFP” which is a HashMap that provides function references to other objects within the same column. This reference map is set inside the constructor when the statistic objects are instantiated by the StatisticProcessor. This allows to get dependent calculation results (e.g. the standard deviation object need the results from the average object). The dependency table guarantees the availability of the results from dependent objects.