Integration of Data Mining into Scientific Data Analysis Processes

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt
von
Dennis Wegener
aus
Siegburg

Bonn, 2012

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Stefan Wrobel
2. Gutachter: Prof. Dr. Oswaldo Trelles Salazar
Tag der Promotion: 28.11.2012
Erscheinungsjahr: 2012

Dennis Wegener

University of Bonn
Department of Computer Science III
Römerstraße 164
53117 Bonn
Germany
(from 10/2009)

and

Fraunhofer Institute for Intelligent Analysis
and Information Systems IAIS
Schloss Birlinghoven
53754 Sankt Augustin
Germany
(6/2006 - 5/2012)

and

GESIS - Leibniz Institute for the Social Sciences
Unter Sachsenhausen 6-8
50667 Köln
Germany
(from 6/2012)

dennis.wegener@gesis.org

Declaration

I, Dennis Wegener, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g. ideas, equations, figures, text, tables, programs) are properly acknowledged at the point of their use. A full list of the references employed has been included.

Abstract

In recent years, using advanced semi-interactive data analysis algorithms such as those from the field of data mining gained more and more importance in life science in general and in particular in bioinformatics, genetics, medicine and biodiversity. Today, there is a trend away from collecting and evaluating data in the context of a specific problem or study only towards extensively collecting data from different sources in repositories which is potentially useful for subsequent analysis, e.g. in the Gene Expression Omnibus (GEO) repository of high throughput gene expression data. At the time the data are collected, it is analysed in a specific context which influences the experimental design. However, the type of analyses that the data will be used for after they have been deposited is not known. Content and data format are focused only to the first experiment, but not to the future re-use. Thus, complex process chains are needed for the analysis of the data. Such process chains need to be supported by the environments that are used to setup analysis solutions. Building specialized software for each individual problem is not a solution, as this effort can only be carried out for huge projects running for several years. Hence, data mining functionality was developed to toolkits, which provide data mining functionality in form of a collection of different components. Depending on the different research questions of the users, the solutions consist of distinct compositions of these components.

Today, existing solutions for data mining processes comprise different components that represent different steps in the analysis process. There exist graphical or script-based toolkits for combining such components. The data mining tools, which can serve as components in analysis processes, are based on single computer environments, local data sources and single users. However, analysis scenarios in medical- and bioinformatics have to deal with multi computer environments, distributed data sources and multiple users that have to cooperate. Users need support for integrating data mining into analysis processes in the context of such scenarios, which lacks today. Typically, analysts working with single computer environments face the problem of large data volumes since tools do not address scalability and access to distributed data sources. Distributed environments such as grid environments provide scalability and access to distributed data sources, but the integration of existing components into such environments is complex. In addition, new components often cannot be directly developed in distributed environments. Moreover, in scenarios involving multiple computers, multiple distributed data sources and multiple users, the reuse of components, scripts and analysis processes becomes more important as more steps and configuration are necessary and thus much bigger efforts are needed to develop and set-up a solution.

In this thesis we will introduce an approach for supporting interactive and distributed data mining for multiple users based on infrastructure principles that allow building on data mining components and processes that are already available instead of designing of a completely new infrastructure, so that users can keep working with their well-known tools.

In order to achieve the integration of data mining into scientific data analysis processes, this thesis proposes an stepwise approach of supporting the user in the development of analysis solutions that include data mining.

We see our major contributions as the following: first, we propose an approach to integrate data mining components being developed for a single processor environment into grid environments. By this, we support users in reusing standard data mining components with small effort. The approach is based on a metadata schema definition which is used to grid-enable existing data mining components. Second, we describe an approach for interactively developing data mining scripts in grid environments. The approach efficiently supports users when it is necessary to enhance available components, to develop new data mining components, and to compose these components. Third, building on that, an approach for facilitating the reuse of existing data mining processes based on process patterns is presented. It supports users in scenarios that cover different steps of the data mining process including several components or scripts. The data mining process patterns support the description of data mining processes at different levels of abstraction between the CRISP model as most general and executable workflows as most concrete representation.

Acknowledgement

First, I would like to thank Prof. Dr. Stefan Wrobel for the supervision of my thesis, for lots of helpful suggestions and for pointing into the direction to go. I also thank the members of my thesis committee Prof. Dr. Christa E. Müller, Prof. Dr. Armin B. Cremers and Prof Dr. Oswaldo Trelles Salazar.

Special thanks also go to Stefan Rüping, the leader of the Integrated Data Mining working group, for many fruitful discussions and constant feedback.

This work was partially realized in the context of the projects DataMiningGrid, ACGT and p-medicine, funded by the European Commission. I would like to thank our project partners for getting in touch with the European research community and for the collaboration in the projects.

I thank my colleagues at Fraunhofer IAIS for the friendly atmosphere and interesting discussions. In particular I am grateful to Axel Poigné and Michael Mock for their feedback. Furthermore, I would like to thank my students and student workers who worked under my supervision, and which have contributed to this work.

Finally, I would like to thank my family, especially my wife Kerstin and my daughters Mara and Leni for their understanding and support during the last years.

Contents

List of Figures

List of Tables

1. Introduction

This thesis introduces an approach for the integration of data mining into scientific data analysis processes in the area of bioinformatics. The approach enables users to integrate existing data mining components, to interactively develop new data mining scripts and to reuse data mining processes based on these data mining components and scripts in analysis environments that have to deal with multiple users, distributed data sources and distributed computers.

In the following, we will motivate the need for methods for the integration of data mining into analysis processes in bioinformatics scenarios (Section 1.1) and derive research questions (Section 1.2). Based on that, the contributions (Section 1.3) and the structure of this thesis (Section 1.4) will be summarized. Finally, we will present the publications in which parts of the results of this thesis have been published (Section 1.5). This chapter is based on [145].

1.1. Description of the Problem Area

In the following, we will describe the problem area by introducing data analysis scenarios in bioinformatics and by presenting the main challenges that result from these scenarios. The scenarios result from intensive collaboration with bioinformaticians in European research projects ACGT and p-medicine, which is partially published in [145].

1.1.1. Data Analysis in Bioinformatics

In recent years, using advanced semi-interactive data analysis algorithms such as those from the field of data mining gained increasing importance in areas such as bioinformatics, genetics, medicine, and biodiversity [87, 145]. In the past, data has been mainly collected in the context of a specific problem or study only. Nowadays, data from different sources is also extensively collected in repositories potentially useful for subsequent analysis [87]. Roos states in [106] that "GenBank [...] continues to more than double in size every year. New technologies for assaying gene expression patterns, protein structure, protein-protein interactions, etc., will provide even more data. How to handle these data, make sense of them, and render them accessible to biologists working on a wide variety of problems is the challenge facing bioinformatics". According to Wodehouse [157], "in the field of bioinformatics there has been an increasing movement to develop new methods for the analysis of collected data". At the time the data is collected, the type of analysis that the data later will be used for is not known. In addition, recent advances in technology enable collecting data at more and more detailed levels [106], from organism level, organ level, tissue level up to cellular and even sub-cellular level [87, 157]. As a consequence, a huge amount of non-focused data from different levels of detail is available for analysis.

In this thesis we focus on the field of bioinformatics, as it is prototypical for complex analysis problems because of the complex data, intensive collaboration and huge amount of domain knowledge involved in the analysis scenarios. Bioinformatics is an area in which highly qualified people are dealing with huge amounts and different types of data. Comprehensive metadata describing the semantics of the heterogeneous and complex data are needed to leverage it for further research [152]. To address this issue, efforts exist for describing the data in a comprehensive way by domain specific ontologies [23].

Bioinformaticians and biostatisticians are people who spend their days combining information from different data sources and applying different analysis methods to the information extracted from these repositories. The data sources include data from gene annotation [125], SNPs (single-nucleotide polymorphisms) [27], medical literature [100], public data repositories such as GEO [56], clinical databases, etc. Today, the well known GEO repository contains more than 2700 genomic datasets with over 700.000 samples in total from microarray and high-throughput sequence technologies. PubMed includes more than 21 million citations for biomedical literature. Once the bioinformaticians have found some good combinations of data sources and methods implemented as analysis processes, they want to keep them in stock and slightly alter them later with different inputs or methodologies. This is what we call a *scenario*.

Typical scenarios make use of data mining for answering research questions [109], e.g., finding predictive or prognostic biomarkers, defining subtypes of diseases, classifying samples by using genes, etc.

Roos states in [106] that "computational biology is a fundamentally collaborative discipline". In projects in the bioinformatics area, bioinformaticians are working together with people from IT and with different collaborators like clinicians, biologists, etc. A typical analysis scenario involves multiple users and experts from different departments or organizations.

1.1.2. Challenges

From the discussion and collaboration with bioinformaticians results that today's data analysis scenarios in bioinformatics face the following challenges [145]:

Heterogeneous group of users in different locations: In today's bioinformatics scenarios, users working at different locations have to collaborate. Bioinformaticians of today are from various backgrounds such as data mining, mathematics, statistics, biology, IT development, etc. Thus, the scenarios involve a heterogeneous and distributed group of users. Depending on their background, knowledge, and type of job, users can interact with an analysis environment in a different way and use different tools. E.g., some bioinformatics people might want to configure and run predefined workflows via simple form-based web pages. Other users might want to design new workflows based on existing components or reuse workflows from colleagues. Users might want to develop new components by just writing their analysis algorithms in their own language of choice or use software from colleagues, and might want to integrate them into the system by writing a plug-in module for the code to run within the environment. Advanced users, e.g., might even want to partially modify the structure of the workflow environment. When multiple users work together at different locations and with different background, the set of tools

used is also quite heterogeneous. However, the users typically do not have an overview over the full system and no detailed knowledge about all parts of the system.

Large, heterogeneous and distributed data sources: Today, data is not longer only collected and evaluated with focus on a specific problem or study, but it is more and more reused for further studies in the bioinformatics and healthcare domain. Thus, data from different sources is extensively collected in repositories to allow a subsequent analysis. As not all types of analyses the data will be used for are known at the time the data is collected, content and data format are not focused. Moreover, recent advances in technology allow for collecting data on more detailed levels. Thus, the volume of data of a certain type can become very large. In analysis scenarios in the context of bioinformatics several different data and data types are involved. People with different responsibilities and analysis questions work with different sets of data sources. The corresponding data sources are distributed by nature. There exist a large number of public data sources and repositories that are accessible via the internet. In addition, private data sources are distributed across several departments of a hospital or institute, or even across different hospitals or institutes. As a result, a huge amount of distributed and not-focused data is available for usage. Scenarios involve an increasing number of data sources and amount of data. Typically, bioinformatics scenarios include the development of a solution based on a certain restricted data repository and the evaluation on public available data. The semantic of the datasets is complex and needs to be described to allow a proper usage. Due to the heterogeneity and complexity of the data, several domain specific ontologies exist for the description of the semantics of the data by comprehensive metadata.

Multi computer environments: Today's analysis scenarios have to deal with distributed and heterogeneous users as well as distributed and heterogeneous data sources. Instead of single-computer environments or environments hosted inside a certain organization, the scenarios involve users working with different tools and distributed data sources managed in different systems spread over the globe. In addition, today's data analysis applications in bioinformatics increase in complexity and in their demand for resources. To address this issue, solutions have to integrated into distributed environments such as grid systems that provide secure data access for the different participating organizations and computing resources that allow for scalability.

Complex process chains: Content and data format of the data collected in the area of medicine and bioinformatics are not focused on a certain problem or research question, but they continuously change to the new needs. Thus, complex process chains are needed for the analysis of the data. Building specialized software for each analysis problem that is going to be addressed tends to be the current solution, but this is not ideal as this effort can only be carried out for huge projects running for several years. Thus, such process chains need to be supported by the environments that are used to setup analysis solutions.

1.1.3. Existing Solutions

Today's existing solutions for data mining processes consist of different components that represent different steps in the analysis process. There exist toolkits that allow for combining such components in a graphical or script-based way. Classical data mining tools such as Weka [156], R [103] or RapidMiner [93], which can serve as components in anal-

ysis processes, are based on single computer environments, local data sources and single users. However, analysis scenarios in bioinformatics have to deal with multi computer environments, distributed data sources and multiple users that have to cooperate. Users need support for integrating data mining into analysis processes in the context of such scenarios, which lacks today [145]. The middleware of distributed environments is complex and not easy to use by bioinformaticians. Existing approaches on the integration of existing data mining components in grid environments either demand for programming new services for each data mining algorithm to be integrated [22, 31], for the adoption of the data mining algorithm to fit into a certain environment [63, 54], or are limited to algorithms from a certain data mining toolkit such as Weka [127]. Further details on existing solutions will be presented in Chapters 3 and 4.

In addition, the reuse of existing data mining processes needs to be further supported. The standard data mining process model CRISP [121] is not focussed on reuse. It has been identified that CRISP lacks in the deployment phase [116], in guidance towards implementing particular tasks of data mining methodologies [120], and in the definition of phases important for engineering projects [89]. Reuse at implementation level is also not sufficient, as existing workflows or processes, e.g. provided by repositories such as myExperiment [58], are too specific for being reusable efficiently. In addition, requirements and pre-requisites for the analysis processes are not contained in the process descriptions. Thus, the reuse and integration of existing solutions is not often or only informally done in practice due to a lack of formal support, which leads to a lot of unnecessary repetitive work. Details on this will be presented in Chapter 5.

1.2. Research Questions

Based on the description of the problem area, the basic question that we are going to address in this work is the following: *How can we support users in the bioinformatics and medical domain in data mining based data analysis in the context of heterogeneous settings including heterogeneous user groups, heterogeneous data sources and heterogeneous computing environments?*

This general question can be further broken down to sub-questions. Grid environments are used to address the requirements for secure, distributed and dynamic environments. A lot of data mining components developed for single computer environments already exist, which have to be integrated into such grid environments. However, bioinformaticians typically do not have knowledge in grid based distributed systems. From this, we can derive the first question:

Q1: *What is a suitable integration mechanism that allows users with a lot of domain knowledge but without knowledge on grid systems to integrate data mining components that have been developed in single computer environments into distributed grid-based analysis environments used for bioinformatics scenarios?*

In addition to reusing existing data mining components, bioinformaticians also need to develop new data mining components and to compose these components to address more complex scenarios. A frequently used possibility for the development of new and for the composition of existing data mining components are data mining scripts. Instead of

developing these scripts in a single computer environment and integrating them afterwards, the users need to develop them inside the distributed analysis environments to allow for combining information from different data sources and applying different methodologies to the information extracted from these repositories. This leads to the next question:

Q2: *Is it possible to create a technical system that allows scientific users to interactively develop data mining scripts consisting of one or more data mining components for bioinformatics scenarios in distributed grid-based analysis environments?*

In today's analysis scenarios in bioinformatics, complex process chains have to be set-up. These process chains can consist of several data mining components and scripts. The composition of such process chains is a huge effort. Thus, the reuse of existing processes becomes much more important. The question to be asked is:

Q3: *Can we define a description for data mining processes that allows for the reuse of existing data mining processes based on data mining components and scripts?*

The goal of this work is to give answers to the questions Q1 to Q3. An overview of the dissertation's contributions is outlined in the next section.

1.3. Contributions

In this thesis we will introduce an approach for addressing the presented research questions based on infrastructure principles that allow building on data mining components that are already available, developing new data mining scripts, and reusing existing data mining based analysis processes, so that users can keep working with their well-known tools. In the following, the main contributions of this thesis will be summarized.

1.3.1. Integration of Existing Data Mining Components

To support users in reusing standard data mining components with small effort there is a need for an approach to integrate data mining components being developed for a single-computer environment into grid environments. This work contributes an approach for the integration of data mining components into grid environments based on a metadata schema definition. The schema, which we call Application Description Schema (ADS), is used to grid-enable existing data mining components that can be used as atomic components, for example as tasks in workflows. By describing an existing data mining component with the ADS, the effort needed for deploying and executing the data mining component in a grid environment can be reduced. In detail, the ADS is used to manage user interaction with services of grid systems in order to grid-enable existing data mining components, to register and search for available data mining components in the grid, to match analysis jobs with suitable computational resources, and to dynamically create user interfaces. The approach allows for an integration by users without deeper knowledge on the underlying system and without any intervention on the data mining component side, and thus addresses the needs of the community to support users in (re)using standard data mining components.

1.3.2. Interactive Development of Data Mining Scripts

In addition to reusing components from single-computer environments, users like bioinformaticians and biostatisticians typically need to develop new data mining components in the analysis environment interactively to allow for combining information from different data sources and for applying different methodologies to the information extracted from these repositories. Furthermore, users need to compose data mining components to address complex scenarios. The development of atomic components in a local environment and frequent integration would result in inefficiencies in the development life-cycle due to complex debugging procedures and repeated integration effort. In addition, the development might rely on data or computing resources which are not accessible in the local environment. Data mining scripts allow for the development of data mining components and their composition in a way that they do not have to be treated as atomic components. In addition, atomic components can still be used within data mining scripts. This work contributes an approach for interactive development of data mining scripts in the context of bioinformatics scenarios in grid environments. It allows for developing and executing data mining scripts without the need for developing, describing and deploying atomic components. Our approach for interactively developing data mining scripts in grid environments is implemented in the GridR toolkit. The underlying method of GridR reduces the complexity of integrating and executing analysis scripts in such environments. Instead of registering each single script separately in the environment, the method is technically based on a single grid service with complex inputs and outputs that allows for providing the script as parameter. In addition, the method allows for interactively developing data mining scripts in grid environments. The approach efficiently supports users when it is necessary to enhance available or to develop new data mining scripts.

1.3.3. Data Mining Process Patterns

In today's analysis solutions in bioinformatics, complex process chains have to be setup. The composition of such process chains, which can include several atomic data mining components or more complex data mining scripts, is a huge effort. Thus, the reuse of existing processes becomes much more important. However, analysis processes often cannot be reused directly, as they are customized to a certain analysis question and the information on how the process was set-up and which requirements have to be met for applying the process is often not available. Thus, processes in a deployable form, e.g. executable workflows, are not suitable for efficient reuse, as they are often too specific and too detailed. Abstract process descriptions, e.g. based on CRISP, are exchangeable but need too much effort for developing a deployable solution. This thesis contributes the concept of Data Mining Process Patterns to describe processes in a form that better supports the reuse of existing processes. Data Mining Process Patterns allow for facilitating the integration and reuse of data mining in analysis processes by describing the steps of a process at different levels of abstraction. The description is based on a task hierarchy that allows for generalizing and concreting tasks for enabling their reuse and includes the encoding of requirements and pre-requisites inside the analysis process. The Data Mining Process Patterns support the description of data mining processes at different levels of abstraction

between the CRISP model as most general and executable workflows as most concrete representation. Hence, they allow for easy reuse and integration of data mining processes. Our approach supports users in scenarios that cover different steps of the data mining process or involve several analysis steps.

1.4. Structure of this Dissertation

The remainder of this thesis is structured as follows: Chapter 2 introduces the field of scientific data analysis in the area of bioinformatics and presents background information on data mining and analysis environments. In Chapter 3 we introduce our approach to integrate data mining components being developed for single computer environments into grid environments. Chapter 4 presents our approach for interactively developing data mining scripts in grid environments. In Chapter 5, we present our approach on facilitating the reuse of existing data mining processes based on Data Mining Process Patterns. Chapter 6 concludes.

1.5. Publications

The main contributions of this thesis have been published by the author in the following publications:

Journals and Magazines:

- Wegener, Dennis and Sengstag, Thierry and Sfakianakis, Stelios and Rüping, Stefan and Assi, Anthony. *GridR: An R-based tool for scientific data analysis in grid environments.* Future Generation Computer Systems 25 (4), pp. 481-488, 2009.
- Stankovski, Vlado and Swain, Martin and Kravtsov, Valentin and Niessen, Thomas and Wegener, Dennis and Röhm, Matthias and Trnkoczy, Jernej and May, Michael and Franke, Jürgen and Schuster, Assaf and Dubitzky, Werner. *Digging Deep into the Data Mine with DataMiningGrid.* IEEE Internet Computing, vol. 12, no. 6, pp. 69-76, 2008.
- Stankovski, Vlado and Swain, Martin and Kravtsov, Valentin and Niessen, Thomas and Wegener, Dennis and Kindermann, Jörg and Dubitzky, Werner. *Grid-enabling data mining applications with DataMiningGrid: An architectural perspective.* Future Generation Computer Systems 24 (4), pp. 259-279, 2008.
- Wegener, Dennis and Rüping, Stefan. *Integration and reuse of data mining in business processes - a pattern-based approach.* Int. J. Business Process Integration and Management, vol. 5 (3), pp. 218-228, 2011.
- Rossi, Simona and Christ-Neumann, Marie-Luise and Rüping, Stefan and Buffa, Francesca and Wegener, Dennis and McVie, Gordon and Coveney, Peter and Graf, Norbert and Delorenzi, Mauro. *p-Medicine: From data sharing and integration via VPH models to personalized medicine.* ecancermedicalscience 5(218), 2011.

Book Chapters:

- Wegener, Dennis and May, Michael. *Specification of distributed data mining workflows with DataMiningGrid.* In: Data Mining Techniques in Grid Computing Environments, W. Dubitzky (Editor), John Wiley & Sons, 2008.

Conferences:

- Wegener, Dennis and Sengstag, Thierry and Sfakianakis, Stelios and Rüping, Stefan and Assi, Anthony. *GridR: An R-based grid-enabled tool for data analysis in ACGT clinico-genomic trials.* In: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (eScience 2007), pp. 228-235, Bangalore, India, 2007.

- Rüping, Stefan and Wegener, Dennis and Sfakianakis, Stelios and Sengstag, Thierry. *Workflows for Intelligent Monitoring Using Proxy Services.* In: Healthgrid Research, Innovation and Business Case - Proceedings of HealthGrid 2009, Solomonides, Tony, and Hofmann-Apitius, Martin, and Freudigmann, Mathias, and Semler, Sebastian Claudius, and Legre, Yannick, and Kratz, Mary (eds.), IOS Press, pp. 277-282, 2009.

- Wegener, Dennis and Rüping, Stefan. *On Integrating Data Mining into Business Processes.* Proceedings of the 13th International Conference on Business Information Systems (BIS 2010), Witold Abramowicz, Robert Tolksdorf (eds.), Lecture Notes in Business Information Processing, vol. 47, pp. 183-194, Springer Berlin Heidelberg, 2010.

Workshops:

- Wegener, Dennis and May, Michael. *Extensibility of Grid-Enabled Data Mining Platforms: A Case Study.* In Proceedings of the 5th International Workshop on Data Mining Standards, Services and Platforms, San Jose, California, USA, pp. 13-22, 2007.

- Wegener, Dennis and Hecker, Dirk and Körner, Christine and May, Michael and Mock, Michael. *Parallelization of R-programs with GridR in a GPS-trajectory mining application.* In Proceedings of the ECML/PKDD 2008 First Ubiquitous Knowledge Discovery Workshop (UKD08), Antwerp, Belgium, September, 2008.

- Wegener, Dennis and Sengstag, Thierry and Sfakianakis, Stelios and Rüping, Stefan. *Supporting parallel R code in clinical trials: a grid-based approach.* In Proceedings of the 4th IEEE ISPA 2008 Workshop on High Performance and Grid Computing in Medicine and Biology (HiPGCoMB08), Sydney, Australia, December, 2008.

- Wegener, Dennis and Mock, Michael and Adranale, Deyaa and Wrobel, Stefan. *Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters.* In: Proceedings of the IEEE International Conference on Data Mining Workshops, pp. 296-301, Miami, USA, 2009.

- Sfakianakis, Stelios and Graf, Norbert and Hoppe, Alexander and Rüping, Stefan and Wegener, Dennis and Koumakis, Lefteris. *Building a System for Advancing Clinico-Genomic Trials on Cancer.* In: Bassiliades, N. (ed.), Proceedings of the Biomedical Informatics and Intelligent Methods in the Support of Genomic Medicine (BIMIINT) Workshop of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI-2009), pp. 36-47, Thessaloniki, Greece, April, 2009.

- Rüping, Stefan and Wegener, Dennis and Bremer, Philipp. *Re-using Data Mining Workflows.* In: Proceedings of the ECML PKDD 2010 Workshop on Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD'10), pp. 25-30, Barcelona, Spain, 2010.

- Wegener, Dennis and Rüping, Stefan. *On Reusing Data Mining in Business Processes - A Pattern-Based Approach.* In: Business Process Management Workshops, LNBIP 66, pp. 264-276, Springer, Heidelberg, 2011.

- Bucur, Anca and Rüping, Stefan and Sengstag, Thierry and Sfakianakis, Stelios and Tsiknakis, Manolis and Wegener, Dennis. *The ACGT project in retrospect: Lessons learned and future outlook*, Procedia Computer Science, Volume 4, Proceedings of the International Conference on Computational Science (ICCS 2011), pp. 1119-1128, 2011.

- Wegener, Dennis and Anguita, Alberto and Rüping, Stefan. *Enabling the reuse of data mining processes in healthcare by integrating data semantics.* Proceedings of the 3rd International Workshop on Knowledge Representation for Health Care (KR4HC 2011), pp. 222-235, 2011.

- Wegener, Dennis and Rossi, Simona and Buffa, Francesca and Delorenzi, Mauro and Rüping, Stefan. *Towards an Environment for Data Mining based Analysis Processes in Bioinformatics and Personalized Medicine.* BIBM Workshops 2011, pp. 570-577, 2011.

2. Scientific Data Analysis, Data Mining and Data Analysis Environments

The goal of this chapter is to introduce preliminaries and state-of-the-art from the field of data mining based scientific data analysis that are necessary for comprehending the work presented in this thesis and to discuss related prior work. First, Section 2.1 introduces data mining and the standard data mining process model CRISP. In addition, we describe how data mining is used in practice and we give some basic definitions. Second, Section 2.2 presents background information on data mining in the context of larger systems. This includes the area of distributed data mining, grid computing, process modelling and workflow environments. Third, we describe bioinformatics scenarios in more detail in Section 2.3. Fourth, the European research projects DataMiningGrid, ACGT and p-medicine are introduced in Section 2.4. These projects provide grid- and workflow-based scientific data analysis environments in which the contributions of this thesis are partially implemented. Finally, Section 2.5 wraps up.

2.1. Data Mining

In this section, we will introduce data mining and the standard data mining process CRISP [121]. In addition, we will describe how data mining is used in practice and present basic definitions.

2.1.1. Data Mining Problems, Goals and Methods

Fayyad [45] describes data mining as follows: Data mining, also called knowledge discovery in databases (KDD), is the process of extracting (unknown) patterns from data involving methods from artificial intelligence, machine learning and statistics. In general, a data mining process includes several iterations of single data mining steps (algorithm executions). The goals of the data mining process are defined by the intended use of the system from the user perspective and can be classified into two types: *verification*, where the system is limited to verifying the user's hypothesis, and *discovery*, where the system autonomously finds new patterns. The discovery goal can be further subdivided into *prediction*, where the system finds patterns for predicting the future behaviour of some entities, and *description*, where the system finds patterns for presentation to a user in a human-understandable form.

A variety of data mining methods exists which help in achieving the goal of prediction and description. For each of these methods a variety of data mining algorithms exist that incorporate these methods. Understanding the details of certain data mining algorithms

Outlook	**Temperature**	**Humidity**	**Windy**	**Play**
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Table 2.1.: The weather data from [156].

is not necessary for comprehending the thesis. Thus, we refer to [156] for an updated reference on data mining methods and algorithms.

As example for a data mining scenario we can have a look at the tiny weather dataset from [156] that is used to illustrate data mining methods. It concerns the conditions that are suitable for playing some unspecified game. Instances in the dataset are characterized by the values of the attributes outlook, temperature, humidity and windy that measure different aspects of the instance. The outcome (also called class attribute) is to whether play or not. Examples of instances with nominal values can be seen in Table 2.1. A set of classification rules that can be learned from this data could look as follows:

If outlook = sunny and humidity = high	then play = no
If outlook = rainy and windy = true	then play = no
If outlook = overcast	then play = yes
If humidity = normal	then play = yes
If none of the above	then play = yes

The rules are meant to be interpreted in order. A decision tree for the weather data can be seen in Figure 2.1. The rules as well as the decision tree represent a description of the dataset in a human understandable form. In addition, these can be used to predict future behaviour. E.g., a new data instance could look like "Outlook=sunny, Temperature=hot, Humidity=normal, Windy=false". The prediction for this instance would be "Play=yes".

A slightly more complex example with some numeric and some missing values can be seen in Table 2.2. As some methods only work on nominal values, some only on numerical values, and some have problems in dealing with missing values, such data usually needs to be preprocessed before it can be processed by a certain data mining method. The preprocessing could, e.g., include the discretization of numerical values or the replacement of missing values.

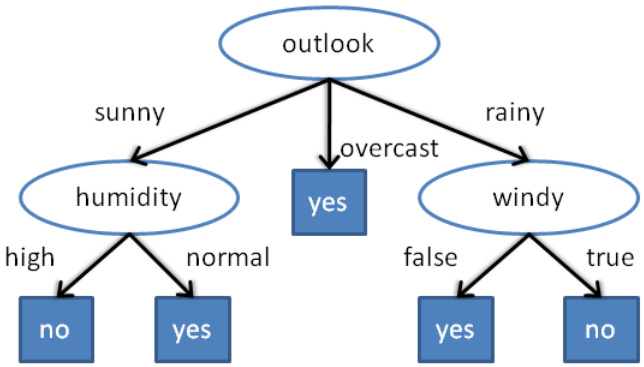

Figure 2.1.: A decision tree for the weather data from [156].

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	?	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	?	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

Table 2.2.: The weather data from [156] with some numeric attributes.

2.1.2. The Data Mining Process

A data mining process is an interactive and iterative process that involves numerous steps with many decisions made by the data miner. The Cross Industry Standard Process for Data Mining (CRISP) is a standard process model for data mining that depicts corresponding phases of a project, their respective tasks, and relationships between these tasks [121]. According to CRISP, the life-cycle of a data mining project consists of the following six different phases (see Figure 2.2):

- *Business Understanding* - understanding the project objectives and requirements from a business perspective and converting this knowledge into a data mining problem definition; Looking at the weather example, from the business perspective the goal is to get information on whether to play or not to play the game. From the data mining perspective, the goal is to describe the given data by a model which can

also be used to make a prediction for new data instances.

- *Data Understanding* - getting to know the data and to identify data quality problems; In the example, this means to look at the available data and create a data table.
- *Data Preparation* - construct the final dataset from the initial raw data as input for the modelling; In the example, the data has to be pre-processed, e.g. by discretization of numerical values and the replacement of missing values.
- *Modeling* - various modelling techniques are selected and applied, including the calibration of their specific settings; In the example, this means to create a set of rules or to create a decision tree.
- *Evaluation* - assess how well the built model achieves the business objectives; In the example, it has to be checked if the model is useful for deciding whether to play or not to play the game.
- *Deployment* - the results of the data mining and the knowledge gained are delivered to the user, reaching from generating a simple report up to a complex implementation of a repeatable data mining process; In the example, this means to report the model in human understandable form and information about its quality to the end user.

When developing the solution for a given data mining problem, the abstract CRISP process is instantiated by concreting the abstract tasks of the phases of the CRISP process model. The process in general is iterative, but also foresees stepping back between certain phases to adjust some of the decisions made. From the data mining perspective, the user is mainly involved in the phases Business Understanding and Deployment, while the other phases are mostly performed only by the data mining expert. It has to be distinguished between the (technical) deployment of the data mining solution as a whole, which might be done only once for a given business process, and the deployment of new data mining models, which might be done frequently.

At the end of the modelling step of the data mining process, the data mining model is evaluated in order to determine its quality. Model-evaluation criteria are quantitative statements of how well a particular data mining model meets the goals of the KDD process [45] and the business objectives [70]. The criteria differ according to the data mining goal. For each of the goals verification, prediction and description a number of performance measures exist for the evaluation. In addition, techniques like cross-validation or bootstrap [156] are applied in order to ensure the statistical validity of the evaluation. Despite these metrics for evaluating the result of the data mining process, other important factors like the time spent for the process, the resources that were used, etc., can be included in the evaluation. In the evaluation phase of the data mining process, the model as well as the way it was constructed is evaluated according to the business objectives.

2.1.3. Data Mining in Practice

Today, there exist a variety of commercial and open source data mining toolkits developed for single computer environments. Commercial tools include SAS Enterprise Miner [77],

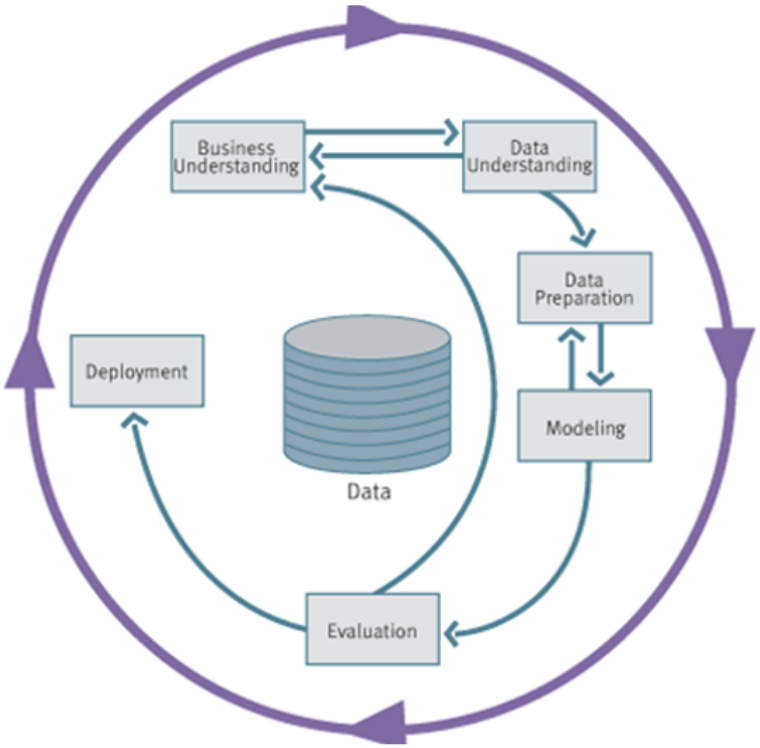

Figure 2.2.: The CRISP-DM process model for data mining.

IBM SPSS [73], Oracle Data Mining [96], Microsoft SQLServer Data Mining [92], and many others. In addition, there exist lots of common open source data analysis tools such as Weka [156], RapidMiner [93], R [103], and many more. All these tools can be utilized for data mining processes.

Typically, data mining processes instantiated from the CRISP process model include common steps or tasks that are independent from the application scenario and thus can be seen as standardizable. Thus, these steps and tasks are addressed by functionality of data mining toolkits. This includes the following:

- **Chaining**: A typical data mining process spans a series of steps from data preparation to analysis and visualization that have to be executed in sequence. Chaining is the concatenation of the execution of different algorithms in a serial manner. The result of an algorithm execution is used by the next execution in the series.

- **Looping**: Looping means the repeated execution of an algorithm. The algorithm can run on the same (or different) data or with the same (or different) parameter settings until a condition is fulfilled.

- **Branching**: Branching means to let a program flow follow several parallel executions paths from a specified point or to follow a special execution path depending on a condition.

- **Parameter Variation and Optimization**: Parameter variation means to execute the same algorithm with different input data and different parameter settings.

- **Cross-validation and optimization**: The cross-validation technique is used to evaluate the results of the data mining process. These results are often optimized according to the results of the validation, which is a very time-consuming step. In detail, cross-validation involves partitioning a dataset into complementary subsets, executing the data mining algorithm on one subset (the training data), and validating the result on the other subset (the test data). To reduce variability, the validation is performed multiple times using different partitions. The validation results are averaged or combined afterwards. E.g., in k-fold cross-validation, the original dataset is randomly partitioned into k subsets. A single subset from the k subsets is used as the test data for testing the data mining model, and the remaining $k-1$ subsets are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsets used exactly once as the test data.

- **Data Partitioning**: Data partitioning means to cut the data into different pieces and execute the data mining algorithm on the different subsets of the data.

While setting up a solution for a data mining problem, users can try to reuse an existing implementation of an algorithm. The choice of an available algorithm depends on the statistical solution, the general plan aims, the type and the amount of data that needs to be analysed. Sometimes, a step of the solution needs to be customized in order to match the objectives. This means that the implementation of an existing algorithm needs to be adapted or enhanced and tested. In some cases, existing solutions are not available to pursue the project aims, so new algorithms need to be developed and implemented from scratch.

There exist process environments that also enable the usage of data mining, e.g., the workflow environments Taverna [72], Triana [128], Kepler [86] or Galaxy [61] in the area of bioinformatics, and the business process management systems Activiti [6], Aris [7], Intalio BPMS [76], jBPM [79] or YAWL [129] in the area of business processes.

In the following, we will describe Weka and R in more details. These are the underlying data mining tools that are integrated into grid environments based on the contributions of the Chapters 3 and 4. Information on workflow environments will be later given in Section 2.2.3.

Weka

The Waikato Environment for Knowledge Analysis (Weka) is a popular and freely available data mining toolkit [156]. Weka contains many well-known data mining algorithms and can be used via GUI (see Figure 2.3), via command line or as library. In detail, the Weka machine learning workbench provides a general-purpose environment for classification, regression, clustering and feature selection, which are also common data mining problems in bioinformatics research [52]. In addition to the extensive collection of machine learning algorithms and data pre-processing methods, Weka contains graphical user interfaces for

data exploration as well as to evaluate and compare different machine learning techniques. Weka can be also used via command line in the following way:

```
java "classname" "parameters"
```

E.g., the following command line call starts a decision tree algorithm with the weather dataset:

```
java weka.classifiers.trees.J48 -t data/weather.arff
```

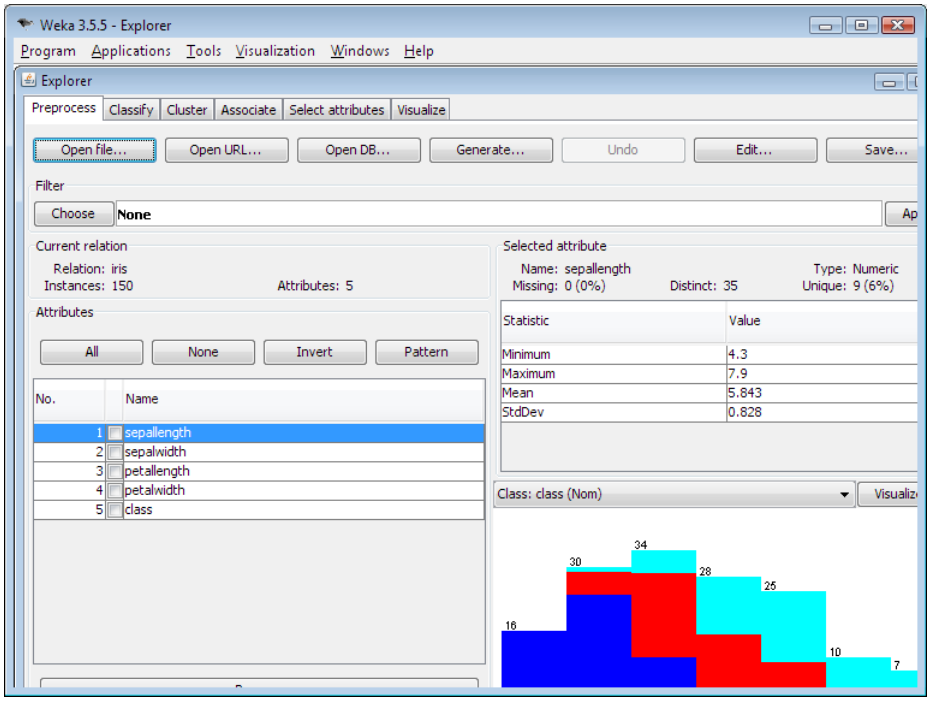

Figure 2.3.: Screenshot of the Weka Explorer Version 3.5.5 (from wikipedia.org).

The R-Project

R [103] is a system for statistical computing and graphics published under the GNU General Public License. It consists of a programming language and a run-time environment and is being developed for the Unix-like, Windows and Mac families of operating systems [71]. The R language allows branching and looping as well as modular programming using functions. In addition, the environment supports running commands interactively in a console or as batch programs stored in script files.

The R environment is extensible via packages. Additional modules are available for a variety of specific purposes. The R environment provides a broad range of state of the art statistical, graphical techniques and advanced data mining methods (including comprehensive packages for linear and non-linear modelling, cluster analysis, prediction,

hypothesis tests, re-sampling, survival analysis, time-series analysis, etc.), and turned out as the de facto standard for statistical research in many applied statistics project.

In the following, we will present some details on R based on [103]. Technically, R is an expression language with a simple syntax. When starting the R program (start of a session) it issues a prompt ">" when it expects input commands. Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed, and the value is lost. An assignment also evaluates an expression, but passes the value to a variable. The result is not automatically printed. During an R session, objects are created and stored by name. The collection of objects currently stored is called the workspace. The workspace can be stored into and be restored from a file. A session is terminated with the "$q()$" command.

Commands can be stored in external files, e.g. *"script.R"*. They can be executed at any time in an R session with the command

```
> source("script.R")
```

R packages can be loaded by

```
> library("packageName")
```

An R function is defined by an assignment of the form

```
> name <- function(arg_1, arg_2, ...) expression
```

The expression is an R expression that uses the arguments ("arg_i") to calculate a value. The value of the expression is the return value of the function. A call to the function looks as follows:

```
name(expr_1, expr_2, ...)
```

Functions can depend on objects stored in the workspace. In addition, R supports scripting. You can run a file of R commands ("*script.R*") by executing

```
R CMD BATCH script.R
```

Parameters can be passed to scripts via additional arguments on the command line:

```
R CMD BATCH --args arg1 arg2 script.R &
```

The R script is then executed as a whole in it's own session. R output is written to *stdout* and *stderr* and can be collected in a file, e.g. *script.Rout* for a script named *script.R*.

2.1.4. Basic Definitions

In the following, we will present the definition of data mining services, data mining components, data mining scripts and data mining workflows.

Data mining components and data mining services encapsulate a certain functionality and are (re)usable via (user) interfaces. Data mining services are based on web services. A web service is a software system designed to support interoperable machine-to-machine interaction over a network [118]. It is deployed somewhere in the network and its interface is described in a machine-readable format.

Definition 1 ***(Data Mining Service)*** *A* ***data mining service*** *consists of the implementation of a data mining algorithm which is executable via a webservice interface. A data mining service is considered as atomic, as usually only the interface of the service is published and all details on the implementation are hidden.*

Data mining components, in contrast to services, share the characteristic that the program that implements the algorithm is locally available. In this work, we consider a data mining component to support at least a command line interface. Data mining components have two groups of inputs. The first group of inputs is data. The data typically shares the characteristics of being large and not easily movable. The second group of inputs is a number of values defining how the component's algorithm processes the data. The algorithms often create a model that describes the data in a certain way. The quality of the model is determined by different performance values. The output of the component is some information on the data.

Definition 2 ***(Data Mining Component)*** *A* ***data mining component*** *consists of the implementation of a data mining algorithm which is executable on a single computer that provides a runtime environment for the component and which is used via command line. We consider a component to be atomic, which means that is cannot easily be split and has to be used as a whole either stand-alone, in a data mining script or in a workflow.*

An example for a data mining component is the Weka jar file from the Weka toolkit, which is executable on computers where the java environment is installed. It can be considered as atomic, as it is not directly possible to look into and change the java code. In fact, the Weka jar file represents a set of components, as it contains implementations of several algorithms. Components can also be developed and composed in scripting languages.

Definition 3 ***(Data Mining Script)*** *A* ***data mining script*** *consists of one or more data mining components and is based on a language that allows for scripting, which means that it's code is interpreted, but not compiled.*

A script can include several existing (atomic) components. In addition, a script can be used to develop new non-atomic components. Furthermore, the script itself can be used as a whole in a workflow. An example for a data mining script is an R script, which can be executed on computers where the R environment is installed. By the *library* command, existing components are made available. New components can be developed by defining new functions. Components can be connected inside a script.

Scripts and components can also be composed by using workflows. A workflow is basically a description of the order in which a set of services or components have to be called with a certain input in order to solve a given task. Workflows are defined at a higher level of abstraction than programming or scripting languages. They are more structured and are limited in their expressiveness, but are easier to compose and to validate.

Definition 4 ***(Data Mining Workflow)*** *A* ***workflow*** *consists of a set of technical tasks that are connected to each other to achieve a goal. A* ***data mining workflow*** *addresses one of the data mining goals described in Section 2.1. The tasks of data mining*

workflows can consist of data mining components, scripts or other services. The workflow can be specified in a workflow description language and is executable in a workflow environment. On certain datasets the workflow is able to produce output that is related to one of the data mining goals.

An example of a data mining workflow designed with the Weka knowledge Flow tool is visualized in Figure 2.4. The workflow consists of loading a data file (task *ArffLoader*), specifying the class attribute (task *ClassAssigner*), a cross validation component (task *CrossValidationFoldMaker*), a decision tree component that works on the training and test data provided by the cross validation task (task *J48*), the evaluation of the performance values (task *ClassifierPerformanceEvaluator*), and the visualization of the performance values (task *TextViewer*). A data mining workflow can be (part of) a solution for a bioinformatics or analysis scenario.

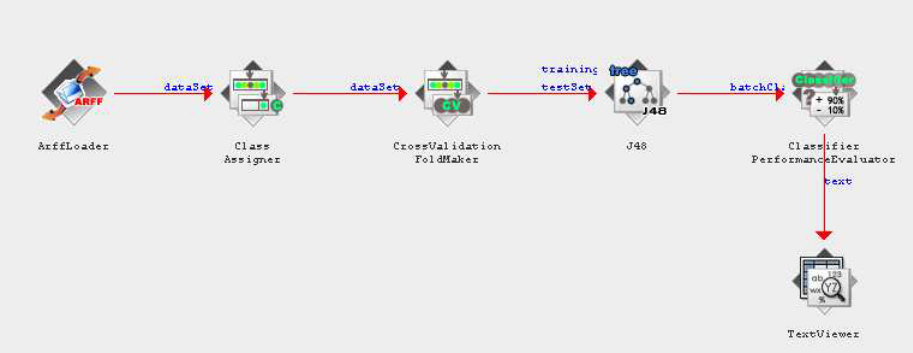

Figure 2.4.: Screenshot of a Weka workflow (from [65]).

2.2. Distributed Computing and Data Mining Systems

In this section, we will introduce the field of distributed data mining and the field of grid computing, as this type of distributed system meets the requirements of the bioinformatics scenarios presented in Section 2.3. In addition, we give details on process modelling and present selected workflow and process environments that are important for the work presented in this thesis.

2.2.1. Distributed Data Mining

Distributed computing is a field of computer science that studies distributed systems [14]. A distributed system consists of multiple computers (or machines) that interact with each other via a network to achieve a common goal. A computer program that runs in a distributed system is called a distributed program. Distributed computing also refers to solving computational problems by the use of distributed systems. Typically, a problem is divided into a set of smaller independent tasks, each of which is solved by one or more computers. Various hardware and software architectures are used for distributed

computing and there exist several types of distributed systems, e.g. computing clusters, grid systems, computing clouds, etc.

The field of distributed data mining deals with the problem of analysing and monitoring distributed data sources by data mining technology designed for distributed systems. There exists a huge amount of distributed data mining algorithms (we refer to [18] for an updated reference).

For determining the performance of a distributed data mining task, which means to find out if the task is executed in an efficient way, there exist several measures on the resources used to solve the task. In addition to time and (disk or memory) space, in the area of distributed computing the number of computers is another resource to consider. Often, there is a trade-off between the running time of a task and the number of computers: the problem can be solved faster if sub-problems can be solved on several computers in parallel. In distributed computing, speedup refers to how much a task is faster due to parallel processing of sub-tasks than a corresponding sequential task. We define speedup as $S_p = \frac{T_1}{T_p}$ where p is the number of processors, T_1 is the execution time of the sequential task, and T_p is the execution time of the parallel sub-tasks with p processors.

Linear or ideal speedup is obtained when $S_p = p$. When running an algorithm with linear speedup, doubling the number of processors doubles the speed. As this is ideal, it is considered very good scalability.

However, not all parallelization effort for distributed computing results in speed-up. If a task is split into subtasks, these subtasks, the data and the results might have to be transferred over the network. In addition, intermediate results might have to be exchanged. Thus, there is an overhead for communication and data transfer involved which donates to the time spent for solving the task.

Scalability is the ability of a system to handle growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth [20].

There exist several attempts to adapt common data mining tools to multi computer environments. Among these, solutions for distributed data mining with Weka [8, 127, 28, 82, 98] and with R [85, 160, 88, 111, 16, 135, 62] have been developed. Further details will be given in the related work sections of Chapters 3 and 4.

2.2.2. Grid Computing

Grid technology was developed to address requirements for secure, distributed and dynamic environments. It provides solutions for issues like data security, resource sharing, resource brokering and standardization and supports the usage of multi-computer environments, distributed data sources and multiple users. Forming virtual organizations in a grid allows people to collaborate in a secure environment. Furthermore, mechanisms for reliable file transfer and the sharing of computational and storage resources are provided.

According to [47] a grid is a system that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver non-trivial qualities of service.

Grid technology evolves rapidly and this often poses challenges, such as interoperability problems, when building applications on open source technology as the available functionality frequently changes. In the past, grid standardisation efforts have concentrated on

achieving the goal to define an open framework for modelling and accessing stateful resources by using Web services. The actual OASIS standard named Web Services Resource Framework (WSRF) v. 1.2 was approved in April 2006 [3].

In the following, we will introduce the OGSA standard [46] and the Globus Toolkit [48]. The approaches which will be presented in the Chapters 3 and 4 are based on OGSA and are evaluated in grid systems based on Globus toolkit.

Open Grid Services Architecture (OGSA)

The Open Grid Services Architecture (OGSA) [49, 46], being sustained by the Global Grid Forum (GGF), describes a reference architecture for a service-oriented grid computing environment. This architecture addresses the need for standardization by defining a set of core capabilities and behaviours for grid systems.

OGSA provides a logical 3-tiered view of distributed environments realized by the use of grid systems [36] (see Figure 2.5). The bottom layer (fabric) includes different types of base resources that are virtualized through web-services. These base resources are supported by some underlying entities or artifacts that may be physical or logical, and that have relevance outside of the OGSA context. Examples of physical entities include CPUs, memory, and disks. Examples of logical artifacts include licenses, contents, and OS processes [46].

The middle layer (middleware) represents a higher level of virtualization and logical abstraction. This layer is defining a wide variety of capabilities that are relevant to OGSA grids. These capabilities are provided by services, which build the *grid middleware*. In detail, the following capabilities (or categories of services) are defined to develop grids [36]: Infrastructure Services, Data Services, Resource Management Services, Execution Management Services, Security Services, Self Management Services, Optimization Services, and Information Services.

The top layer (applications) includes applications and other entities that use the OGSA capabilities to realize user- and domain-oriented functions and processes. These functions and processes make use of the grid middleware to undertake their activities.

Definition 5 ***(Grid middleware) Grid middleware*** *is a collection of services according to the middleware layer of OGSA implemented as grid services.*

Definition 6 ***(Grid based system)*** *A* ***grid based system*** *is a distributed system that is designed and implemented based on the grid reference architecture OGSA.*

The following list describes the capabilities and services that are important in the context of this thesis in more details:

- **Information Services.** Grid environments are typically not designed for a certain application context. The components to be run in the system can differ from each other and can change over time dynamically. Thus, grid environments provide a registry of currently available services and components that can be used within the grid environment. New services and components have to be registered to become part of the grid environment.

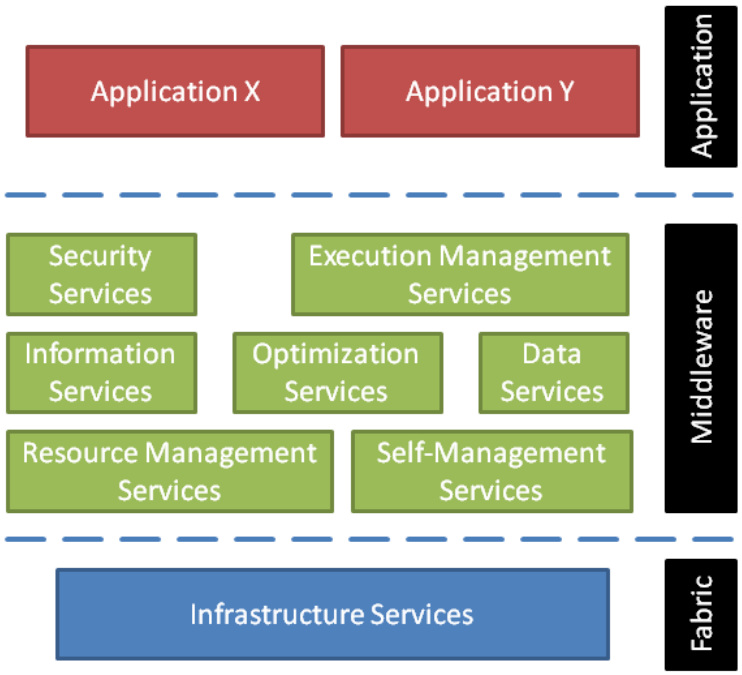

Figure 2.5.: Open Grid Service Architecture (based on [36]).

- **Data Services.** Grid environments provide data services for secure and reliable data transfer.

- **Application Services & User Interfaces.** The components to be run on the grid need to be configured and started by the users. Thus, grid environments typically include user interfaces for end users which are often workflow based. Services and components can be configured and parametrised by these user interfaces before execution.

- **Resource and Execution Management Services.** Computing and storage resources in the grid environment can change over time dynamically. Thus, grid environments must provide manageability for execution of user-defined work (jobs) throughout their lifetime. Functions such as scheduling, job control and exception handling of jobs must be supported when the job is distributed over a great number of heterogeneous resources [46]. Job descriptions are used to describe the execution specific requirements of a job that is to be executed on the grid in order to map submitted jobs onto available hard- and software resources in the grid environment.

- **Self-Management Services.** Users need to track and analyse the executions of their components and the resources used. Grid environments include monitoring and analysis functionality that allow for the monitoring and analysis of the jobs that are executed in the grid environment, the usage of resources and the occurrence of errors and faults.

Globus Toolkit

The Globus Toolkit [48] is an open source software tool-kit used for building grids developed by the Globus Alliance. Version 4 of the tool-kit (GT4), which forms the basis for some of the implementations presented in this thesis, represents a web service based realization of a grid. The Globus Toolkit is an implementation of the standards Open Grid Services Architecture (OGSA) and Web Services Resource Framework (WSRF). In addition, the Globus Toolkit includes service implementations to provide:

- Resource management: Grid Resource Allocation & Management Protocol (GRAM)
- Information Services: Monitoring and Discovery Service (MDS)
- Security Services: Grid Security Infrastructure (GSI)
- Data Movement and Management: Global Access to Secondary Storage (GASS) and GridFTP

The Globus Toolkit can be considered as a standard reference for a state-of-the-art grid middleware. GT4 can be connected to computing clusters, e.g. to the Condor batch processing system [130]. Condor is a cluster management system for computing-intensive jobs, providing mechanisms for queuing, scheduling and resource management. Jobs can be submitted to the Condor system as independent tasks. Condor places them in a queue and decides upon a policy when and where to run the tasks.

2.2.3. Process Modelling and Workflow Environments

Workflow environments including workflow designers and enacting systems are a popular technology in business and e-science alike to flexibly define and enact complex data processing tasks (see also Section 2.1.3). Driven by specific applications, a large collection of scientific workflow systems have been prototyped in the past, e.g. Taverna [72], Triana [128], Kepler [86] or Galaxy [61]. There also exist distributed workflow management systems that are integrated with grid computing environments, such as SWIMS [41]. The next generation of workflow systems are marked by workflow repositories such as myExperiment [58] or workflow sharing functionality, which tackle the problem of organizing workflows by offering the research community the possibility to publish, exchange and discuss individual workflows. In the area of business process management (BPM), there also exist a lot of environments for managing business processes and workflows, e.g. commercial and open source systems such as Activiti [6], Aris [7], Intalio BPMS [76], jBPM [79], YAWL [129], and many more.

A conceptual basis for process and workflow technology is provided by the concept of workflow patterns [117]. Such workflow patterns represent simple patterns for modelling certain structures in processes and workflows. According to [15], process patterns have many advantages: processes in BPM systems serve as both the specification and the source code. The modelled processes become the solutions deployed and provide a simple communication tool between end-users, business analysts, developers and the management.

In the context of the process and workflow systems mentioned above, several languages and notations for describing, modelling and visualizing processes emerged, e.g. XPDL [33], BPEL [4], YAWL [129], BPMN [153], and many others.

In the following, we will describe success factors for process modelling, the modelling notation BPMN which will be used for visualising process in this thesis, and the Triana system as prototypical workflow environment.

Success Factors for Process Modelling

Modelling processes and designing executable workflows usually is done to replace manual, semi-automatic or inefficient processes that have been used before. According to Hammer and Champy [66], process reengineering is "the fundamental rethinking and radical redesign of (business) processes to achieve dramatic improvements in critical, contemporary measures of performance". There exist several metrics from the area of business process reengineering (BPR) [17], which aim at measuring efficiency and effectiveness of an existing business. In literature, BPR is often evaluated according to a set of dimensions in the effects of redesign measures, e.g. time, cost, quality and flexibility [104], cost, quality, service and speed [66] or cycle time, cost, quality, asset utilization and revenue generated [91]. Ideally, a redesign or modification of a process decreases the time required to handle incidents, it decreases the required cost of executing the process, it improves the quality of the service that is delivered and it improves the ability of the process to react flexible to variation [17]. However, a property of such an evaluation is that trade-off effects become visible, which means that in general, improving upon one dimension may have a weakening effect on another (see Figure 2.6).

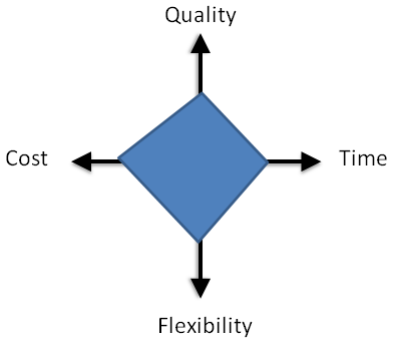

Figure 2.6.: Dimensions in the effects of process (re)design (based on [104]).

BPMN

Business Process Model and Notation (BPMN), previously known as Business Process Modeling Notation, is a graphical representation for specifying business processes in a business process model [153]. BPMN is a standard for business process modelling maintained by the Object Management Group (OMG). The graphical notation for specifying business processes is based on a flowcharting technique. The data mining patterns, which

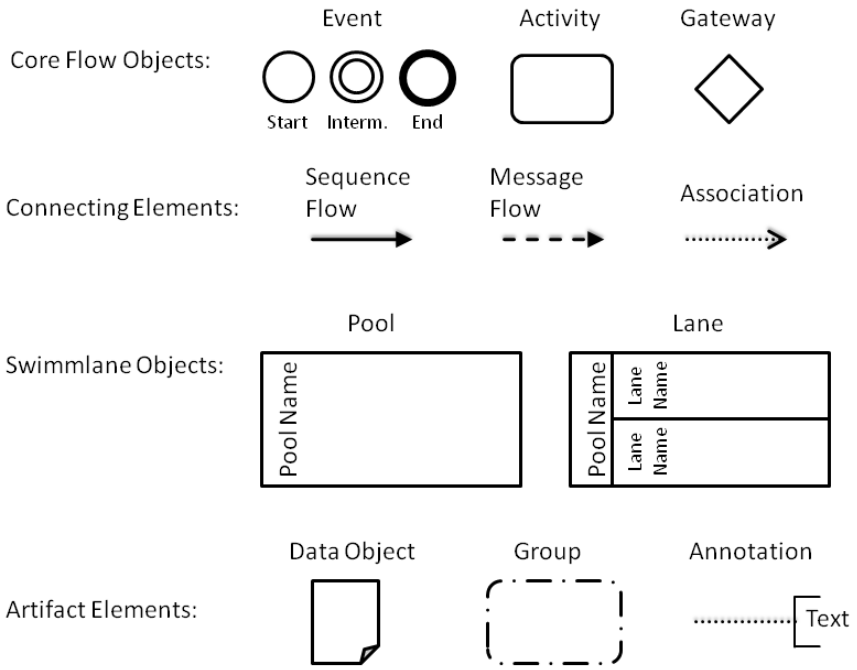

Figure 2.7.: BPMN overview.

will be described in Chapter 5, are visualized using BPMN. In detail, the modelling in BPMN is based on a small set of graphical elements from the following categories [154]:

- **Flow Objects** are the main describing elements that represent *Events*, *Activities* and *Gateways*. An Event is something that happens in the course of the process and affects the flow of the process. There are three types of Events, based on when they affect the process flow: Start, Intermediate, and End. Events are visualized as circles with open centres to allow internal markers to differentiate different triggers or results. An Activity represents work that is performed in the process and can be atomic or non-atomic. Activities are visualized by a rounded-corner rectangle. A Gateway is used to control the divergence and convergence of the process flow and determines decisions, forking, merging, and joining of paths. Gateways are represented by the diamond shape with internal markers to indicate the type of behaviour control.

- **Connecting Objects** are used for connecting the Flow Objects. Connecting Objects represent *Sequence Flow*, *Message Flow* and *Associations*. A Sequence Flow is used to show the order (the sequence) of the activities in a process. Sequence Flows are visualized by a solid line with an arrowhead. A Message Flow is used to show the flow of messages between two separate process participants that send and receive them. Message Flows are visualized by a dashed line with an arrowhead. An Association is used to associate data, text, and other Artifacts with flow objects and to show the inputs and outputs of activities. Associations are visualized by a dotted line with an arrowhead.

- **Swim lanes** are used as visual mechanism of organising and categorising activities. A *Pool* consists of different *Lanes*, a lane holds the Flow Objects, Connecting Objects and Artifacts. Two separate Pools represent two different process participants.
- **Artifacts** represent Data Objects, Groups and Annotations. *Data Objects* are used to show how data is required or produced by activities and are connected to activities through Associations. *Groups* are visualized by a rounded corner rectangle drawn with a dashed line. Grouping can be used for documentation or analysis purposes, but does not affect the Sequence Flow. *Annotations* are a mechanism for a creator of the diagram to provide additional text information for the reader.

Figure 2.7 gives an overview over the core BPMN elements. In Figure 2.8 we can see an example of a process modelled in BPMN. The start of the process is modelled by the Start Event. After the process starts, two tasks are performed for receiving and checking an order, which are modelled by two Activities. After that, a Gateway visualizes that there is a decision on whether the order is valid. The X in the Gateway further specifies that it is an XOR-gateway, which means that the sequence flow can follow only one way. If the order is valid, it is processed and closed. If not, the order is rejected. Finally, the process ends, which is modelled by the End Event.

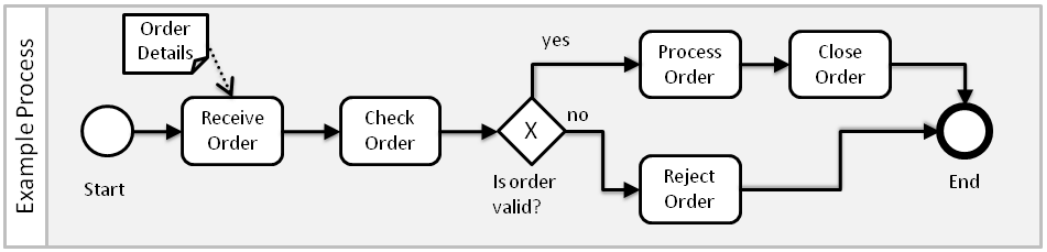

Figure 2.8.: Example of a Process modelled in BPMN.

Triana

Triana [128], a Java-based application distributed under the Cardiff Triana Project Software License (based on the Apache Software License Version 1.1), has been developed at Cardiff University as part of the GridLab [1] and GridOneD [2] projects. Triana consists of two distinct components: a graphical workflow editor for visual composition of workflows and a workflow manager (also known as 'engine') for executing workflows.

In a Triana workflow, any atomic operation is represented by a separate workflow unit. A workflow unit is supposed to be a light-weight component that is concerned with the correct progression of the workflow, but it does not implement the operation itself. The actual operation may take the form of a web service or a WSRF-compliant service. The Web Services Resource Framework (WSRF) is a family of OASIS-published specifications for web services [3]. The unit, however, implements the required logic for setting the necessary properties for execution, and for passing input data to the service. It may contain classes for visualization of the results of the conducted operation and may pass the output data to the next unit in the workflow. The properties of each unit can be modified

by using a control dialogue box. The input and output data of a unit are represented by simple or complex data types.

In the GUI, workflow units can be packed into folders in a tree-like structure. A workflow is created by selecting and dragging units from the folders and dropping them onto a workspace. The workflow developer can connect units, which means that the output of one operation can be used as input to the next operation in the workflow. By doing so, it is possible to compose workflows of arbitrary complexity.

Triana is capable of discovering and binding to web services and WSRF-compatible services. When binding to a WSRF-service, each of its public methods are displayed as a single unit. Triana provides two units, WSTypeGen and WSTypeViewer, which process the WSDL file of the service, dynamically render the respective input and output fields and create request (i.e. input) and response (i.e. output) data types.

Workflows are executed by the workflow manager residing either on the user's client machine or on a dedicated manager machine in a grid environment. For this purpose the Triana workflow editor produces a Java object representing the visual workflow, which is then executed by the manager. The manager can be launched on any machine where an appropriate Java Virtual Machine (JVM) is installed. Although the execution is performed on a single machine, the tasks can be distributed, while using the execution machine as central synchronization manager. Triana's workflow manager is completely independent of the Triana workflow editor; it is self-contained and needs no additional software in order to execute pre-defined workflows.

Figure 2.9 shows a screenshot of the Triana environment. The DataMiningGrid Application Description Schema, which will be described in Chapter 3, is developed in a workflow environment which is based on Triana.

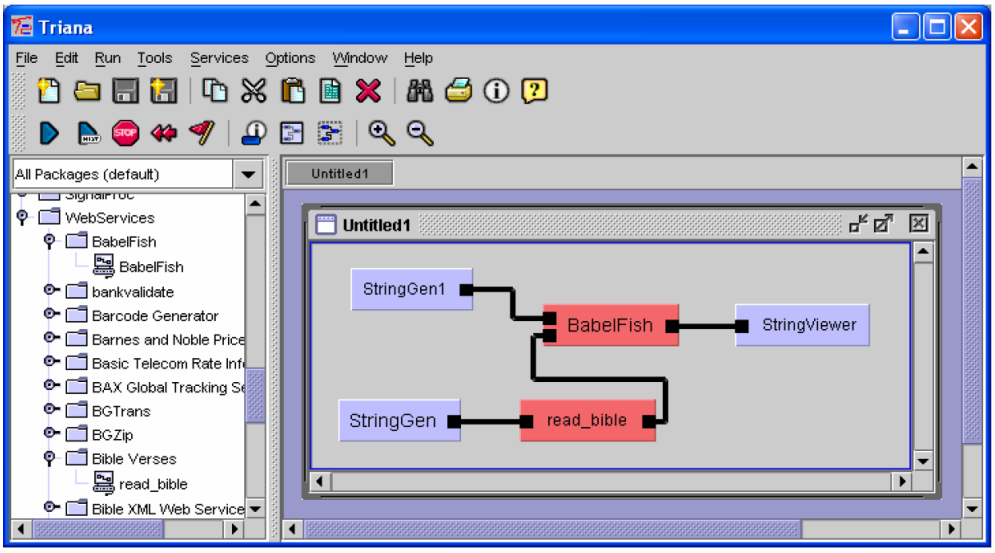

Figure 2.9.: Screenshot of the Triana environment (from trianacode.org)

2.3. Bioinformatics Scenarios

In this section, we will give an introduction into bioinformatics and describe some of the user groups, techniques and the data sources and repositories that are typically involved in bioinformatics scenarios. Furthermore, we describe data mining tools used in bioinformatics scenarios and present a concrete scenario as example.

2.3.1. Introduction to Bioinformatics

Bioinformatics is a big field of research that is conceptualizing biology in terms of macromolecules and applying information technology techniques from applied maths, computer science, and statistics to understand and organize the information associated with these molecules [87]. Several years of work by the scientific community, and in particular the bioinformatics community, have generated resources for data storage and standardization processes for data re-use so that a large quantity of previously collected genomic data is now available from public repositories and can be re-used for further analyses. In fact, it is now a general requirement that published genomic studies deposit their data in public databases so that analyses can be reproduced and/or data reused for further analyses [94]. Alongside this, tools have been generated that allow the analysis of such datasets. Many of these tools are open source and shared amongst the bioinformatics community. Although the type of data and type of analyses are continuously changing as the research in these areas is fast moving, some processes are starting to be of general interest, more frequently reproduced and they would benefit from standardization and reproducibility [145]. An example for this is the meta-analysis of gene expression datasets. From an abstract point of view, such a process includes data acquisition from different data sources, the preprocessing of these data, and a repeatedly performed analysis step based on merged data. The general schema of a meta-analysis process is visualized in Figure 2.10.

Typical research questions in bioinformatics are, e.g., finding predictive or prognostic biomarkers, defining subtypes of diseases, classifying samples by using genes, etc. In order to answer such questions, bioinformaticians, statisticians, medics and biologists combine different heterogeneous data sources from private or public repositories, and they develop and apply different analysis methods to the information extracted from the repositories and interpret the results until they have found good combinations of data sources and analysis methods. This process can be short or long, straightforward or complex, depending on the nature of the data and questions. According to [145], this is what we will call a *scenario*. As there exist a lot of complex interrelations among the data involved, a lot of domain knowledge is needed for setting up processes for such scenarios. Furthermore, there is a need for well-defined components for the processes, as it is usually too complex to develop all components needed for a scenario from scratch.

When composing a solution to an analysis problem, bioinformaticians mainly work together with biologists and clinicians to provide the best possible solution to the project questions. The solution is typically composed of different components in form of scripts or workflows (see also Section 2.1.4). Many of them are recycled from previous solutions and often need to be adapted. The choice of an available component depends on the general aims, the type and the amount of data that needs to be analysed. Although components

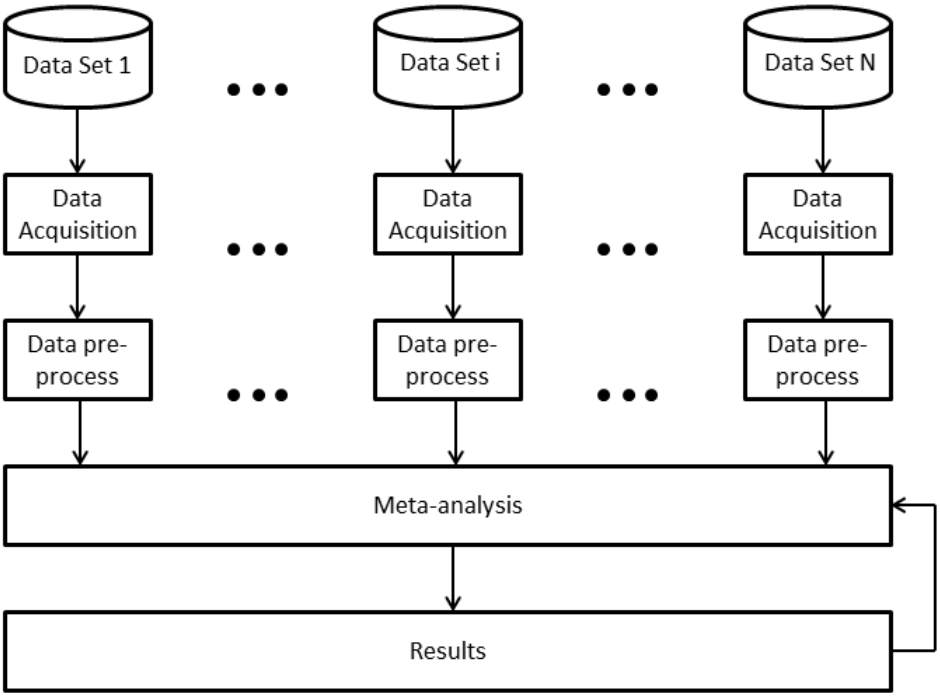

Figure 2.10.: Abstract process for meta-analysis of gene expression datasets (from [145]).

can be re-used, parameters often need to be revised and set properly to address the planned solution. Sometimes, a step of the solution needs to be customized in order to match the objectives, after that it needs to be integrated into the process and needs to be fully tested. In some cases, existing components are not available to pursue the project aims, so new components need to be developed and implemented from scratch.

Bioinformatics employs a wide range of techniques from maths, computer science and statistics, including sequence alignment, database design, data mining, prediction of protein structure and function, gene finding, expression data clustering, which are applied to heterogeneous data sources [87].

The implementation of bioinformatics scenarios is typically done with tools chosen as most appropriate by the bioinformaticians. Due to the various backgrounds, there is a quite heterogeneous set of tools and languages in use. Thus, analysis processes can be very different, depending of the type of data, the technology used, the tools used, the aim of the study, etc. Some common steps among processes for the analysis in the biomedical field are, e.g., data quality control, data normalization, data filtering, data visualization, and finding differentially expressed entities. Methods used for these steps are, e.g., local regression for normalization, grouping similar genes or samples together using k-means or hierarchical clustering for finding differentially expressed genes, and the Kaplan-Meier method for survival analysis.

The common procedure for data analysis for scenarios from bioinformatics can be described in an abstract way as follows [145]:

- Design of the experiment with the collaborators involved, and understanding of the methods and data needed.
- Based on the research question, data of different types is acquired from data repositories.
- Based on the research question, the methods are gathered.
- If method and data are ready available the process can start, otherwise more collection, development or implementation is needed and the process is temporarily halted until new data or tools are available.
- For each data of a certain type there is a pre-processing done.
- The data is merged.
- The analysis is performed.
- The results are discussed with the collaborators.
- The whole process can be iterated if new hypotheses are generated.

Analysis processes involve both manual and automated steps. Results of the analysis processes have to be interpreted to use them, e.g., for support to the clinical decision making process [99, 110]. The steps included in the common data analysis procedure are covered by the CRISP process model described in Section 2.1.2.

Many scenarios are based on privacy sensitive data from private or in-house data repositories. Often, this data cannot be moved due to the privacy policies and security directives. This means that the data might be inherently distributed. In other cases, the size of the data demands for distribution. An example of a scenario will be later given in Section 2.3.5.

2.3.2. User Groups

Bioinformatics is a collaborative discipline [106]. Bioinformaticians of today are highly qualified and specialized people from various backgrounds such as data-mining, mathematics, statistics, biology, IT development, etc. A typical analysis scenario involves multiple users and experts from different departments or organizations. Bioinformaticians are often working together with different collaborators [145]:

- IT people usually support bioinformaticians by providing and helping with the needed computational power, network infrastructure and data sharing.
- Clinicians are often a key point for patients information access and for the design and planning of the clinical part of the experiment.
- Pharmaceuticals Companies are might be interested in discoveries that have a commercial potential at the end of the research project.

- Statisticians and Data Miners can provide help on designing the study and correctly analysing the data.

- Biologists can provide help on designing the experiment and correctly interpreting the data. They can also be key people for managing the clinical samples.

2.3.3. Data Sources

Bioinformatics is an area in which analysis scenarios include huge amounts and different types of data [140]. Analyses in bioinformatics predominantly focus on three types of large datasets available in molecular biology: macromolecular structures, genome sequences, and the results of functional genomics experiments such as expression data [87].

Recent advances in technology enable collecting data at more and more detailed levels [106], from organism level, organ level, tissue level up to cellular and even sub-cellular level [87, 157]. In detail, we can distinguish several abstraction levels in multi-cellular organisms [145]:

- **Organism level**: an organism is the biological system in its wholeness, typically including a group of organs. Organism level related data is the clinical data, which usually comes from a hospital database manager.

- **Organ level**: an organ is a group of tissues that together perform a complex function. Organ level related data usually comes from a pathologist.

- **Tissue level**: tissues are groups of similar cells specialized for a single function. Tissue level data usually comes from a pathologist.

- **Cellular level**: in a multi-cellular organism such as a human, different types of cells perform different tasks. Cellular level related data usually is organized by a lab manager;

- **Sub-cellular level**: data at the sub-cellular level is composed by the structures that compose the cells. Usually, a data analyst can retrieve this data by performing ontologies analyses.

Additional information, e.g., includes the content of scientific papers. The main type of data for the scenarios presented in this thesis is microarray data. A DNA microarray (also known as gene chip) is a collection of microscopic DNA spots attached to a solid surface (see Figure 2.11). DNA microarrays are used, e.g., to measure the expression levels of large numbers of genes simultaneously. Several complementary technologies for measuring gene expression have evolved [67]. Examples for such so called platforms are Affymetrix [74] and Illumina [75].

Data from different sources is extensively collected in repositories potentially useful for subsequent analysis [106, 87, 157]. The number of such repositories has dramatically grown in the last fifteen years [53]. Some of these repositories are publicly available. Common public data sources include background information and literature, e.g. provided by

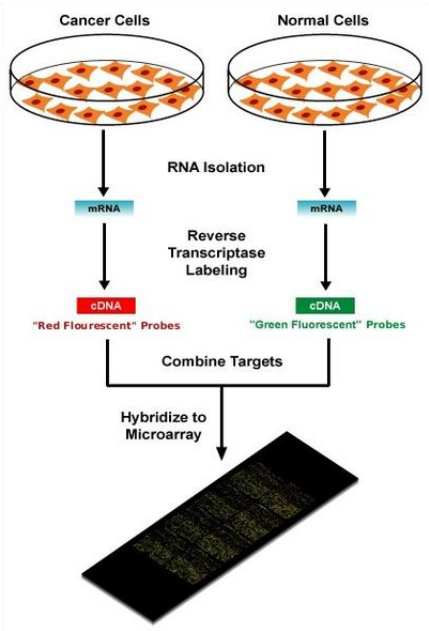

Figure 2.11.: Microarray Experiment (from [155]).

PubMed [100], or genomic and clinical data, e.g. provided by **GEO** [56] and **ArrayExpress** [13]. Private repositories are restricted in access and it is not always possible to copy the data due to privacy issues.

Comprehensive metadata describing the semantics of the heterogeneous and complex data are needed to leverage it for further research [152]. To address this issue, efforts exist for describing the data in a comprehensive way by domain specific ontologies [23]. Ontologies are, according to the definition given by Gruber, the specification of a conceptualization [131]. An ontology describes a domain or area of knowledge in a formal manner, understandable by both humans and machines, by providing consistent taxonomies of the concepts that belong to a domain, as well as the relations among those concepts. Ontologies have been employed to support the integration and selection of data.

When using heterogeneous data in analysis processes there is a need to normalize and homogenize the heterogeneous data [140]. Different data repositories adopt different standards, interfaces and data models, which hinder seamless data access. This is especially true for the free public access databases, private repositories with limited access, or even unstructured sources in the biomedical domain. Thus, a lot of background knowledge is needed for the data understanding and data preparation phase. As a result, great effort has been put into the development of systems that provide automatic uniform and homogeneous access to such sources - the so called semantic mediation systems [140]. Semantic mediation systems must deal with syntactic and semantic heterogeneities. The former refer to those due to differences in the interface to the data source, the language, and the data model. They are usually tackled by specific software modules wrapping the data source and translating the syntactic functionalities. The latter, in contrast, refer to differences in the schemas employed and data codification. These are handled by adopting

unified vocabularies or taxonomies that cover the domain of the sources being integrated.

Semantic mediation systems are able to work on distributed data repositories. However, efficiency and performance becomes an issue, as it has been shown that the process of translating queries for accessing the different databases is an NP-hard problem [64]. To deal with this, a compromise between efficiency and functionality must often be adopted.

2.3.4. Data Mining Tools in Bioinformatics

Although data mining methods have become integral part of biomedical research, standard data mining toolkits such as Weka [156] do not fully support the handling of raw biomedical data. The BioWeka project [57] extends Weka by various input formats for bioinformatics data and bioinformatics methods to allow users to easily combine them with Weka's classification, clustering, validation and visualization facilities on a single platform.

R [103] turned out to be one of the de-facto standards for data analysis in bioinformatics. In particular the associated project BioConductor [55] addresses the needs of the biomedical and biostatisticians community by providing R packages to analyse data issued from new technologies appearing in the biology laboratory. Bioconductor is an open source software project to provide tools for the analysis and comprehension of high-throughput genomic data and is based primarily on the R programming language. Numerous methods available as R/BioConductor packages and considered experimental a few years ago have been stabilized and became accepted standard in the analysis of high-throughput genomic data [150].

Bioconductor has advanced facilities for the analysis of microarray platforms like Affymetrix [74] and Illumina [75] and for expression arrays. In addition, it provides interfaces to community resources such as GEO [56] or ArrayExpress [13].

Bioinformatics tools can be also made available as web services. E.g., BioMOBY [19] is a well-known registry of web services used in bioinformatics. It allows interoperability between biological data hosts and analytical services by annotating services with terms taken from standard ontologies.

Furthermore, there exist efforts for enhancing service discovery [107] and pipelining of compatible services [80] as well as user interfaces that provide integrated access to many types of web services repositories with different protocols for resource description and invocation [90].

However, in this thesis we focus on data mining components and data mining scripts as defined in Section 2.1.4 rather than on web services which provide individual functionality. In principle, the functionality provided by a web service can be wrapped into a component by adding a command line based web service client. The other way around, a component can be wrapped into a web service by exposing its interface via web service operations.

2.3.5. Example Scenario

Genomic data is often analysed in the context of a clinical retrospective study, in some rare so far but increasing number of cases this can be a clinical trial or an intervention study [94]. The analysis of this class of data involves a variety of data ranging from

genomic, transcriptomic, proteomic data to imaging data. In addition, clinical and demographics data include attributes such as age, gender, tumour location, tumour diameter, tumour grade, tumor stage, histology, pathology attributes, nodal invasion, etc. The exact attributes for clinical data vary depending on the study or trial, and specific disease.

One scenario of the p-medicine project [97] describes a statistical analysis of tumour samples with associated gene expression data and clinical features [94, 112]. This analysis is a semi-standardized procedure which is usually performed by statisticians and bioinformaticians using several tools. In the scenario, cancer samples are analysed to identify gene signatures which may indicate whether a tumour is malignant or not, or whether the tumour will metastasise or not.

The aim of this scenario is to provide evidence that could assist in clinical decisions in the future. The patient is the focus and patient data are dealt with specifically. Although there is no mechanism to feed the results back to the single patients the results will increase the information about the disease and long term it will contribute to new and better treatment solutions. The scenario has the following inputs and outputs:

- Input: cancer probes for 'Uveal melanoma' in the Affymetrix HG-U133 Plus 2 format (set of CEL files, a single file is named like GSM550624.cel). Each file represents one tumour related to one patient and is anonymous. The files can be retrieved from `ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SeriesMatrix/GSE22138/` or `http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE22138/` .

- Input: Clinical Data as csv tables (e.g., named like GSE22138_clin.csv). These can be used to attach personal data to the anonymous cel file. The following clinical and personal properties are available: tissue, age, sex, eye (right, left), tumour location, tumour diameter (mm), tumour thickness (mm), tumour cell type, retinal detachment, extrascleral extension chromosome 3 status, months to the end point of metastasis.

- Output: Heatmaps, survival statistics, Kaplan-Meier plots and tests.

The scenario includes a manual preparatory part. The input data needs to be read, normalized and controlled. After the data is saved into a 'local workspace', the data needs to be manually examined by an expert. There are several reasons why the microarray data has to be normalized, among them are the following:

- The array is divided in regions and several of them can have a systematic higher (lower) intensity and background compared to others.

- Scratches on the glass.

- Presence of dirty zones.

- Undetectable spots (intensity lower than the background).

Quality control plots are used to detect if there are discrepancies between samples, e.g., if the samples belong to two or more different batches (batch effect). A degradation plot, an intensity-density plot and a box plot are used to check if one or more samples have

a behaviour (outliers) that differs from the others (the main group). In particular, the box plot should be performed before and after normalization. The normalization process usually allows minimizing both the random and batch effects. After the data is verified as 'clear', the main analysis process can start. First step is to find the differentially expressed genes between classed of samples. The result can then be visualized by a volcano plot or by an heatmap (see Figure 2.12). The Heatmap for example, is a method to visualize omic data and helps identify patterns of activity or expression with respect to clinical groups, for example patients with or without metastatic disease. The Heatmap visualization and interpretation is followed by a survival analysis where Cox regression is used, a risk index is generated and patient subgroups are visualized using the Kaplan-Meier plot, which shows the probability for survival of subsets of patients, for example those that are predicted as being at high risk and those that are predicted as low risk for a malignancy. Figure 2.12 shows some plots and results from the scenario.

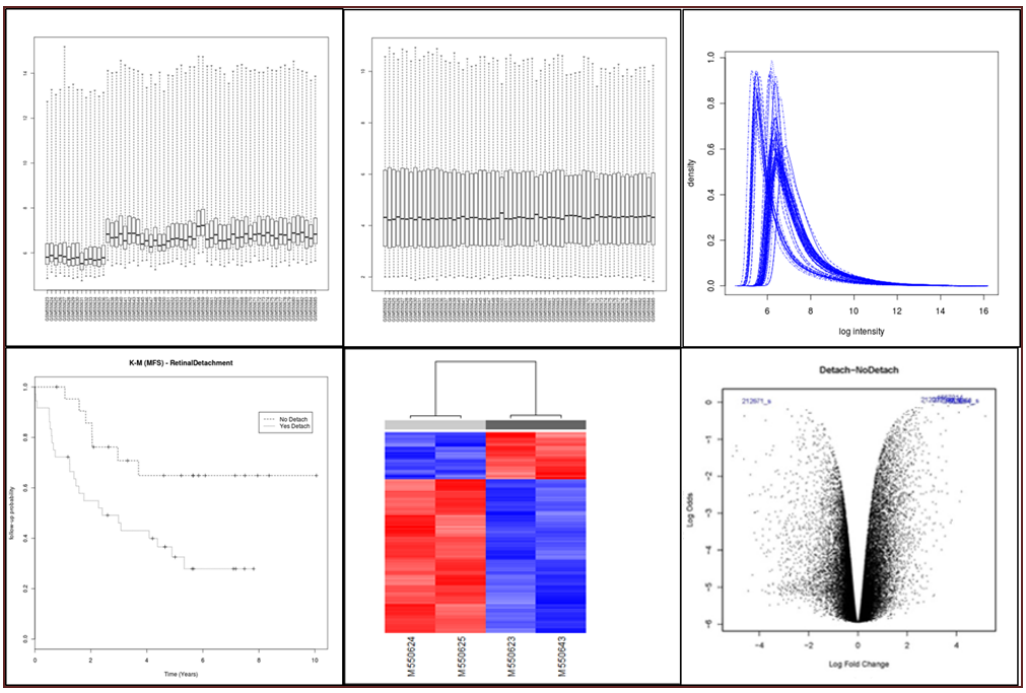

Figure 2.12.: Overview on scenario plots and results. From the left upper panel: unnormalized samples box plot, normalized samples box plot, intensity/density plot, survival analysis (Kaplan-Meier plot, metastases free survival), Heatmap (shows the regulation of the differentially expressed genes), Volcano plot (genes displayed in blue are those that are differentially expressed between the categories 'detach' and 'non-detach').

2.4. DataMiningGrid, ACGT and p-medicine

Parts of the results of this thesis were achieved in the context of the European research projects DataMiningGrid [35], ACGT [34] and p-medicine [97]. These projects included the development of grid- and workflow-based environments supporting data mining. In the following, the projects will be introduced briefly. The approaches which will be presented in the Chapters 3, 4 and 5 are implemented and evaluated in the context of the analysis environments of these projects.

2.4.1. The DataMiningGrid Project

The DataMiningGrid project [35], which was supported by the European Commission under FP6 grant No. IST-2004-004475, was a large-scale effort which aimed at developing a system that brings WSRF-compliant grid computing technology to users and developers of data mining applications. The output of the DataMiningGrid project is the DataMiningGrid system.

A challenge of DataMiningGrid was to develop an environment suitable for executing data analysis and knowledge discovery tasks in a wide range of different application sectors, including the automotive, biological and medical, environmental and ICT sectors [123]. Based on a detailed analysis of the diverse requirements of these applications, generic technology for grid-based data mining was developed in the DataMiningGrid project.

Existing grid technology already provided a diverse set of generic tools, but due to the generality the available functionality partially lacked to specifically support advanced data mining use-cases. In the DataMiningGrid project, enhancements to open source grid middleware were developed in order to provide the specialised data mining functionality required by the use-cases. This included functionality for tasks such as component discovery, data manipulation, resource brokering, and component execution. The result is a grid-based system including all the generic functionality of its middleware and additional features that support the development and execution of complex data mining scenarios.

The contribution of Chapter 3 is implemented in the DataMiningGrid system. The technical architecture of the DataMiningGrid system will be described in more details in Section 3.4.1.

2.4.2. The ACGT Project

The Advancing Clinico-Genomic Clinical Trials on Cancer (ACGT) project [34], which was partially funded by the European Commission under the project identifier FP6-2005-IST-026996, aimed at providing an open environment for supporting clinical trials and related research through the use of grid-enabled tools and infrastructure.

With the accelerating development of high-throughput technologies in the domain of biomedical research and of their use in the context of clinical trials, hospitals and clinical research centres are facing new needs in terms of data storage and analysis [149]. E.g., in the context of microarray analysis of a tumour biopsy the data for a single patientincludes 10'000s to 100'000s of gene-expression values summarizing up-to millions of microarray features [150]. New technologies based on imaging, genome sequencing and proteomics

are pushing even further the needs for data processing in the clinical research. The ACGT project aimed at addressing the analysis of such complex data sets by developing appropriate data exploitation approaches, integrating the know-how acquired in many independent fields into a powerful environment that physicians can easily and safely use, for the benefit of the patients [149]. Several initiatives with a similar goal started worldwide, among which NCI's caBIG (Cancer Biomedical Informatics Grid) [25] in the USA and CancerGrid in the UK [134].

The ACGT project aimed at providing an IT infrastructure supporting the management of clinical trials (e.g. patient follow-up), as well as the data mining involved in the translational research that often occurs in parallel. It is in relation to the latter aspect of the project that the need for a high-performance environment supporting the R language (see Section 2.1.3) and the vast collection of already existing biostatistics algorithms was recognized. The contribution of Chapter 4 is implemented in the ACGT system. The technical architecture of the ACGT system will be described in more details in Section 4.7. Furthermore, parts of the case study in Chapter 5 are based on the ACGT project

In ACGT, the problem of describing the content and semantics of heterogeneous data was addressed in the following way (summarized from [140]): The semantic mediation layer included in ACGT is designed to allow end users and client applications performing integrated queries against a set of heterogeneous biomedical databases. The ACGT Master Ontology was specifically designed within the ACGT environment to serve as unified schema of this mediation layer [23]. This ontology acts as database schema for performing integrated queries to a set of underlying databases. The main component of the layer - the semantic mediator - accepts queries in SPARQL [5], thus sticking to the most widespread standard for querying of resources in the web. Employing a self-designed mapping architecture, the mediator is able to detect which data sources are needed to solve a query and generate the queries for such sources. At this level, the mediator delegates on the database access wrappers, which hide the syntactic peculiarities of each database by offering SPARQL access to all of them. The mediator is then able to collect all sub-results and combine them in a single result-set that is returned back to the user.

The ACGT Query Editor is a web-based graphical tool designed to allow simple and intuitive access to the semantic mediator. SPARQL is the most widespread query language for RDF-based databases, but requires certain degree of technical knowledge to be able to construct queries with it. The aim of the ACGT semantic mediation layer was to offer integrated querying services to clinicians and biostatisticians who would presumably lack advanced technical background. The goal of the Query Editor was to allow such users to take advantage of the querying capabilities. The tool guides the user in the construction of SPARQL queries. The user simply needs to click on the elements which he wants to retrieve, and then select which restrictions (if any) should be included.

By the use of an ontology to describe the underlying data, differences in terminology, data representation and structure could thus be (to a certain degree) easily tackled. In addition, the use of an ontology offers additional interesting features. The mediator takes advantage of the hierarchical organization of the information in the ontology to offer functionalities such as testing if a query has a translation to a specific dataset. The Query Editor also manages the information of the ontology so that the user can build more complex queries with ease. For example, the Query Editor is able to generate

ontology views which encapsulate several source views at the same time, thus giving the user the possibility to build queries that retrieve and integrate data from all sources. Semantic mediation techniques will be used for specifying data requirements for data mining processes in Section 5.6.

2.4.3. The p-medicine Project

The project p-medicine - "From data sharing and integration via VPH models to personalized medicine" - is a 4-year Integrated Project co-funded under the European Community's 7th Framework Programme (FP7/2007-2013) under grant agreement No. 270089 [97]. It aims at developing new tools, IT infrastructure and VPH models to accelerate personalized medicine for the benefit of the patient.

The p-medicine consortium is creating a "biomedical platform to facilitate the translation from current practice to predictive, personalized, preventive, participatory and psycho-cognitive medicine by integrating VPH models, clinical practice, imaging and omics data" [109]. The project includes multi-level data collection within clinico-genomic trials and interdisciplinary analysis by clinicians, molecular biologists and other specialists involved in life science, as this is mandatory to further improve the outcome of cancer patients' treatment. It will allow for merging research results of biomolecular findings, imaging studies, scientific literature and clinical data from patients.

The main part of the engineering of the requirements, the example scenario presented in Section 2.3.5 and parts of the case study in Chapter 5 are based on the p-medicine project.

2.5. Wrap-up

In this chapter we have introduced the preliminaries and state-of-the-art and prior work from the field of data mining and scientific data analysis. In detail, we introduced data mining and the standard data mining process model CRISP. In addition, we described how data mining is used in practice and gave some basic definitions. In the area of bioinformatics, data mining is used in the context of larger systems. Thus, subsequently we presented background information on the area of distributed data mining, grid computing, process modelling and workflow environments. After that, we gave an introduction into bioinformatics including user groups, data sources and tools, and presented a clinical trial scenario as example of a scenario in bioinformatics. Finally, we introduced the European research projects DataMiningGrid, ACGT and p-medicine, in which the contributions of this thesis are partially implemented.

3. Integration of Existing Data Mining Components into Grid-based Systems

The goal of this chapter is to introduce an approach for the integration of existing data mining components into distributed systems based on grid middleware. For many analysis purposes there already exist available data mining components which can be reused for setting up data mining processes. However, such available components need to be integrated into grid-based analysis environments used in the medical and bioinformatics domain. But, today's grid-based analysis environments are complex in handling in terms of user interaction, resource management, and service discovery, which makes the integration of data mining components into such environments a complex task. Thus, there is an approach needed for the integration and reuse of state-of-the-art data mining components by users without deeper knowledge on the underlying grid middleware and without intervention on the component side. In the following, we present an approach that allows for an integration of atomic components into grid environments based on a metadata schema. The approach addresses the needs of the community to setup data mining processes by reusing available data mining components.

This chapter first introduces the requirements for the integration method in Section 3.1. Second, related work is presented in Section 3.2. Subsequently, we present our approach for integrating data mining components that have been developed in a single computer environment into distributed systems based on grid middleware in Section 3.3. The process of integrating components into grid-based systems is also called grid-enabling data mining components. Our approach is based on a meta-data schema definition, the Application Description Schema (ADS), and associated services. The schema is used to manage user interaction with system components of the grid system in order to grid-enable existing data mining components, to register and search for data mining components, to match requests for the execution of data mining components with suitable computational resources, and to dynamically create user interfaces. Section 3.4 provides case studies that demonstrate the applicability of our approach. We show that it is possible to implement the architecture of our approach in the grid environment of the DataMiningGrid project, that it is possible to create an user interface which is easy to use by users who do not have knowledge on grid technology, and that our approach supports standard data mining components and data mining scenarios in general. Finally, Section 3.5 wraps up. This chapter is mainly based on [123, 124, 142, 143].

3.1. Requirements

In this chapter, we focus on supporting the user in reusing standard data mining components that are developed for single processor environments, e.g. provided by the Weka data mining toolkit [156] or R [103] (see also Section 2.1.3), in a grid-based system with small effort by an approach on grid-enabling these data mining components.

In the following, we assume that the analysis scenario of the user can be addressed by reusing and correctly composing available data mining components. Thus, we do not focus on the development of new data mining components, but on an efficient development of new data mining processes in grid-based distributed systems based on existing data mining components.

The amount of electronically available data for bioinformatics scenarios is increasing over time. The fact that the data is not focussed on a specific research question necessitates complex process chains for analysing the data. Resulting from that, the reuse of existing components is needed for efficiently setting up analysis solutions. Hence, the complexity of analysis solutions is increasing, and so does their demand for computational and storage resources.

Scenarios in bioinformatics involve data from different hospitals and corresponding research organizations (see Section 2.3). Thus, the data is inherently distributed. Researchers in different organizations are collaborating when working on analysis scenarios based on these data. Typically, the data includes detailed information on patients. Thus, the data are privacy sensitive.

Grid technology [49, 50] is used to address the needs of the bioinformatics scenarios for providing an environment for handling distributed, privacy sensitive data that can be used in analysis processes (see also Section 2.2.2).

From the characteristics of bioinformatics scenarios, data mining and grid-based systems several requirements can be derived for the method for integrating and reusing data mining components in grid-based systems. Basing on the requirements engineering in the context of the DataMiningGrid project [123], the following requirements have to be met from the point of view of an bioinformatician that is developing a solution for a bioinformatics scenario:

- **Reuse by users themselves**: As there exists a huge amount of data mining components for very different purposes, it is not feasible to integrate all components in advance. Thus, it should be an easy task for users to integrate and reuse components by himself.
- **Reuse without component modification**: The users should be able to integrate existing data mining components with little or no intervention in existing code of the component, as they might have no knowledge on the implementation of a component, as they might not be able to further develop atomic components at all or as the code of the component might be unavailable.
- **Extensibility without platform modification**: It should not be necessary to programmatically modify the grid system, neither on the server nor on the client side, to integrate a new data mining component.

- **Transparency**: The details of the underlying technology of the grid system should be hidden from the users, since they usually do not have expertise in distributed systems.

Data mining processes include data intensive and complex tasks that require many computational resources and a lot of domain knowledge. From CRISP [121], which was already described in Section 2.1.2, results that a lot of very different data mining processes exist in the form of iterative, complex workflows. Such workflows are, e.g., published at repositories such as myExperiment [58]. Thus, the following requirements have to be met [123]:

- **Generality of the extensibility mechanism**: The integration method should cover a variety of different data mining components and should support a wide range of data mining processes. Thus, it has to support the common steps of data mining processes that can be seen as standardizable as described in Section 2.1.3: chaining, looping, branching, parameter variation (parameter sweep), cross-validation, and data partitioning.
- **Efficiency**: Data mining components should be able to run in an efficient way in the grid system, based on batch job execution and parallel processing of the standard tasks parameter sweep and cross-validation (see Section 2.1.3) to save execution time. This also requires shipping of data, which means to send the data to the machine where the data mining component is executed, and shipping of components, which means to send the component to the machine where the data it operates on is located. In addition, the integration into the grid environment should introduce only little overhead.

Shipping of data is important in cases where either the full data set is partitioned to facilitate distributed computation (e.g. for k-NN, where objects are assigned to the class most common amongst its k nearest neighbours [156]), or where the same source data set is moved to different machines and repeatedly analysed (e.g. for ensemble learning, where different data mining components are executed and the results are combined afterwards). Shipping components to the location of the data for execution helps to save data transfer time. This is one of the major options in setting up a distributed data mining process. It is required when, as it is often the case, no pre-configured pool of machines is available that already has the data mining functionality installed. The option to ship components to data allows for flexibility in the selection of machines and reduces the overhead in setting up the data mining environment. It is especially important when the data naturally exists in a distributed manner and it is not possible to merge it. This may be the case when data sets are too large to be transferred without significant overhead or when, e.g., security policies prevent the data to be moved.

3.2. Related Work

In recent years, a number of environments for grid-enabling data mining tools have been described. The importance of extensibility for data mining platforms has already been

argued in [158]. Today, there exist a lot of systems which are capable of distributed data mining. While in [123] a general comparison of a variety of systems has been done, we focus on grid-based system that are related to OGSA here.

GridMiner [22] is designed to support data mining and online-analytical processing (OLAP) in distributed computing environments. The system is based on a service oriented architecture (SOA) supporting OGSA grid services and OGSA-DAI database access. GridMiner implements a number of common data mining algorithms, including parallel versions. In the GridMiner system, each data mining component is integrated by wrapping it by a single OGSA-based grid service. This means that for each data mining component to be integrated there is a need for developing a new service. Thus, the approach does not fully support the reuse by the users themselves, as the integration of components in terms of creating new grid services is complex for users. Furthermore, the approach does not support extensibility without component modification, as the components have to be developed into grid services.

The Federated Analysis Environment for Heterogeneous Intelligent Mining [8] (FAEHIM) implements a toolkit for grid-based data mining. It consists of grid services for data mining and a workflow engine for service composition. Based on algorithms taken from Weka, the grid services split into the types classification, clustering and association rules. The services are not limited to algorithms from Weka, but for each data mining component or set of components a new service has to be developed. Thus, similar to GridMiner, the approach does not fully support the reuse by the users themselves, as the integration of components in terms of creating new grid services is complex for users. In addition, the approach does not foresee that the components can be executed as grid jobs, which limits the ability for efficient execution.

Weka4WS [127] is a framework for supporting distributed data mining on grid environments, designed by using the Web Service Resource Framework (WSRF) to achieve integration and interoperability with standard grid environments. The Weka4WS system is based on the data mining toolkit Weka. A single web service interface is used to provide access to the data mining algorithms implemented in Weka. Thus, the extensibility of the system is restricted to algorithms that are contained in the Weka toolkit, which constraints the generality. Similar to FAEHIM, the approach does not foresee that the components can be executed as grid jobs, which limits the ability for efficient execution.

Knowledge Grid (K-Grid) [26] is a service-oriented framework that has been designed to provide grid-based data mining tools and services. The system facilitates data mining in distributed grid systems and related tasks such as data management and knowledge representation. The system architecture is organized in different layers: The Core K-Grid Services handle the publication and discovery of data sources, data mining and visualization tools, and mining results as well as the management of abstract execution plans that describe complex data mining processes. The High-level K-Grid Services are responsible for searching resources, the mapping of resource requests from the execution plans to the available resources in the grid, and the task execution. The Knowledge Directory Service (KDS) is responsible for maintaining the descriptions of components that can be used in the Knowledge Grid. The description of components is based on XML documents, which are stored in a Knowledge Metadata Repository (KMR). The metadata about data mining components includes information about the implemented

task (e.g., classification, clustering, regression), complexity of the used algorithm, location of executables and manuals, syntax for running the program, and format of input data and results. From the technical point of view, the approach for describing and using pre-existing components is similar to our approach. However, in K-Grid the integration is focused only on components that are installed on computers that are part of K-Grid, which limits the generality of the approach. It does not support the reuse by the users themselves, as the components need to be installed by administrators on the machines attached to the grid first.

Discovery Net [9] provides a service-oriented computing model for knowledge discovery, focused on scientific discovery from high-throughput data generated in life science, geo-hazard and environmental domains that allows to access and use third party data analysis software and data sources. Based on Globus Toolkit, the system provides services to declare the properties of analysis tools and data stores, to integrate various data sources (e.g. SQL-Databases, OGSA-DAI sources etc.), to discover and compose Knowledge Discovery Services, to integrate data from different data sources using XML, and to deploy knowledge discovery processes as new services. In Discovery Net, components are defined as encapsulated code, which can be bound to a resource or resource-free. Resource-free means that the execution engine decides where to execute the component, whereas bound means that it can only be executed on a certain computing resource. An XML schema called Discovery Process Markup Language (DPML) is provided for describing processes including data mining components. The description of a data mining component includes information on input and output types, parameters, constraints (e.g. range of validity, discrete set of possible values, optional or required) and registration information such as category, keywords and description [38]. However, each component has to be made available via a separate service, which makes it complex for users to reuse the components by themselves in grid service development. In addition, support for the parallelization of cross validation and parameter sweeps is not supported by the description schema, which limits the ability for efficient execution.

3.3. Integration of Data Mining Components

In this section, we will present our approach on grid-enabling data mining components and describe which architecture is needed to fulfil the requirements from Section 3.1. We assume that the data mining component to be reused and integrated into the grid environment is a component according to the definition from Section 2.1.4. The approach for the integration of data mining components is based on a meta-data schema definition, which is called Application Description Schema (ADS), and associated services. The schema is used to manage user interaction with components of the grid environment to grid-enable existing data mining components, to register and search for components on the grid, to match jobs with suitable computational resources, and to dynamically create user interfaces. The process of grid-enabling data mining components is supported by a web based procedure providing a user interface for end users. By our approach we address the requirements on the integration and reuse of data mining components in grid environments.

In the following, we introduce the architecture of our approach. Subsequently, we present the details of the Application Description Schema. Finally, the processes for registering and for executing data mining components will be presented.

3.3.1. Layered Architecture

The Open Grid Services Architecture (OGSA) [46], presented in Section 2.2.2, describes a reference architecture for a service-oriented grid computing environment. This architecture represents a blueprint for grid-based systems. Thus, our method for integrating data mining components into grid-based systems has to build on the grid reference architecture.

As it is required to allow for the reuse of data mining components without the need to modify the component, our approach for the integration needs to be able to directly reuse the executable files of the components. From the requirement for reuse without platform modification results that there should be no need for extending the grid middleware by new services or to add new client components for integrating and executing new data mining components. Thus, our approach needs to base on a generic mechanism for the integration at the level of the grid middleware and needs to allow for generic client components. In addition, the details of the grid middleware should be hidden from the user to achieve transparency. To address these issues, we foresee a registry for data mining components based on meta-data and client components that can build dynamic GUIs based on that meta-data. OGSA based grid environments provide Information Services, which allow for service and resource discovery. Thus, the component registry of our approach can build upon the OGSA Information Services. The client user interfaces are part of the application layer of OGSA and can build on available middleware services. In our approach, the client components are designed to create GUIs dynamically and to allow the specification of data inputs, parameters and the configuration of the data mining components.

In addition, it is required that the data mining components can be executed in the grid environment in an efficient way. From this results that there is a need for batch execution of data mining components including the possibility to transfer the data or the data mining component to certain computational resources in the grid. Services for Resource and Execution Management, Data Management and Monitoring are provided by OGSA. Thus, the execution of data mining components in the grid can build upon the Execution Management Services of OGSA, which provide job management functionality for job execution on grid resources managed by the Resource Management services. Data files can be transferred by the Data Services of OGSA anyway. As our data mining components are based on executable files and not on individual grid services, they can be also transferred with the same mechanisms.

The architecture of our approach can be described as a layered architecture with 3 layers: The fabric layer, the middleware layer and the application layer (see Figure 3.1). The bottom layer (fabric) consists of the infrastructure services, providing computational resources and software resources. This includes the executable files and associated libraries of the components, the input and output data as well a the machines where the executions take place.

The middle layer (middleware) includes the services provided by the grid middleware foreseen by the grid middleware layer of OGSA (as described in Section 2.2.2). The

important OGSA services for our approach are the following: The **Information Services**, to make components searchable in the registry mechanisms; the **Data Services**, to transfer executable files, associated libraries and input and output files; the **Resource & Execution Management Services**, to execute components on grid resources; the **Self-Management Services (Monitoring & Analysis)**, to monitor and analyse the executions.

In the top layer (application), there exist client components which interface with the grid middleware services and provide user interfaces for the end users. The client components include the application enabler, which is used to grid-enable existing data mining components, and a set of clients to execute the grid-enabled components in the grid environment. The latter allow for searching components in the grid registry (Explorer), for specifying the execution details such as parameters and inputs (Control), for starting and monitoring the execution (Execution) and for viewing provenance information about the execution (Provenance).

When grid-enabling data mining components, a detailed description of the components is needed for the integration with the core components of OGSA. This information represents meta-data of the component. In our approach, the handling of this information is based on describing the data mining component by meta-data in a single schema that includes the necessary information: the Application Description Schema (ADS). The interfaces between the client components in the application layer and the services in the middleware layer are based on the ADS. Details on the ADS meta-data description schema will be presented in the next section.

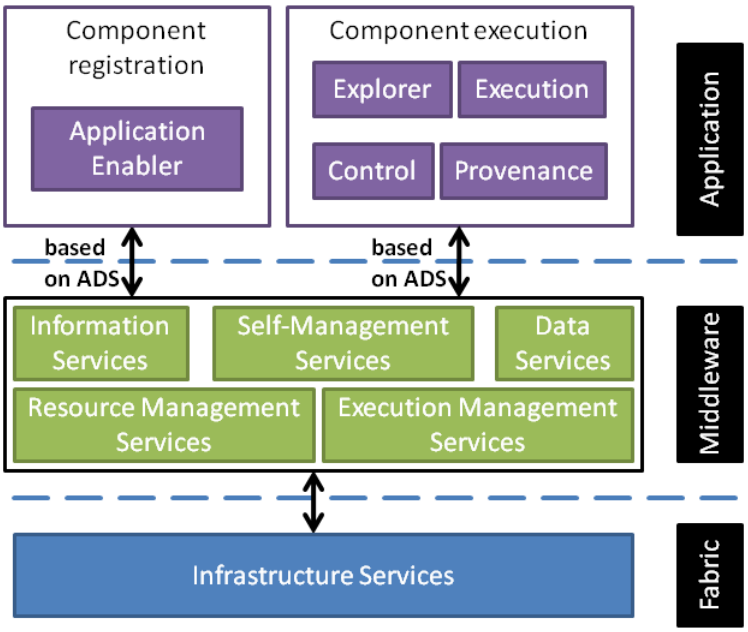

Figure 3.1.: Architecture for grid-enabling data mining components.

3.3.2. The Application Description Schema

In our approach, the information about the data mining components needed by the middleware services of OGSA to allow for integration and execution in the grid environment are maintained by a single meta-data schema. The schema forms the basis for interactions and information exchange with the grid middleware. Figure 3.2 visualizes this idea. The schema includes a common part, which could be used for describing any command line component independent from the application domain, and a data mining specific part. By this, our approach can be adapted to other domains by reusing the common part of the schema and just exchanging the data mining related part.

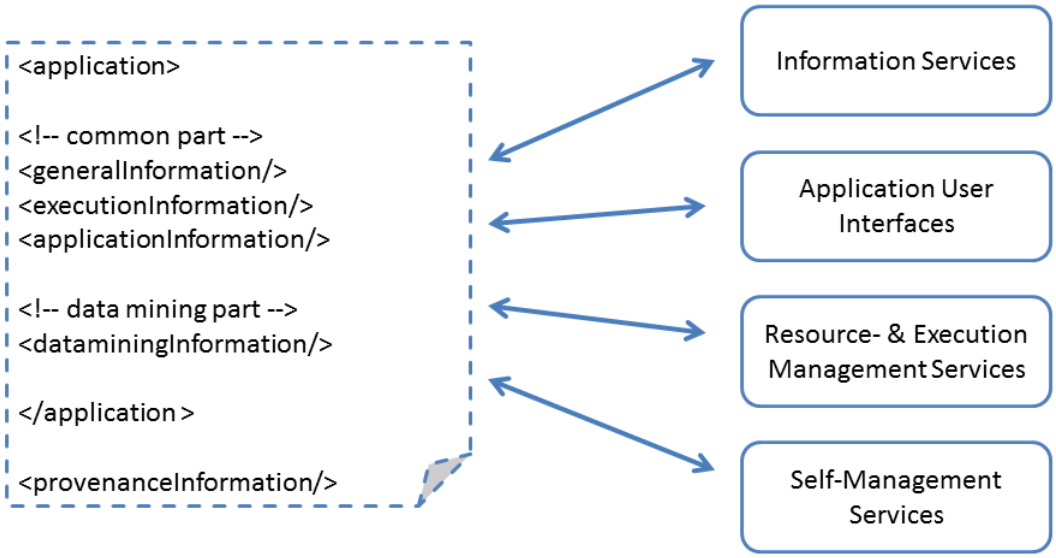

Figure 3.2.: The Application Description Schema (ADS) for interaction and information exchange between grid middleware services of OGSA.

For registering and executing data mining components in the grid environment, there is information needed for the following OSGA services:

- **Information Services.** There is a need for information on the grid-enabled data mining components in order to make them searchable in the registry mechanisms provided by the information services of the grid middleware. This includes information that allows for searching and finding the new data mining component in the grid registry, such as name, domain, and description.
- **Data Services.** There is a need for uploading executable data mining components and their inputs and the transfer of the outputs by the data services of the grid middleware. Thus, there is information needed on the location of the executable file of the component as well as the input and output files.
- **User Interfaces or Workflow Environment.** There is a need for (dynamic) user interfaces that allow the specification of data inputs, parameters and the configuration of the data mining components as part of the application layer of the

grid middleware. Thus, there is a need for detailed information on the parameters, options, inputs and outputs of the component.

- **Resource & Execution Management Services.** There is a need for specifying information for executing the data mining components to enable a correct execution by the grid middleware. This includes information on the system requirements of the data mining component such as the runtime environment and the required memory. In addition, the information on inputs and parameters of the component is needed here also.

- **Self-Management Services (Monitoring & Analysis).** There is a need for storing detailed information on the execution of the data mining component including inputs, parameters, outputs and execution and monitoring information for analysis and reuse.

Similar to the approaches presented in [26, 9] we use XML to define the ADS metadata schema for describing the executable data mining component in order to define how it is used with the system.

The ADS descriptions include general information about the data mining component (e.g., metadata like a name and a textual description), execution information (e.g. the executable file, programming language and required libraries), application information (e.g. the number and type of the component's options and data in- and outputs and the minimum resource requirements) and data mining specific information (such as technique and CRISP phase), which have to be specified when grid enabling the component.

Before executing the component, users have to specify additional information that is needed for the execution such as the values for the parameters, the in- and output files or directories or the execution machine if the component is to be shipped to a certain machine. Thus, each grid-enabled data mining component refers to a particular instance of the ADS, which are passed, at different levels of specification, between the system components to manage interaction. After the execution of a component, provenance information can be collected about the execution in the grid.

In the following, we will present the details of the ADS definition. Please see the Section A.1 of the Appendix for a complete XML schema definition of the ADS.

Main Schema

As visualized in Figure 3.2, the ADS is divided into different parts: the common part, the data mining part and the provenance part. Below we summarize the structure and content of the ADS. The common part of the ADS describes aspects that are common to all components enabled to run on the grid. It captures component-relevant information that falls into three major categories: *generalInformation*, *executionInformation* and *applicationInformation*. The data mining part describes information that specifically relates to data mining components and captured in the category *dataminingInformation*. Furthermore, provenance information can be collected in the category *provenanceInformation*. The reason for this segmentation is that the grid services are depending on different kinds of information (see also Figure 3.3).

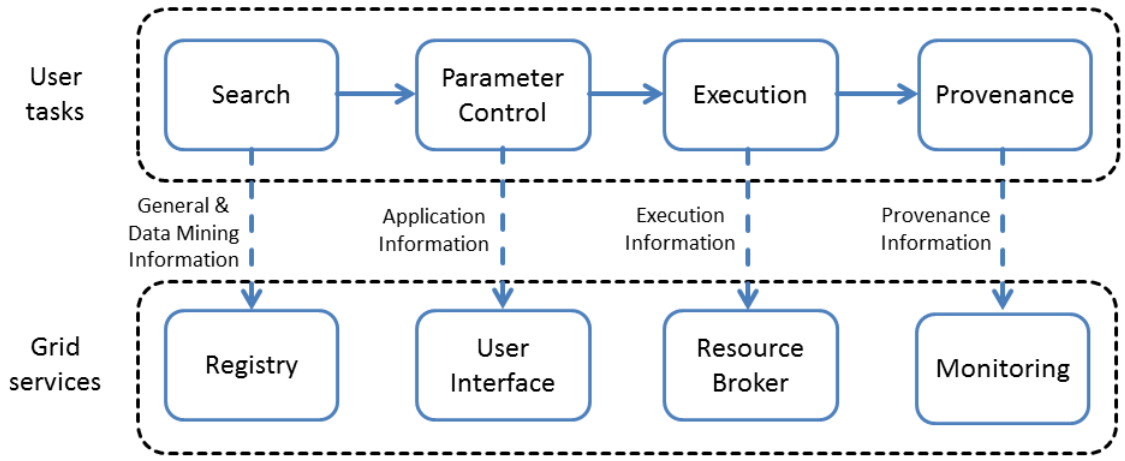

Figure 3.3.: The different parts of the ADS address the need for information for different grid services.

The ADS schema definition consists of the definition of the ApplicationDescriptionType, a single *application* element of the type ApplicationDescriptionType, and a single *provenanceInformation* element of the type ProvenanceType (see Listing 3.1 and also Section A.1.1 in the Appendix). The ApplicationDescriptionType is defined as a complexType consisting of a sequence of 4 elements. These elements are the following: *generalInformation*, *dataminingInformation*, *executionInformation* and *applicationInformation*. Details on these elements will be presented in the following.

Listing 3.1: Main Part

```
<xsd:complexType name="ApplicationDescriptionType">
        <xsd:sequence>
                <xsd:element name="generalInformation"
                    type="com:GeneralInformationType" />
                <xsd:element name="executionInformation"
                    type="com:ExecutionType" />
                <xsd:element name="applicationInformation"
                    type="com:ApplicationinformationType" />

                <xsd:element name="dataminingInformation"
                    type="dm:DataMiningType" />
        </xsd:sequence>
</xsd:complexType>

<xsd:element name="application"
    type="app:ApplicationDescriptionType" />

<xsd:element name="provenanceInfo" type="prov:ProvenanceType" />
```

generalInformation

The *generalInformation* part specifies different general aspects of the component which will be useful for different purposes (searching for components, provenance, administration). The generalInformation element is of the type GeneralInformationType, which is defined as a complexType consisting of several elements and attributes (see Section A.1.2 in the Appendix for details) and covers the following information:

- shortDescription - a short description of the component.
- long description - a long description of the component.
- comment - a comment about the component.
- id - a unique id that identifies the component.
- vendor - the vendor of the component.
- swVersion - the software version of the component.
- codeVersion - the source code version of the component.
- build - the build version of the component.
- uploadDate - the date when the component was uploaded.

dataminingInformation

The *dataminingInformation* part of the ADS describes information that is specific to the data mining aspect of the component, which is used for the registration information needed by the Information Services of the grid middleware. This information is used to facilitate the discovery of data mining components and data mining-aware resource brokering. The dataminingInformation element is of the type DataMiningType, which is defined as a complexType consisting of several attributes (see Section A.1.3 in the Appendix for details). In detail, this includes the following information:

- applicationDomain - a description of the domain (e.g. Data Mining).
- applicationGroup - the software suite the component is part of (e.g. Weka).
- applicationName - the name of the data mining component (e.g. IBK).
- functionalArea - the data mining method or methodology the component employs (e.g. classification).
- technique - the data mining algorithm used by the component (e.g. KNN).
- CrispDMphase - the CRISP phase [121] the component is used for (e.g. Modeling).

executionInformation

The *executionInformation* part contains information relevant to the execution of the component, which is used for the job descriptions needed by the Execution Management services of the grid middleware. The executionInformation element is of the type ExecutionType, which is defined as a complexType consisting of several elements and attributes (see Section A.1.2 in the Appendix for details). In detail, this information includes the:

- execution environment - the execution environment (e.g. java, bash-shell or python), including information on the executable file(s) and the commands and arguments used at start-up.
- requiredLibrary - specifies if there are additional libraries needed for executing the component.
- stage-in - specifies if additional files need to be staged-in.
- remoteSubdirectory - specifies if the files have to be staged-in into a certain remote subdirectory.

applicationInformation

The *applicationInformation* part is by far the most comprehensive one. It provides configuration information of the component such as options, parameters, inputs and outputs, which is used for the job descriptions needed by the Execution Management services and the creation of the user interfaces for specifying the execution details. The applicationInformation element is defined as a complexType consisting of several elements (see Section A.1.1 in the Appendix). In detail, this information includes the:

- options - options or parameters used when the data mining algorithm implemented by the executable file is executed. All options or parameters are typed and can be optional (a default may or may not exist and may be overwritten by the user) and hidden (an option transparent to the user).
- dataInputs - data input slots are used to describe the component's input data, i.e., data types (file or directory), transfer protocols permissible for particular data, physical location of data, and other descriptors (e.g. whether data input is optional or mandatory).
- dataOutputs - data output slots are used to describe the component's output data.
- hostEnvironmentVariables - environment variables that need to be set at the execution machine before executing the component.
- requirements - the requirements section of the ADS captures the component's system requirements. This information is processed by the Execution Management Services of the grid middleware and used to match components to computing resources. Typical entries include requirements for memory, disk space, the type of WSGRAM job

manager specifying the way of executing the job by the middleware (*Fork* for execution on the machine that runs the grid middleware or *Condor* for submitting the job to a Condor cluster [130]), optional user-defined IP addresses of execution machines, operating systems and processor architectures (e.g., Motorola PowerPC, Intel i386 and higher).

- parameterLoops - parameter loops for executing iterations over loop counters. The loop element is used for sweeps or iterations over a user-defined interval with a fixed step size.
- parameterLists - parameter lists for executing iterations over a list of values (e.g. parameters or inputs). The list element facilitates a sweep over a list of numeric or nominal values. Such a list may either be provided explicitly by the user, or generated automatically by the system if a repeated execution with a list of different values is required.

provenanceInformation

The *provenanceInformation* part is used for storing information about the executed component and details on the execution. Based on this information, the users can track the executions and analyse errors in the specification of the executions The provenanceInformation is defined as a single element (see also Section A.1.4 in the Appendix). In detail, this includes the following information:

- submissionTime - the time when the job was submitted.
- completionTime - the time when the job was completed.
- schedulerStatus - the status of the scheduler after job completion.
- resultsLocation - the location of the results of the job.
- jobsStatus - the status of the job after completion.
- gramsStatus - the status of the resource manager after job completion.

3.3.3. Process of Registering and Executing Data Mining Components

Bioinformaticians who want to grid-enable their data mining components have to describe the components according to the ADS. This means that they have to create an instance of the ADS for their component. Instances of the ADS are XML documents at various levels of specification, which are passed among system components. The documents can differ in the specification of the values for the options, inputs, requirements, etc. For an execution, the ADS instance must be fully specified. This means that all non-optional values have to be set. The ADS is expressive enough to accommodate data mining components from a wide range of platforms, technologies, application domains, and sectors, and has been tested with eight different application domains [124]. Code listing 3.2 gives an abstract example. Detailed examples will be given as part of the case studies presented in Section 3.4.

Listing 3.2: An ADS instance (as pseudo code)

```
<?xml version="1.0"?>
<app:application ...>
  <generalInformation ...>
    <longDescription>...</longDescription>
    <comment .../>
  </generalInformation>
  <dataminingInformation .../>
  <executionInformation>
    <javaExecution ...>
      <interpreterArguments>...</interpreterArguments>
      <applicationRunFile.../>
    </javaExecution>
  </executionInformation>
  <applicationInformation>
    <options .../>
    <dataInputs .../>
    <dataOutputs .../>
    <requirements>
      <minMemory .../>
      <operatingSystem .../>
      <architecture ...>
        <value>...</value>
      </architecture>
    </requirements>
  </applicationInformation>
</app:application>
```

The Application Enabler, which is part of the architecture described in Section 3.3.1, is a tool that is used to create an ADS instance for a given data mining component. The process of grid-enabling data mining components is as follows:

1. **ADS instance**. Based on the ADS, the different types of information needed for a successful integration of the component in the grid are collected and formalized. The result is an instance of the ADS specifically created for the component.

2. **Registration**. After it's creation, the ADS instance is stored via the Information Services of the grid middleware in the grid registry and the executable is transferred to a storage component attached to the grid via the data services of the grid middleware.

Figure 3.4 visualizes the idea on how to grid-enable data mining components. An implementation of the Application Enabler as web application in the DataMiningGrid system will be described in Section 3.4.2.

When the component is successfully registered, it is ready to be found and executed. The tools for the execution of grid-enabled data mining components consist of the client tools Explorer, Control, Execution and Provenance (see Section 3.3.1).

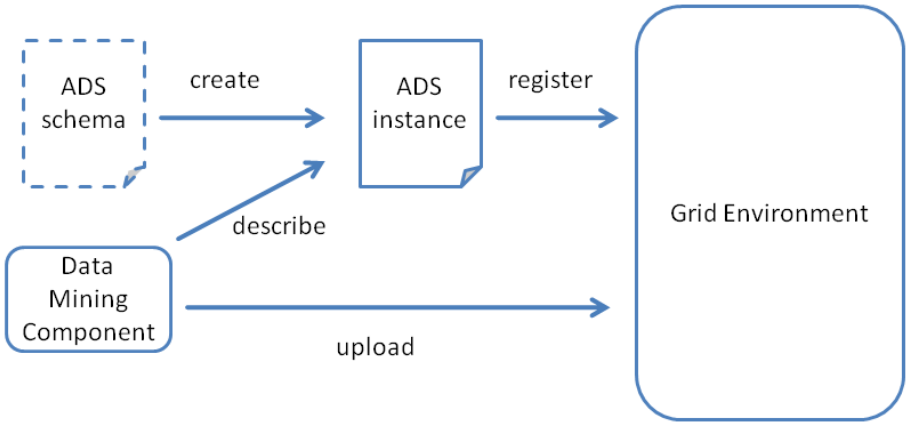

Figure 3.4.: Reusing data mining components by registering them in the grid environment with the help of the ADS.

In the following, we will describe how the ADS interacts with the grid system components on the execution of a data mining component. Figure 3.5 visualizes the interactions.

1. **Search**. With the Explorer tool the user searches the grid registry for the registered data mining component to execute by specifying search terms for metadata fields of the ADS. The Explorer tool accesses the Information Services of the grid middleware to search for information on the registered components based on the metadata from the ADS instances. By selecting a component, the respective instance of the ADS is fetched from the registry and is transferred to the Explorer tool. From there, it can be passed to the next tool (Control tool). Note that the ADS instance is not necessarily fully specified at this stage.

2. **Parameters**. The Control tool dynamically creates a GUI for specifying the options, inputs and requirements of the data mining component based on the information from the ADS instance. If the ADS instance is fully specified, which means that all of the necessary parameters are set, it can be passed to the Execution tool. If it is not yet fully specified, the user specifies the parameters and data input for the selected data mining component via a user interface. In addition, file- or parameter-sweeps can be configured. The ADS instance is now fully specified and is prepared for the execution with the grid middleware.

3. **Execution & Monitoring**. The execution tool is responsible for transforming the ADS instance into a format that can be passed to the Resource and Execution Management Services of the grid middleware. According to the specified requirements and parameters, the job is scheduled and executed on the hardware resources on the grid. During the execution, the job status can be monitored via the Execution tool, which represents a client for the execution services of the grid middleware. When the execution is finished, further information about the execution are added to the ADS instance as provenance information. This information can be visualized by the Provenance tool.

4. **Provenance**. After execution, the ADS instance and other relevant information on the execution can be inspected and stored as provenance information for a later analysis via a user interface by the Provenance tool.

An implementation of these tools as extensions to the Triana workflow environment in the DataMiningGrid system will be presented in Section 3.4.1.

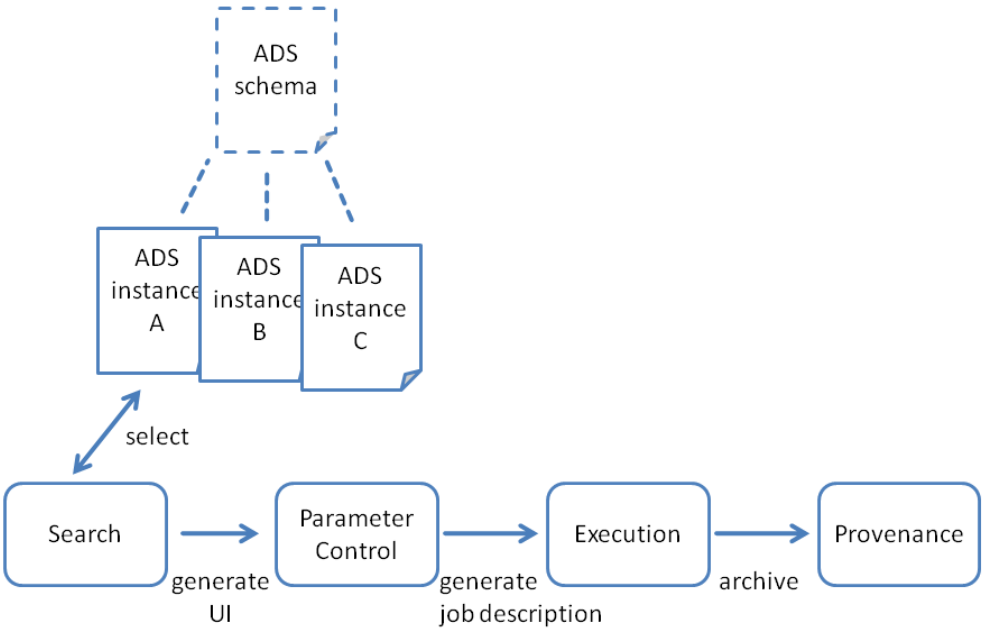

Figure 3.5.: Executing a grid-enabled data mining component with the help of the ADS in the grid environment.

3.4. Case Study DataMiningGrid

In this section we will present case studies that demonstrate the applicability of our approach for the integration of existing data mining components into grid environments. In the first case study in Section 3.4.1 we show that it is possible to implement the architecture of our approach in the context of the DataMiningGrid project. In Section 3.4.2, the second case study demonstrates that it is possible to create an user interface for grid-enabling data mining components based on the ADS which is easy to use by users who do not have knowledge on grid technology. In the third case study, we show that our approach is applicable for the integration of standard data mining components into grid environments in Section 3.4.3. The forth case study demonstrates that our approach also supports data mining scenarios based on the grid-enabled components in Section 3.4.4.

3.4.1. Implementation in the DataMiningGrid System

The DataMiningGrid system [123] was developed in the context of the DataMiningGrid project [35] (see also Section 2.4). The sourcecode of the DataMiningGrid system is partially available at sourceforge[1]. In this case study, we show how our approach on integrating data mining components into grid-based systems is implemented in the DataMiningGrid system. The case study is based on [123, 143].

The DataMiningGrid system is based on two important distributed computing standards: the Open Grid Service Architecture (OGSA [46]) and the Web Services Resource Framework (WSRF [3]). The layers of the DataMiningGrid architecture split up into the *Software and Hardware Resources Layer*, the *Globus Toolkit 4 Layer*, the *DataMiningGrid High-Level Services*, and the *DataMiningGrid Client Components*. Figure 3.6 depicts the DataMiningGrid system architecture. Generally, components in higher layers use the components from lower layers. The bottom layer is the software and hardware resources layer, which refers to the fabric layer of OGSA. The Globus Toolkit 4 layer includes some of the system's core grid middleware components and refers to the middleware layer of OGSA. The high-level services layer shows components providing central DataMiningGrid service. It refers to the application layer of OSGA but does not include client components. The Application Clients layer depicts the client-side components of the DataMiningGrid system and refers to the application layer of OGSA. The DataMiningGrid system is based on the ADS as described in Section 3.3. Please see the Appendix A.1 for the detailed XML schema definition of the ADS for the DataMiningGrid system.

In the following, we describe how our approach is implemented in the layered architecture of the DataMiningGrid system.

Software and Hardware Resources Layer.

Software resources in the DataMiningGrid system include data resources, such as database and file systems, and data mining components. Typical hardware resources include processing units, storage devices, and computer clusters. The executable files of the data mining components that are grid-enabled by our approach are included in this layer. The destination of the upload of an executable file is one of the machines of the fabric layer that provides file storage resources via GridFTP. The execution of the components take place on machines or clusters providing computation capabilities.

Globus Toolkit 4 Layer.

The Globus Toolkit 4 (GT4) layer of the DataMiningGrid architecture provides core grid middleware functionality. GT4 [48] is an open source toolkit for building grid systems provided by the Globus Alliance, which meets the requirements of OGSA and implements the WSRF (see also Section 2.2.2).

The Monitoring and Discovery System 4 (MDS4) of GT4 is essentially a system for storing and searching dynamically changing distributed XML documents that describe the grid's software and hardware resources and their status. It implements the Information

[1] `http://sourceforge.net/projects/datamininggrid/`

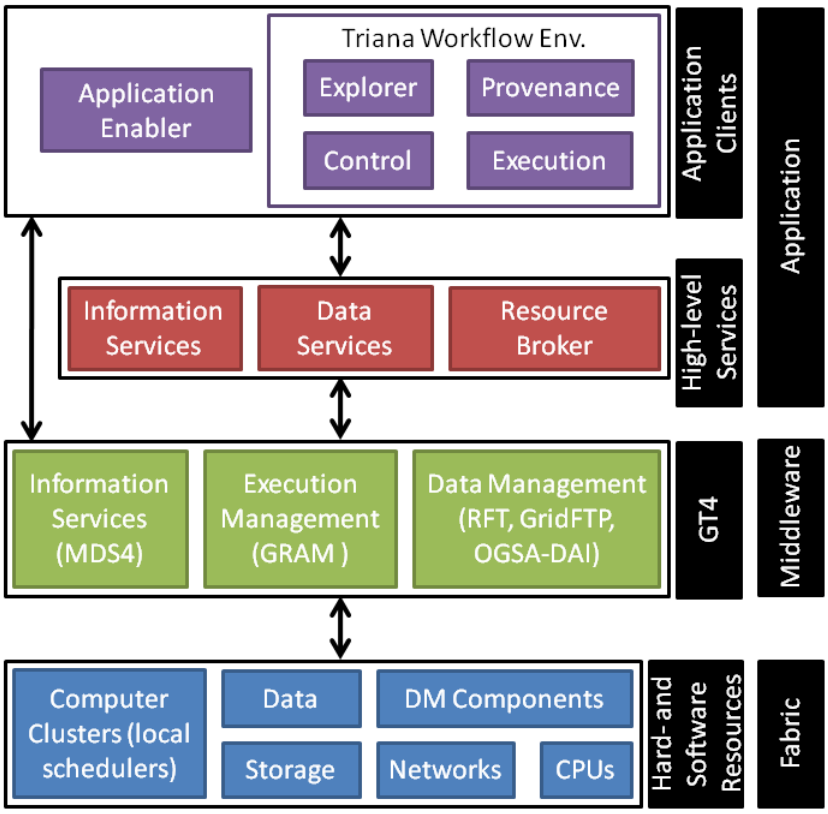

Figure 3.6.: The DataMiningGrid system architecture (based on [123]).

Services of OGSA and is used to store and retrieve the XML-based ADS instances of the grid-enabled data mining components.

The data management components of GT4 include grid file transfer functionality (GridFTP, Reliable File Transfer) and data services (OGSA-DAI) [12]. The GT4 data access and integration tools (OGSA-DAI components) provide grid-enabled access to files, relational databases, and XML databases. The GT4 data management and access components implement the Data Services of OGSA and are used to transfer the data mining components including their inputs and outputs.

The execution management tools of GT4 handle the initiation, monitoring, management, and coordination of remote computations. GT4 is used as a front-end to either single machines or computational clusters, but it does not implement a global scheduling functionality per se. GT4 provides a web service version of the Grid Resource Allocation and Management (WSGRAM) interface, which is responsible for either the job's execution on the local single machine running the GT4 middleware or for forwarding and controlling the execution via a local cluster manager such as Condor [130]. The GT4 WSGRAM component implements the Execution Management services of OGSA and is used to execute data mining components on matching computational resources based on the job descriptions generated from the ADS instances.

DataMiningGrid High-Level Services.

In the DataMiningGrid system, a job refers to the execution of a grid-enabled data mining component that needs data files as input and appropriate computing resources to be executed. The DataMiningGrid resource broker service is based on the GridBus resource broker [138], which was modified to adhere to the WSRF standard. Based on a fully specified ADS instance describing the job to be executed, the resource broker determines the list of available hardware resources from all administrative domains. From that list, it determines the best matching resources according to user requests and the requirements of the data mining component, creates a job description for GT4 from the ADS instance, and then submits the job to the WSGRAM of the selected sites. In particular, the submission process includes data and data mining component staging and setting up environmental variables. Afterwards, the resource broker monitors jobs and performs data stage-out and clean-up tasks when a job completes.

The DataMiningGrid information services are designed to support the discovery, characterization, and monitoring of various resources and services. These services create a registry via the underlying MDS4 to keep records of grid-enabled software resources and to provide information on other resources (e.g. available clusters, storage and CPU capacity or operating systems). The registry managed by the information services include the ADS instances of the grid-enabled data mining components. Users can search the registry for software resources. The resource broker also requires the registry to plan, allocate, and perform job execution.

DataMiningGrid Client Components.

The client components of the DataMiningGrid system split up into the Application Enabler web application and the workflow environment. The Application Enabler is a client for the creation and registration of ADS based component descriptions and the upload of the executable file of the component in order to integrate the component into the grid environment. It will be described later in Section 3.4.2.

The workflow environment is based on Triana [32, 128] (see also Section 2.2.3) and is designed to facilitate fine-grained user control of data mining components. It supports flexible workflow composition, parameter settings, input and output data flows, and so on. Triana was extended by workflow components which allow access to and interaction with the DataMiningGrid grid environment, especially for executing grid-enabled data mining components.

A component inside a Triana workflow is called *unit*. In the user interface, the Triana units are grouped in a tree-like structure. The units are split into several subgroups referring to their functionality, e.g., data mining components, data resources, execution, provenance and security. Figure 3.7 gives an overview of the user interface.

A workflow can be constructed by using the standard units provided by Triana and the DataMiningGrid extensions. By using and combining these units, workflows performing many different operations can be defined. A typical workflow for executing a single grid-enabled data mining component consists of four different groups of units for component exploration, data selection, component control and execution.

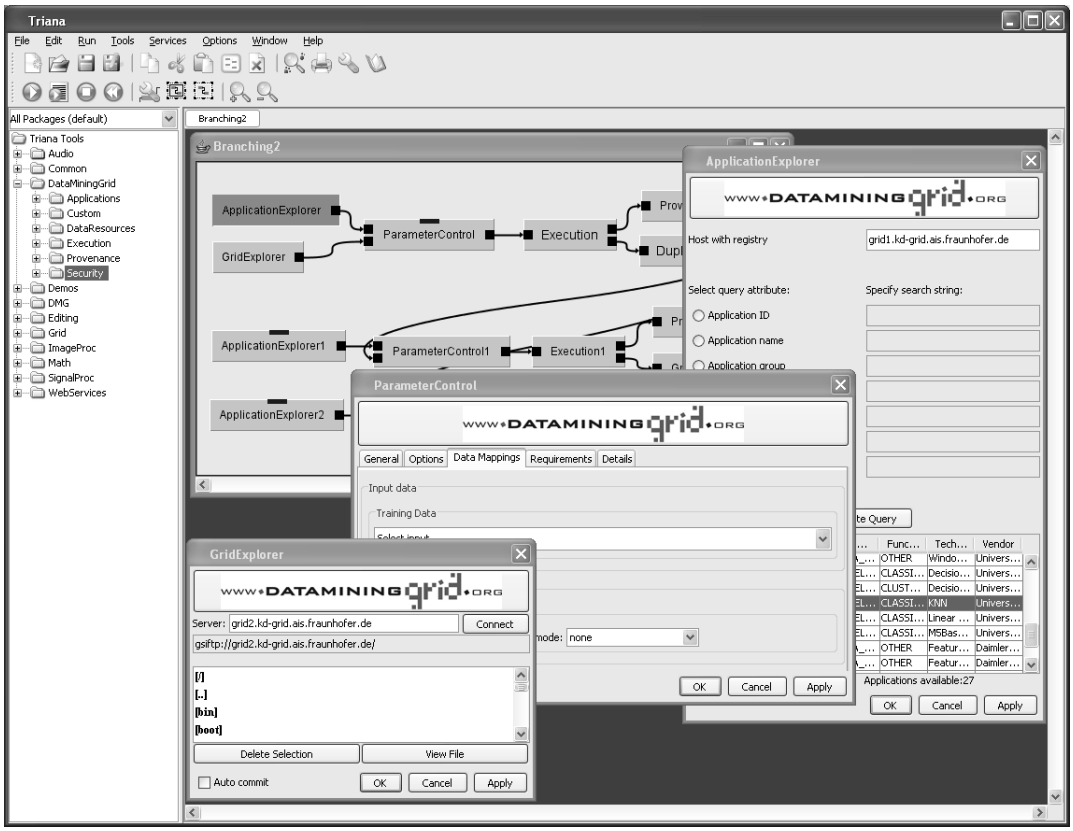

Figure 3.7.: The GUI of the extended Triana Workflow Editor and Manager — this figure shows the tree-like structure of units containing the DataMiningGrid extensions, a graphical workflow and the GUI of the DataMiningGrid units ApplicationExplorer, ParameterControl and GridExplorer (from [143]).

The following workflow units are related to our approach:

- **ApplicationExplorer**: The unit ApplicationExplorer is used to browse the grid wide registry for selecting grid-enabled data mining components.
- **LoadDescription**: Instead of browsing the grid registry and selecting an application description, this unit loads an application description directly based on a unique identifier.
- **GridURI**: This unit specifies the URI of the file in the grid which is used as input file for the selected data mining component.
- **ParameterControl**: This unit is used for the specification of the component's parameters, which are in detail the options, in- and outputs and requirements. In addition, parameter sweeps can be specified.
- **Execution**: The execution unit is used for specifying the component's output directory and the execution of the component itself.

- **Provenance**: The provenance unit shows provenance information about the component's execution. This is information about the runtime, the machines used, etc. The provenance information can be stored in an XML-file which is later used for analysis and the generation of runtime figures.
- **GridExplorer**: The grid explorer is used for browsing the component's result directory, which contains the component's output file as well as the standard out and the standard err.

The following functionality is available via Triana and the DataMiningGrid extensions [32, 122, 128, 143]:

- **Chaining** - Tasks can be concatenated inside a Triana workflow.
- **Looping** - Triana contains a *Loop unit* that controls repeated execution of a subworkflow [122, 128]. Additionally, it provides a loop-functionality when grouping units.
- **Branching** - Triana provides the functionality of workflow branching based on conditions or without conditions.
- **Shipping Algorithms** - The DataMiningGrid system allows to ship data mining components to machines attached as computational resources. Via the information services, which publish the ADS-based information on the components, the source location of the component's executable file is accessible. The executable file is then transferred to the execution machine chosen by the resource broker on each component execution. It is also possible to specify a certain machine for the execution in the ADS instance used for the execution via the ParameterControl unit. This might be useful if the data cannot be moved for some reasons.
- **Shipping Data** - Each time a component is executed in the DataMiningGrid environment, the input data for the component is transferred to a local work directory on the execution machine. If the data mining component processing the data cannot be moved, e.g., for copyright reasons, the machine where the data is located can be specified as execution machine.
- **Parameter Variation** - The DataMiningGrid system provides the possibility of using parameter sweeps, which means that a loop (for numeric values) or a list (for any data type) can be specified for each option of a data mining component. In addition, it provides a loop for the input files or directories (see Figure 3.8 for an excerpt of the GUI). By this, it is possible to submit several of jobs at the same time.
- **Parallelisation** - The DataMiningGrid system supports the parallel execution of data mining components at the same time via the execution management services of the grid middleware, e.g., by performing a parameter sweep. The jobs are then executed independently on the execution machines. In addition, Triana supports the parallel execution of workflow branches. The parallelisation of a single data mining

component is not in the scope of this chapter. However, if the component itself supports parallelisation which is compatible with the resources of the fabric layer, e.g. a computing cluster, the component can be executed in parallel.

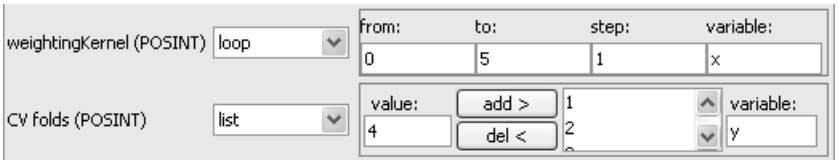

Figure 3.8.: Parameter sweep — parameter loops and lists can be specified in the GUI (from [143]).

3.4.2. User Interface for the Integration

This case study shows that it is possible to collect the information specified in the ADS in an easy and user friendly way, thus allowing users without knowledge in grid technology to grid-enable data mining components. It is based on [139]. In detail, the study describes how to use the DataMiningGrid Application Enabler web interface as a user-friendly way of writing instances of the Application Description Schema. The Application Enabler [139] is a tool which supports the creation and registration of the description and the upload of the executable of the component in order to integrate the component into the grid environment.

Using the ADS, users can create detailed descriptions of their data mining component. This assures on the one hand, that the component will run successfully, and on the other hand, that users have a better chance to find the component in the grid. Providing the description will always rely on the component developer or end-user, someone not really acquainted with the DataMiningGrid system. Thus, the manual creation of such a complex description is presumably error-prone. To avoid erroneous or incomplete descriptions and still rely on the component developer or end-user to create his own component description, we decided to provide a web application.

This web application hides XML-syntax from the user and serves him as a tool which supports the creation of the description and uploads the executable of the component. After upload to the grid, the component description will be registered in the grid registry. By this, it is published in the environment and can be found and used by other users.

The Application Enabler consists of several form-based web pages, leading the user through the whole process of creating and uploading his data mining component. For this purpose, the parts that have to be specified are divided into several functional parts:

1. General Information
2. Execution Information
3. Input Data
4. Output Data

5. Requirements

6. Upload

Each of these steps is presented to the user as a single jsp-web-page, in which specifications can be made.

General Information

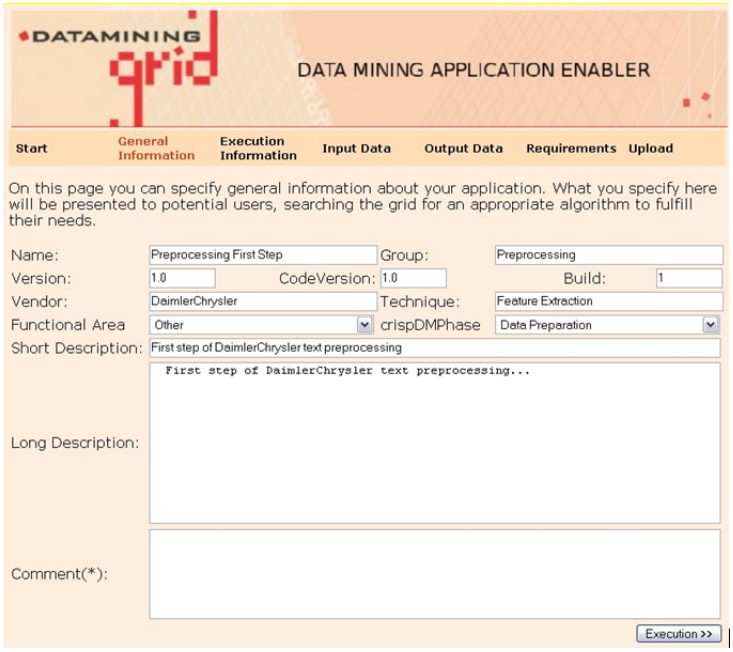

Figure 3.9.: DM Application Enabler - General Information (from [139]).

The first page displayed to the user when creating a new component description is the "General Information" page (Figure 3.9). On this page he can describe basic things about his component, like its name, its version and some description. This information is mainly used to present the component to other users in the grid.

Execution Information

The second page (Figure 3.10) is essential for the execution of the data mining component. The user can describe his executable and its options here. He can choose from four different execution types (Java, Python, BashShell, C), specify interpreter commands and give all the options his data mining component is capable of handling. Very important for a new user is that each option is described by a short tool-tip, so users can learn about the meaning of different options.

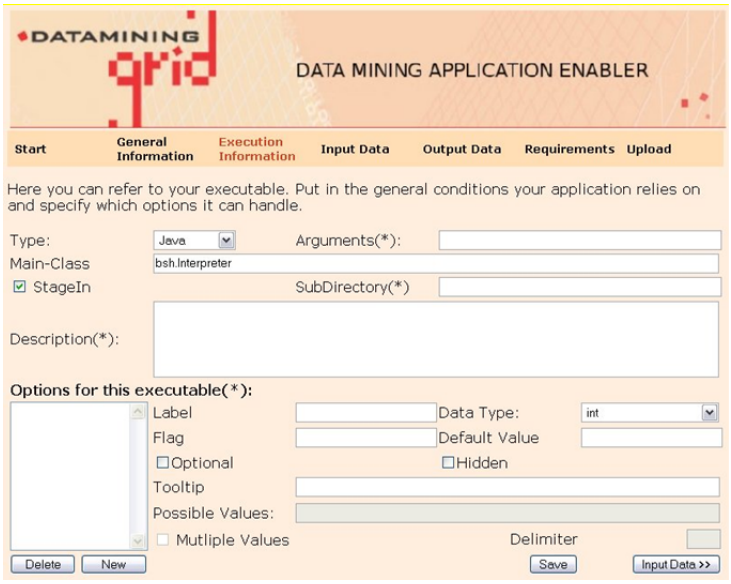

Figure 3.10.: DM Application Enabler - Execution Information (from [139]).

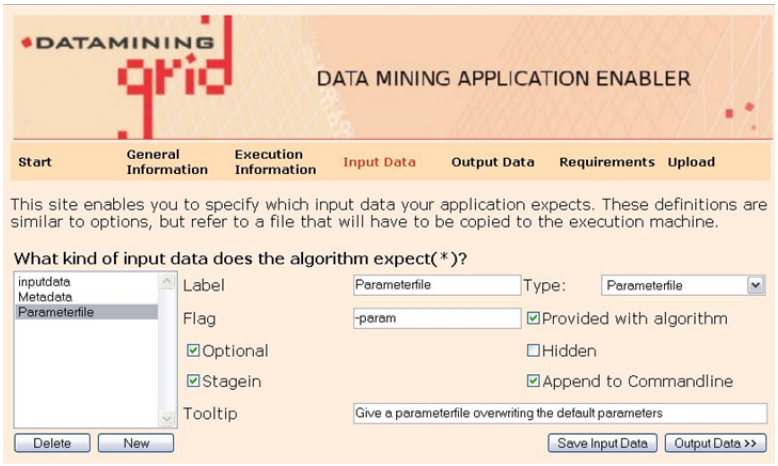

Figure 3.11.: DM Application Enabler - Input Data (from [139]).

Input Data and Output Data

These two pages (Figure 3.11 and 3.12) are very similar, as they both describe the data the component works with. Both input and output data share similar properties, like a label, data type, flag and tool-tip. The only differences are the stage-in/stage-out flags and a flag called "providedWithAlgorithm". Stage-in for input data means that the data will have to be shipped to the execution machine before the execution can start. Stage-out means shipping the specified data to a storage server, from which the user of the workflow, who doesn't know about the execution machine, can late obtain the results. The flag "providedWithAlgorithm" allows the user to upload and assign input data which is then pre-set when later using the data mining component in a workflow.

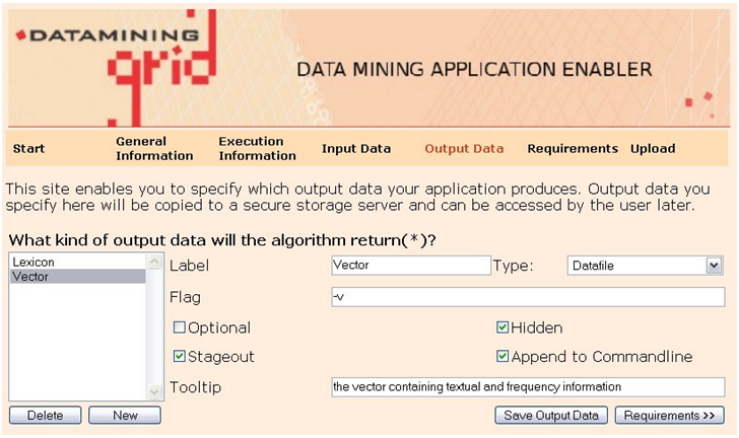

Figure 3.12.: DM Application Enabler - Output Data (from [139]).

Requirements

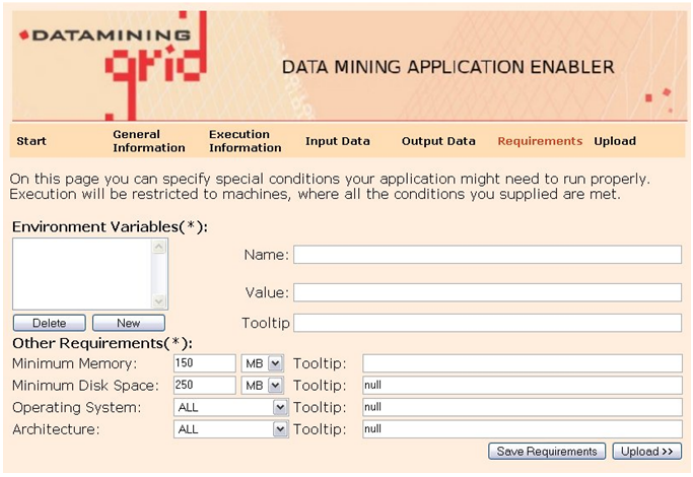

Figure 3.13.: DM Application Enabler - Requirements (from [139]).

After all of the internal conditions for the executable are specified, the user can indicate external conditions, like required environment variables or requirements applying to the execution machine (Figure 3.13). This information is especially important for the resource broker, which uses it to find the most adequate machine for execution and assures a correct environment for the component to run in.

Upload

Finally after having specified the necessary information, the files belonging to the data mining component can be uploaded (Figure 3.14). The only file that has to be uploaded commonly is the executable file. Other files are optional, unless the user specified them before. If he used the "providedWithAlgorithm" flag for some input data, he has to upload

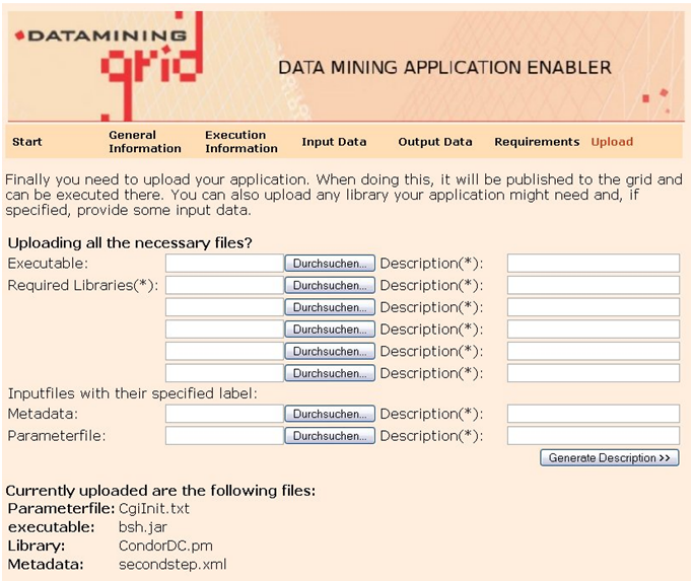

Figure 3.14.: DM Application Enabler - Upload (from [139]).

it to be able to create a valid description. Additionally he can upload required libraries the executable file depends on. These files will be copied to the execution machine before execution and are accessible to the executable file there.

Confirmation

When the user has specified all obligatory form-fields, uploaded the required files and clicked "Generate Description", an Application Description according the Application Description Schema will be created (Figure 3.15). For each new component, a folder will be created on the server. This folder will include all of the uploaded files as well as the ADS instance itself. From this folder it can be published in the grid registry.

3.4.3. Grid-enabling Weka

This case study describes the grid-enabling and execution of data mining components based on our approach presented in Section 3.3. It is based on [142, 139]. We use the Weka data mining toolkit (see Section 2.1.3) for our case study, as it represents a standard toolkit in the area of data mining. It includes a wide range of data mining components which cover the data mining methods presented in Section 2.1. The following subsections will introduce the Weka algorithms which are going to be grid-enabled as data mining components and the process of grid enabling them.

Weka

Weka [156] is a comprehensive data mining toolkit written in Java. It is available as Open Source and is in widespread use especially in the academic community. It includes components for pre-processing, classification, regression, clustering, feature selection and

Figure 3.15.: DM Application Enabler - Confirmation (from [139]).

visualization. Weka is equipped with a set of user interfaces, but the individual components can also be executed via command-line. We use Weka 3.5.5 in our experiments.

In our case study, we focus on a regression problem. We use the following Weka algorithms for our experiments:

- **K-Nearest Neighbours classifier (IBK)**: K-Nearest-Neighbours (kNN) [156] is a well-known, simple yet powerful method both for classification and regression. For predicting an unknown instance the k nearest instances according to some distance function are selected and, for regression, the target value is calculated using some possibly distance-weighted mean of the nearest neighbours. Crucial parameters are the correct choice of the distance function, the weighting function and the number of neighbours.

- **Locally Weighted Learning (LWL)**: Locally Weighted Learning [156] is an instance-based algorithm that assigns weights to instances according to the distance to the test instance. This is similar to kNN, but not limited to a fixed number of neighbours. LWL performs a linear or non-linear regression on the weighted data, using the weighted instances.

- **M5P:** M5P [156] is a tree-based algorithm. In contrast to a regression tree, which uses the mean value in the leafs of a tree for prediction, a linear model is fitted for each leaf.

Process of grid enabling

The user's component to be integrated into the grid environment has to be compliant to the definition of a component from Section 2.1.4. In our case, the jar file of the Weka distribution is the executable file for all Weka components to grid-enable.

In order to make the three components IBK, LWL and M5P available in the grid, we have to follow the procedure of grid-enabling them and create the application description files. These files are instances of the ADS and contain the following description for each component: The information in the element dataminingInformation is data mining specific metadata about the component like the component's name (which is the algorithms name) the group (Weka), the application domain (Data Mining), and the CRISP phase (Modelling). The element generalInformation contains further metadata such as version, id, a description, the upload date and so on. In the execution element we have to specify execution type (java), the main class (e.g. weka.classifiers.lazy.IBk), interpreter arguments (e.g. the maximum java heap size -Xmx1000m) and the component's executable file (path to the jar file). The element applicationInformation contains information about the options of the component (which are the options which can be specified in the Weka GUI) as well as the class attribute to predict. Each of these options is specified by data type, default value, a tool-tip, the flag, a label shown in the GUI, etc. Additionally it specifies the data input, which is a single file in the ARFF format, and the data output, which is a text file containing the textual output Weka creates. The last step is to upload the executable file and the application description files to the grid. Once the component is grid-enabled, it appears in the grid wide component registry.

Examples of the ADS instances for the Weka components IBK, LWL and M5P can be found in the Appendix (Section A.2).

Example Workflow

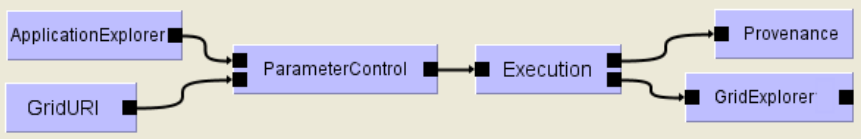

Figure 3.16.: Weka Workflow (from [142]).

In the DataMiningGrid system, Triana workflows are used to specify data mining processes. A workflow for running a grid-enabled Weka component consists, e.g., of the following units (see Figure 3.16):

- ApplicationExplorer
- GridURI
- ParameterControl
- Execution

- Provenance
- GridExplorer

When the workflow is started in Triana, the following steps are executed:

- **Step1**: The GUI of the ApplicationExplorer Unit is shown (see Figure 3.17), which allows to search the grid registry for a data mining component.
- **Step2**: The GUI of the GridURI unit is shown (see Figure 3.18), which allows the user to specify the URI of the file in the grid to be used as input.
- **Step3**: In the ParameterControl unit (see Figure 3.19), the user is able to specify the parameters of the component.
- **Step4**: During execution, the Execution unit (see Figure 3.20) shows information on the execution of the job in the grid.
- **Step5**: In the GUI of the Provenance unit (see Figure 3.21), the user can access information and statistics on the execution of the component.
- **Step6**: The GUI of the GridExplorer unit (see Figure 3.22) allows the user to browse the result directory of the executed job.

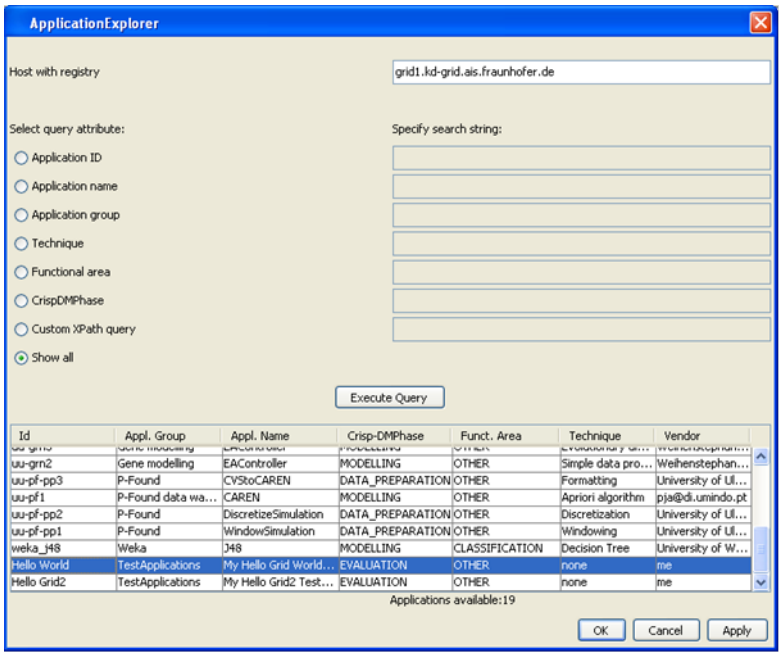

Figure 3.17.: ApplicationExplorer Triana Unit (from [139]).

Figure 3.18.: The GridURI Triana Unit (from [139]).

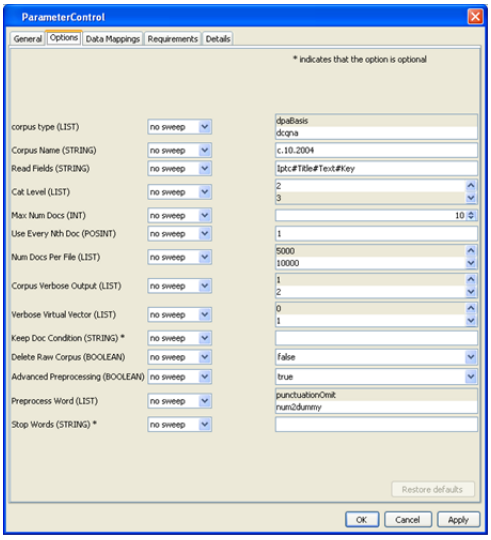

Figure 3.19.: ParameterControl Triana Unit (from [139]).

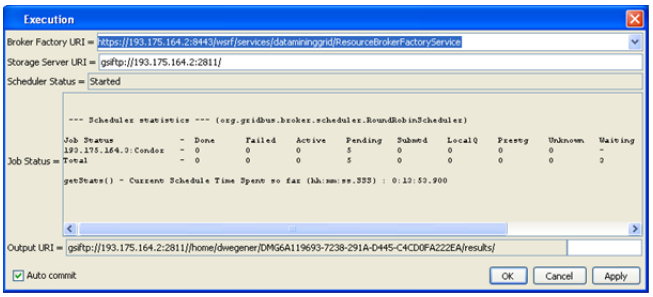

Figure 3.20.: Execution Triana Unit with 8 jobs executing (from [139]).

Runtime Experiments

In the following we will present different runtime experiments based on the grid-enabled Weka algorithms. We show two experiments with the execution of a single component with different parameter settings.

The workflow which is set up for running one of the Weka components is the one already shown in Figure 3.16. As described in the previous section, it consists of the six units LoadDescription, GridURI, ParameterControl, Execution, Provenance and GridExplorer.

In the following we describe the settings during the execution of the workflow for the Weka components. When starting the workflow, the data mining components to execute are selected. After that, the workflow passes on to the ParameterControl unit, where

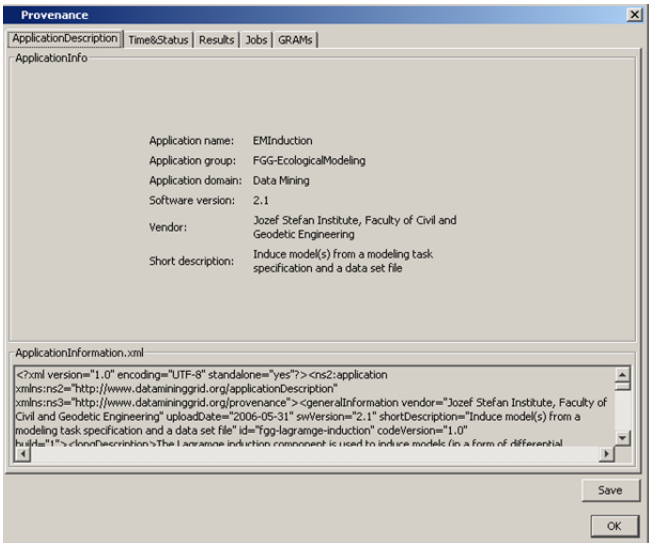

Figure 3.21.: Provenance Triana Unit displaying general information (from [139]).

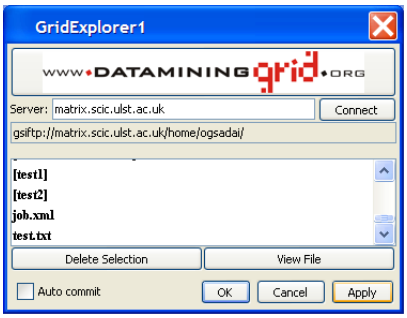

Figure 3.22.: GridExplorer Triana Unit (from [139]).

the component's parameters and options can be specified (e.g., it is specified that each job shall perform 10 fold cross-validation). We want to execute jobs which run the same component with different parameter settings, so we have to select the options on which we will perform the sweep. At the Options panel of the ParameterControl unit we can set the details for the sweep by choosing either a list or a loop for the parameter (as already shown in Figure 3.8). Out of these settings the system generates a (multi-)job description. Additionally we have to specify the component's data input. This is done by selecting the URI which was passed from the GridURI unit in the input data drop down box at the Data mappings tab. We used the dataset House(16L) from a database which was designed on the basis of data provided by US Census Bureau and is concerned with predicting the median price of houses in the region based on demographic composition and a state of housing market in the region. The dataset, which was taken from the UCI Machine Learning Repository [51], contains 22784 examples and 17 continuous attributes. This size of data is justified because we are mainly interested in measurements of the overhead caused by grid computing in the DataMiningGrid environment which becomes

Number of Machines	1	2	3	4	5	6
Max number of jobs per Machine	10	5	4	3	2	2

Table 3.1.: Job distribution for the M5P experiment.

clearer when using smaller datasets.

The next step of the workflow is the Execution unit, which submits the jobs to the resource broker. The jobs will be executed, and after all jobs are finished the Provenance unit and the GridExplorer show the provenance information about the execution and the result directory.

The test environment on which the jobs will be executed contains 2 GT4 GRAMs (Intel Pentium 4 2.40GHz, 2GB memory) and 6 Condor machines (AMD Opteron 244 1.80GHz, 4GB memory). For the evaluation we will vary the number of machines and/or the number of jobs and we will look at and compare the runtime.

Experiment M5P During the M5P experiment we submit jobs to the grid which execute the Weka M5P algorithm with different parameter settings. The execution mode is Condor, which means that all jobs are submitted to the Condor pool which are connected to the GRAMs. In this experiment we will have a fixed number of jobs and we will vary the number of machines in the grid. We generate 10 jobs in total by using a list for the option BuildRegressionTree (true/false) and a loop for the option MinNumInstances (from 2 to 10 step 2). These jobs will be submitted to 1 to 6 machines. Figure 3.23 visualizes the results of the M5P experiments. The graph shows the relation of the number of machines in the grid and the runtime of all 10 jobs. As expected, in general the runtime decreases the more machines the grid contains until the number of machines in the grid reaches the number of jobs. The jobs are distributed equally to the Condor machines. Table 3.1 shows the maximum number of jobs which one of the machines has to compute (e.g. 10 jobs on 3 machines, so 2 machines take 3 jobs and one takes 4). This explains why there is no decrease in total runtime from 5 to 6 machines.

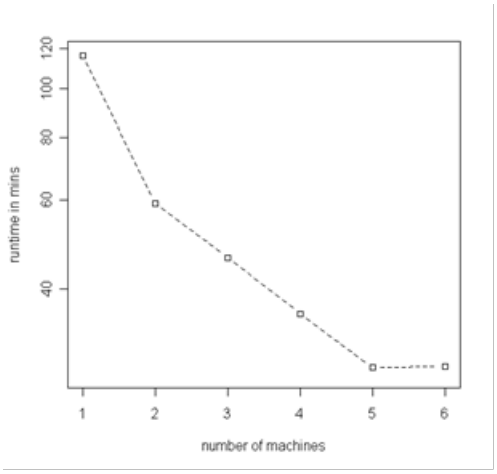

Figure 3.23.: M5P results. Results in logscale (from [142]).

Experiment IBK In the IBK experiment we will submit jobs to the grid which execute the Weka IBK algorithm. We make different experiment series, each on a different kind of machine/pool, which are compared afterwards. The jobs are generated by varying the parameter k (from 1 to maximum 16) so that we have up to 16 jobs in total. These jobs will run a) in fork mode on the Globus machine, b) on a single machine inside the Condor pool and c) on the whole grid (which consists of 6 Condor machines). In each experiment series we have a fixed number of machines and we will vary the number of jobs. The result (Figure 3.24) is as expected. At a) and b) the jobs are all executed on a single machine, the fork execution on the Globus machine has worse performance than the Condor machine. The runtime increases linear, but the Condor execution seems to be faster. This looks confusing, because the submission from the Globus machine to Condor should take some time so that the Condor execution should definitely take longer. The reason for this result is that the Globus machine has older hardware. When executing the jobs c) on 6 Condor machines, the runtime also increases linear, but in comparison to the Condor execution on a single machine the runtime decreases by a factor about 6.

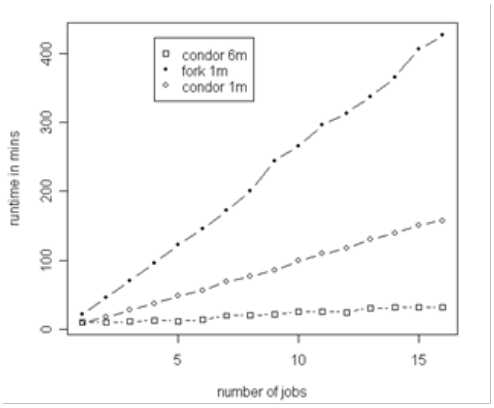

Figure 3.24.: IBK results (from [142]).

3.4.4. Grid-enabling Standardizable Parts of Data Mining Scenarios

Instead of focussing on single data mining components, we now focus on common data mining scenarios in this case study. The study is based on [143]. Typical data mining scenarios include common steps or tasks that are independent from the application scenario and thus can be seen as standardizable (see Section 2.1.3). In the following, we will describe how such scenarios are supported by our approach presented in Section 3.3.

In detail, the following 3 generic and very common data mining scenarios that can be performed with the help of our approach on grid-enabling components in the context of the DataMiningGrid system are presented:

1. Enhancing scalability by data partitioning.
2. Comparing classifier performance.
3. Parameter optimization.

	Partitioning Data	Classifier Comparison	Parameter Optimization
Chaining	yes	yes	yes
Looping	no	no	no
Branching	no	yes (InputURI and OutputFiles)	no
Shipping Algorithms	(yes)	(yes)	(yes)
Shipping Data	yes	(yes)	(yes)
Parameter Variation	yes (InputFiles)	no	yes (Options)
Parallelisation	yes (Jobs)	yes (Components)	yes (Jobs)

Table 3.2.: Operations of data mining scenarios.

Each of these scenarios can build on a broad class of data mining components. Therefore, our approach and its implementation in the DataMiningGrid emerge not as a specialist solution, but as a general solution for enhancing the scalability of a large number of data mining scenarios.

To support the scenarios described above, we have to extend the DataMiningGrid environment by grid enabling a set of new components. In detail, this case study is based on the grid-enabled Weka components from Section 3.4.3 as well as a number of helper-components (e.g. responsible for input file and result processing), which can be grid-enabled in the same way.

The scenarios are realized as Triana workflows in the DataMiningGrid system. Table 3.2 shows which workflow operations are needed to run the scenarios in the grid environment.

Partitioning Data

This section describes an experiment based on partitioning the data based on the k-NN algorithm which was grid-enabled in Section 3.4.3. The data are automatically partitioned and distributed to different machines to increase scalability. This scenario is applicable for all data mining components that are able to compute results on a subset of the data.

The scenario for this case study is the following: A large amount of data is to be analysed. The large data set is not processed on a single machine, but is split into several smaller data sets. These smaller data sets are distributed and analysed in parallel in the grid environment. The results of the analysis are combined afterwards.

In the partitioning data scenario there is a need for two additional components that process the input and output of the k-NN component in order to split the input file into several small files and to combine the results after the k-NN algorithm executions are completed. Thus, in addition to the grid-enabled k-NN Weka component, two helper components for input file splitting and result aggregation are used.

In total, three sub-workflows have to be executed. In principle, the workflow consists of 3 sub-workflows similar to the workflow of the simple experiments which are connected and executed in parallel. Each is based on the same units. The sub-workflows use the same input data and an additional Transfer unit which is responsible for copying the result files to a single directory. The first sub-workflow takes the large input file and splits it into a user-defined number of partitions. The second one is the execution of the k-NN algorithm. A file sweep is used, which results in a separate execution of the k-NN algorithm for each of the partitions in parallel. The distribution of files to the machines for the computation is done transparently to the user. The final step is to combine the results of the different executions. Figure 3.25 visualizes the workflow for the scenario.

This example demonstrates that by our approach a computationally intensive data mining task can be decomposed into simpler sub-tasks by partitioning the data and analysing them in parallel on different machines. In this case, the original component does not need to be modified.

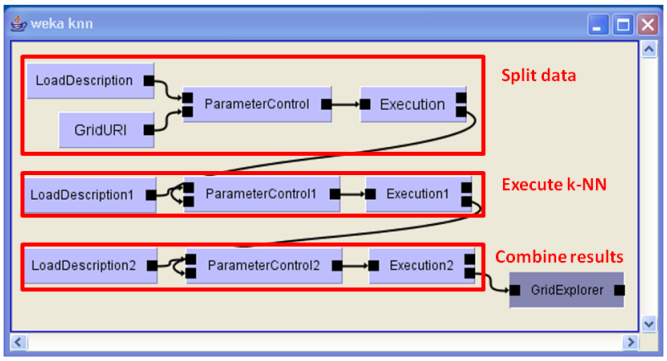

Figure 3.25.: Workflow for partitioning data.

Classifier Comparison Scenario

A typical task in data mining is to asses the performance of several classifiers on the same data-set. It is often not clear in advance which one will perform best, so the data miner has to carry out a number of experiments. In this scenario, a number of different data mining algorithms are applied to the same data set, and their predictive performance is compared. Both the data and the components that implement the algorithms are automatically distributed to different machines. Comparing classifier performance is a basic and often very time-consuming task, especially when many algorithms have to be compared and the data set is large. Performing the evaluation on a grid speeds up the evaluation, since the evaluation can be done in parallel.

The experiment consists of the execution of multiple components on the same data. The following Weka algorithms (learners) are used: M5P, IBK, and LWL (see also Section 3.4.3).

In order to compute an overall result of the learning experiment, the results of the different algorithm executions have to be combined. This is done, similar to the Data Partitioning scenario, with the help of another grid-enabled helper component.

In this experiment, three component executions of the learning algorithms are performed, each consisting of the selection of the data mining component, the control of the component parameters (all operate on the same input data) and the execution. After the learning tasks are completed, the last component combines the results of the learners to an overall result. Figure 3.26 visualizes the workflow of this scenario.

This example demonstrates that by our approach a computationally intensive data mining task can be decomposed into parallel sub-tasks by executing them in parallel workflow branches and combining the results afterwards.

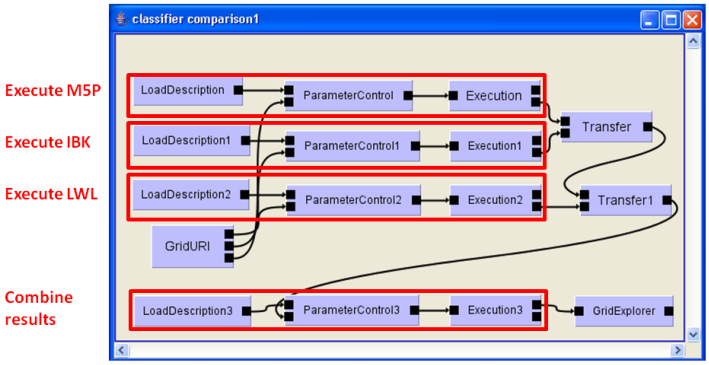

Figure 3.26.: Workflow for classifier comparison.

Parameter Optimization

A further typical scenario in data mining is parameter optimization. Some algorithms have a rich set of options to be set, and selection of an option can have a high impact on the performance of the algorithm. In many cases the best setting has to be found experimentally by systematically exploring various settings, which is a very time-consuming task.

In the scenario, both the data mining components that implement the algorithms and the data are automatically distributed. A parameter optimization experiment consists of different runs of the same algorithm on the same input data but with different parameter settings. For the parameters which are to be optimized, a range of values is used. Each change of a parameter value results in a separate run of the algorithm. The best combination of parameters is found by comparing and evaluating the results of the different runs.

Our experiment consists of the execution of the Weka LWL component for the parameter optimization task. The parameter optimization was performed for four parameters. Similar to the scenarios presented above, there was a need for a helper component in the Parameter Optimization scenario, which was responsible for combining the results of the different component executions in order to compute the result of the parameter optimization.

In the experiment, a single execution of the LWL component on the input data is performed. For this run a parameter sweep, specified at the parameter control, is used for the parameter to optimize. The parameter control component (as already visualized Figure 3.8) saves the user a lot of time in specifying the tasks, since the individual jobs are generated automatically. Depending on the number of possible values for the four parameters to be optimized, a huge amount jobs is generated. After the parameter optimization task is completed, the results are transferred into a single directory and the result files were collected and evaluated for processing the overall result of the experiment. Figure 3.27 visualizes the workflow of this scenario.

This example demonstrates that by our approach a set of computationally intensive data mining tasks can be specified easily with a simple workflow and that the tasks can be executed as jobs in parallel in the grid environment.

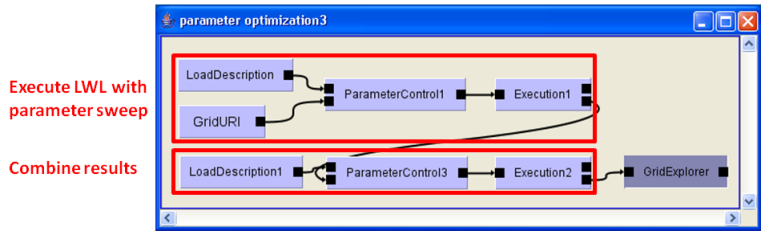

Figure 3.27.: Workflow for parameter optimization.

3.4.5. Discussion

In the first case study (Section 3.4.1) we have shown that it is possible to implement the architecture of our approach, based on the ADS, in the DataMiningGrid system.

In the case study on the Application Enabler (Section 3.4.2), we presented an implementation of the tool for grid-enabling data mining components based on a web based GUI. By this it was shown that users are able to integrate and reuse components easily. They do not need deep grid knowledge or know how in parallel processing to integrate and execute their own data mining components in a grid environment. The details of the underlying grid technology are hidden. The implementation of our approach in the DataMiningGrid system addresses the users' needs by providing a mechanism for the reuse that makes it possible to grid-enable their favourite component without writing any code, by simply providing metadata that can be specified via a web page.

In the case study on grid-enabling Weka (Section 3.4.3) we have shown how data mining components from the Weka toolkit, which was developed for a single computer environment, can be integrated into a grid environment. In detail, we integrated the Weka components IBK, LWL and M5P into the DataMiningGrid system. By this case study it was shown that common data mining components can be integrated and reused without modification of the component or the grid system. We gave an evaluation by experiments which showed that the requirements were met. The system is capable of handling even more complex scenarios, e.g. where algorithms or the data should not be moved. There is no need for the integration of new workflow operations or of new grid services. Because the scenarios can be adapted to run in the DataMiningGrid grid environment, the users of data mining benefit from all the advantages the grid technology provides, among them an user-friendly set up of experiments, a decrease in runtime or other benefits like a massive storage volume, and an integrated security concept. The evaluation of the different scenarios emphasizes the easy set up and submission of data mining tasks. As the runtime analysis shows, the flexibility of the system does not result in a big performance overhead. The runtime of the experiments depend on the speed and the number of available machines. Grid-enabled components in the DataMiningGrid system can reach a very good scalability.

The study on data mining scenarios (Section 3.4.4) demonstrated that the capability of our approach in the context of the DataMiningGrid system, extended by new grid-enabled components, is sufficient to carry out a variety of data mining scenarios. In detail, we presented how to setup scenarios for data partitioning, classifier comparison and

parameter optimization based on grid-enabled components. Additional helper components were needed just for splitting a large data set in to pieces, gathering results, or performing a simple vote.

The potential of using grid technology for data mining consists in the fact that it are simple distribution schemes that seem to give the biggest benefits in terms of performance. Our approach and its implementation in the DataMiningGrid system emerge not as a specialist technology, but as a general solution for integrating data mining components into grid environments. The system is user friendly in a way that a data miner is able to use the system - from the inclusion of new data mining components to their execution in the context of complex experiments - without any specific knowledge of the system details.

3.5. Wrap-up

In this chapter we presented an approach for the integration of existing data mining components into OGSA-based grid environments. The approach allows for integrating data mining components that have been developed as executable files in a single computer environment into grid environments. It is based on the Application Description Schema (ADS), which is an XML-based metadata schema that is used to manage user interaction with grid system components, and associated client-side components, which provide user interfaces and use the ADS for information exchange. The schema allows to grid-enable existing data mining components, to register and search for available data mining components on the grid, to match requests for job executions with suitable computational resources, and to dynamically create user interfaces. We have shown that it is possible to cover all information necessary for the execution of data mining components in OGSA-based grid environments with a single XML schema and that it is possible to create a technical system for the execution of data mining components based on data exchange via the XML schema. Our approach allows for the reuse of a wide range of components existing in the field of data mining and addresses the requirements for reuse without component or platform modification, for transparency of the grid technology, for generality and for efficiency. We implemented and evaluated our approach in the DataMiningGrid system based on GT4 grid services and extensions to the Triana workflow environment.

In addition to reusing existing data mining components, users need to compose and to develop new data mining components. In the next chapter, we will present an approach for flexible and interactive development of data mining scripts, which allow for the development and composition of data mining components.

4. Flexible and Interactive Development of Data Mining Scripts in Grid-based Systems

The goal of this chapter is to introduce an approach for flexible and interactive development of data mining scripts in grid-based systems. In the medical domain, users like bioinformaticians and biostatisticians typically interactively design and set-up their analysis solutions by combining information from different data sources and applying different methodologies to the information extracted from these data sources. In addition to reusing existing data mining components, which was discussed in Chapter 3, users need to develop new data mining components. Furthermore, users need to compose data mining components to address complex scenarios. However, users want to stick to their well known tools and do not have detailed knowledge on grid-based systems. The development of atomic components in a local environment and frequent integration into a grid environment would result in inefficiencies in the development life-cycle due to complex debugging procedures and repeated integration effort. In addition, the development might rely on data or computing resources which are not accessible in the local environment. Data mining scripts allow for the development of data mining components and their composition in a way that the components do not have to be treated as atomic components. In addition, atomic components can still be used within data mining scripts.

Thus, there is a need for an approach that allows for flexibly and interactively developing data mining scripts in grid-based systems that addressed the needs of bioinformatics users. In this chapter, we will present an approach on supporting the development of data mining based analysis processes by the integration of script-based data mining components into grid environments. Instead of developing a data mining script in a local environment and integrating it into a grid environment afterwards, the goal is to allow for an interactive development directly in the grid environment. Our approach is based on the idea of handling data mining algorithms, implemented as script-based data mining components, as parameter for a generic service. The service allows for the integration into grid environments and parallelization by automatically adding additional code snippets to the data mining scripts. In addition, we present how data mining scripts can be developed interactively using a bioinformatics tool at client side. By this, we address the need of the community to support users in enhancing and developing new data mining scripts in grid-based systems.

This chapter first introduces the requirements for the approach on flexible and interactive development of data mining scripts in grid-based systems in Section 4.1. Section 4.2 presents the layered architecture of our approach. Related work is provided in Section 4.3. Subsequently, this chapter presents our approach for supporting users in the development

of data mining scripts. First, we present how to reduce the complexity of integrating and executing data mining scripts in grid environments in Section 4.4. Instead of registering each single script separately, our method is technically based on a single grid service with complex inputs and outputs that allows for providing a data mining algorithm implemented as script as parameter. Second, we describe how to allow interactive development of data mining scripts directly in the grid environment in Section 4.5. In Section 4.6 we introduce parallelization to our approach. Then, Section 4.7 presents how our method is implemented into the ACGT system. Section 4.8 provides case studies that demonstrate the applicability of our approach for the integration and interactive development of data mining scripts and for basic script parallelization. Finally, Section 4.9 wraps up. This chapter is mainly based on [115, 119, 141, 149, 150, 151].

4.1. Requirements

In this section, we present the requirements of the biomedical community for flexible and interactive development of data mining scripts (see Section 2.1.4) in grid-based systems (see Section 2.2.2). In addition to reusing existing data mining components, which was discussed in Chapter 3, users need to enhance and adapt data mining scripts that already have been integrated into a grid environment as well as to develop new data mining scripts in such grid environments for setting up their analysis processes.

The way how bioinformaticians work in general was already described in Section 2.3. In the context of the ACGT project [34], a lot of effort has been performed on analysing the way how bioinformaticians work and on collecting requirements from bioinformaticians [132]. In detail, requirements have been collected by a scenario driven approach from interviews with users, from questionnaires, and from discussions with experts. Thus, from the bioinformatics users point of view, the following requirements have to be met [145, 132]:

Support of standard tools. Bioinformaticians and biostatisticians want to keep working with their well known tools to avoid the effort of changing the development environment and to be able to reuse their existing components and scripts [145, 132]. Thus, new scripts have to be developed based on the same tools and environments. R [103] (see also Section 2.1.3) turned out to be one of the de-facto standards for data analysis in bioinformatics. The R environment provides a broad range of state-of-the-art statistical and graphical techniques and advanced data mining methods (including comprehensive packages for linear and non-linear modelling, cluster analysis, prediction, hypothesis tests, resampling, survival analysis, time-series analysis, etc.), and is easily extensible. In particular, the associated project BioConductor [55] (see also Section 2.3.4) addresses the needs of the biomedical and biostatisticians community by very quickly providing R packages to analyse data issued from new technologies appearing in the biology laboratory. Numerous methods available as R/BioConductor packages and considered experimental a few years ago have been stabilized and became accepted standard in the analysis of high-throughput genomic data. Thus, integrating R/BioConductor in a grid-based data-analysis environment would address the needs of the community.

Quick & easy extensibility. The requirement for extensibility is very important in the context of grid-enabled data mining [119]. Especially, in order to keep track with

new scientific developments, it is crucial to be able to quickly integrate new data mining components and scripts into data analysis processes. Thus, there is a need for extensibility mechanisms that allow for using new data mining scripts without complex and time consuming integration procedures.

Interactive development. Instead of developing new scripts in a local environment and integrating these in a grid environment afterwards, users want to develop new scripts directly in the grid environment [145]. The reason for this is that they want to avoid the effort for frequent re-integration, which might be necessary when scripts are under development. In addition, the full set of data and adequate computing resources might be not available in a local environment. What is needed is an approach for interactive development of data mining scripts within the grid environment.

In addition, the following requirements, which have been already defined for our method on integrating data mining components into grid-based systems in Chapter 3, have to be met:

- **Extensibility without platform modification**: It should not be necessary to programmatically modify the grid system, neither on the server nor on the client side, to integrate a new data mining component or script.
- **Transparency**: The detailed aspects of the underlying technology of the grid system should be hidden from the developers, since they usually do not have expertise in distributed systems.
- **Efficiency**: Data mining components and scripts should be able to run in an efficient way in the grid system, based on batch job execution and the parallelization of the standard tasks parameter sweep and cross-validation (see Section 2.1.3) to save execution time and on the possibility of shipping components and scripts to the location of the data for execution to save data transfer time.

4.2. Layered Architecture

As described in Section 2.2.2, the Open Grid Services Architecture (OGSA) [46] represents a reference architecture for a service-oriented grid computing environment. It is used to address requirements such as data security, resource sharing, resource brokering and standardization and supports multi-computer environments, distributed data sources and multiple users.

OGSA is specified as a layered architecture. Each layer provides a certain set of functionalities, based on several services it includes. The functionality provided by lower layers can be used in the upper layers to provide new functionality at a higher level of abstraction.

According to OGSA, a grid computing environment consists of a fabric layer, a middleware layer and an application layer. The fabric layer includes different types of physical or logical resources that are virtualized through web-services, e.g., CPUs, memory, and disks, or licenses, contents, and OS processes. The middleware layer represents a higher level of virtualization and logical abstraction and defines the main capabilities of the grid environment. The functionality in this layer is defined by Infrastructure Services, Data Services,

Resource Management Services, Execution Management Services, Security Services, Self Management Services, and Information Services, which form the grid middleware. The application layer includes applications and other entities that use the OGSA capabilities to realize user- and domain-oriented functions and processes.

The architecture of our approach has to address the requirements presented in Section 4.1. From the requirement for *supporting standard tools* results that our approach needs to support R as basis for data mining scripts so that Bioconductor packages are also supported. This means that R has to be provided as resource in the fabric layer, that functionality for specifying and executing jobs for R scripts needs to be provided and that the client components providing the user interfaces also need to be based on R.

The demand for *extensibility* results in the following technical implications: It should not be necessary to developing further middleware or application layer services when executing a new data mining script in the grid environment. In addition, the components of the architecture should allow for quick and easy extensibility without the need for a separate process for adding new data mining scripts.

Due to the need for *transparency*, the complexity of the grid middleware needs to be hidden by providing a service in a higher layer which combines middleware functionality.

The middleware layer provides functionality for efficient resource and execution management by its Resource Management and Execution Management services. Similar to our solution presented in Chapter 3, these can be utilized to address the requirement for *efficiency*. Thus, a service of the architecture that makes use of this functionality has to be located in a layer above the middleware layer.

From the requirement for *flexibility* results that there is a need for an architectural component that provides the functionality of executing data mining scripts which can be specified and triggered in a flexible way via client components providing a user interface, such as a workflow environment or a programming language environment.

The demand for *interactivity* means that for the client components providing the user interfaces we have to focus on script-based languages such as R, which also provide a console interface for working with an interactive command line.

Resulting from that, our approach on flexible and interactive development of data mining scripts in grid environments is based on two new architectural components [151]: a generic service (GridR service), which is able to execute R-based scripts in the grid environment, and an extension of the R environment as client side component that makes use of the generic service (GridR client). The architectural components that realize our approach are located in the application layer, as they have to make extensive use of the middleware services provided by the middleware layer of OGSA. The architecture of our approach is designed as follows: The GridR service and the GridR client correspond to the application layer of OGSA. We further split the application layer into two separate layers (high-level services and clients), as the GridR client is based on the GridR service, but the GridR service can be reused in other client components. The high-level services layer contains server-side services, which are built on top of the services provided by the grid middleware to enable domain-oriented functionality. The client layer consists of the client components that provide user interfaces for the services of the high-level services layer. Figure 4.1 visualizes our approach. Thus, the GridR service is a high-level grid service which extends the grid environment by providing the functionality of executing R-based data mining

scripts in the grid environment, based on several services of the OGSA middleware layer. On the client side, the R environment is extended by the GridR client in form of an R package (see also Section 2.1.3). It allows for executing R-based data mining scripts via the GridR service in the grid environment from an R environment on a local computer on client side. Details on the GridR service will be given in Section 4.4 and on the GridR client in 4.5.

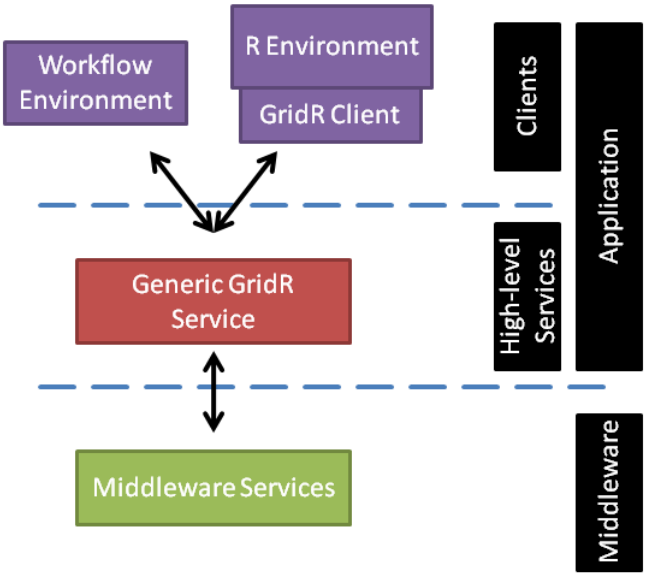

Figure 4.1.: The GridR approach: Generic high-level grid service and client side extension to R.

4.3. Related Work

Analytical services in the grid have been the focus of work in various previous projects. In Section 3.2 we presented a number of systems related to data mining in grid environments. Some of these systems, e.g. Knowledge Grid [26] or Discovery Net [9], would be also able to execute R-based data mining scripts in the grid. However, as already described all these systems have drawbacks regarding our requirements.

In the following, we will now focus on attempts to integrate R with distributed environments. In [55] a short overview over support for concurrent computation in R is given. Support for concurrent computations in R is provided by packages such as rpvm [85], rmpi [160] and snow (Simple Network Of Workstations) [111]. Rpmv and rmpi provide wrappers to the parallel programming packages parallel virtual machine (PVM) [102] and message-passing interface (MPI) [95] respectively. These approaches require explicit orchestration of message passing in the parallel execution of R scripts and are only suitable for closely-coupled homogeneous environments. In contrast to the message-passing interface (MPI) [95] or the parallel virtual machine (PVM) [102], the snow package provides a higher level of abstraction that is independent of the communication technology.

Some rudimentary support for building client-server applications that use R on the server side has been offered by toolkits like Rserve [135] or Rweb [16], but these efforts do not offer a seamless integration with grid technology. In comparison to the efforts described above our approach tries to capitalize on the advantages offered by the grid in terms of computing power and storage for the secure and scalable execution of R scripts with minimal changes to their code and no restrictions on the tasks performed.

sabreR [62] is an R package that offers a remote, web service based interface to the SABRE system. SABRE is designed to model recurrent events for a collection of individuals or cases and many other types of repeated measures, and has been extended to run on parallel processors. The differences to our approach are mainly the applicability of sabreR to communicate only with the SABRE backend rather than providing a generic interface for running all kind of R scripts and also the degree of the exploitation of the grid infrastructure. In particular the parallel server side of SABRE requires a high performance computer (HPC) to be in place and preconfigured, while GridR tasks are submitted as grid jobs, which results in a gain of additional flexibility in terms of scheduling and execution.

While remote computing clusters made available by grid technology are mostly used for the submission of independent computations, parallel computations can either be defined at the client layer or have to be implemented explicitly by the use of parallel programming libraries such as MPICH-G2 [81] or Ibis for Java [136]. Both solutions are not transparent to the user. As a consequence, in most cases computing clusters made accessible by grid technology are not considered as "parallel machines". pR [88] supports a fully transparent and automatic parallelization of R code based on MPI. However, in contrast to GridR, pR as well as the other efforts discussed in this paragraph do not offer a seamless integration with grid technology.

4.4. Developing Data Mining Scripts based on Treating Algorithms as Parameters

In the following, we will present the details on the high-level services layer of the layered architecture described in Section 4.2 and the GridR service as part of this layer. The high-level services layer sits on top of the middleware layer of the architecture. Its components make use of the functionality provided by the middleware layer. The high-level services layer itself provides functionality for client components of the client layer, which is located above the high-level services layer.

In the high-level services layer, the functionality needed for executing R-based data mining scripts in the grid environment to address the requirements of the bioinformatics community is provided by the GridR service. In detail, the functionality consists of executing R-based data mining scripts and R functions on a certain set of input data. Executions can be specified by client components in the clients layer. The execution is based on the functionality provided by the services of the middleware layer to make use of grid resources. Figure 4.2 visualizes the architecture of the GridR service approach.

The functionality is achieved by extending the grid environment by a single generic service that accepts complex inputs, which allows for providing the R-based data mining

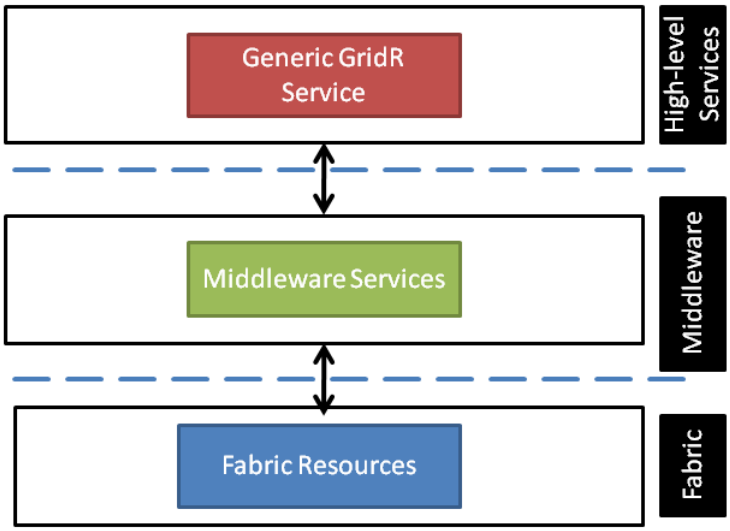

Figure 4.2.: The architecture of the GridR service approach.

scripts as input for the service. Instead of deploying many individual services with few simple parameters for each script, a single service with complex inputs including the script as parameter is used. Figure 4.3 visualizes this idea. In the following, we will present the functionality, interfaces and the detailed architecture of the GridR service.

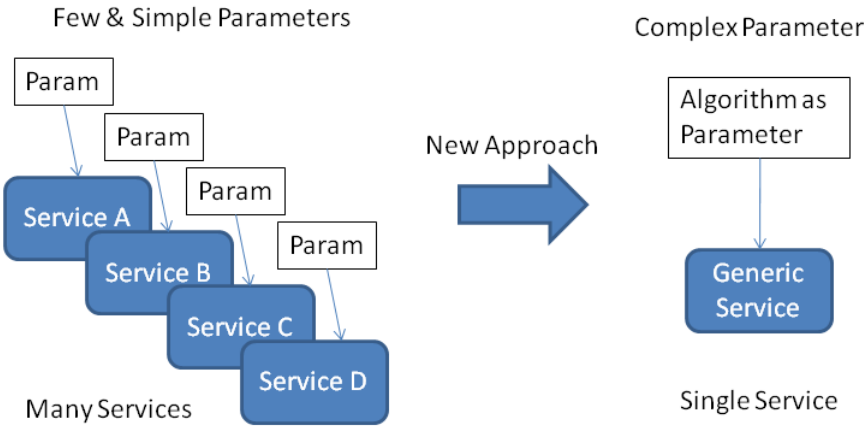

Figure 4.3.: The GridR service approach: single complex service with complex inputs instead of many services with few simple inputs.

4.4.1. Functionality, Interface and Operations

The GridR service is realized as a grid-based web service that provides the functionality of executing R scripts and R functions on given input files in the grid environment. In order to expose its functionality for the upper layers in the architecture, the interface of the GridR service includes the operations executeRScript, executeRFunction and getResult:

- executeRScript: this operation provides the functionality of executing an R script, passed as parameter, in the grid environment. References to the input data are also given as parameter. It would be also possible to pass a reference to a repository which holds the R code instead of passing the R code directly. This would allow for sharing of data mining components, as e.g. supported by myExperiment.org [58] for workflows. However, this would not affect the architecture and realization of the GridR service, as there would be only a simple additional step of resolving the reference for retrieving the script code.

- executeRFunction: this operation allows for the execution of an R function in the grid environment, based on an R workspace from an R session on client side and input files provided as references via parameters.

- getResult: this operation is used for the retrieval of the results of R script and R function executions.

Table 4.1 presents the details on the operations of the GridR service and its parameters.

4.4.2. GridR Service Architecture

In the following, we will describe the details of the architecture of the GridR service. The requirements described in Section 4.1 have impact on the layered architecture (see Section 4.2) as well as on the detailed architecture of the GridR service. The requirement for easy and quick extensibility as well as for flexibility can be addressed by passing the R based data mining components as a parameter to the generic GridR service instead of designing them as separate services in the grid environment. As the data mining components have to be executed on resources of the grid environment and as there is a demand for efficient execution, the execution has to be managed via the Resource, Execution and Data Management services of the grid middleware. In addition, the R environment has to be installed on the machines which will execute the R functions remotely, as part of the fabric layer of OGSA. Transparency can be achieved by hiding the details of the middleware services and just providing a service that aggregates the functionality of the grid middleware and provides a simple interface at a higher level of abstraction.

Thus, we design the GridR service as a high-level service, which is based on the Resource Management, Execution Management, Data Management and Information services of the grid middleware as defined by OGSA. Most of the middleware services required by the GridR service are provided, e.g., by the Globus Toolkit 4 (GT4, see also Section 2.2.2). In addition, the need for resource brokering in the grid (job scheduling and mapping of job to resources) based on the services foreseen by OGSA are, e.g., provided by tools such as Gridge [101] or GridBus Resource Broker [138].

In detail, it is necessary to ensure that the following requirements are met in order to allow for the execution of the R-based data mining scripts in an OGSA based grid environment:

- the input files of the data mining script have to be available at the machine where the script is to be executed

operation name	executeRScript
description	runs an R script on the grid
param rCodeToExecute	the R code to be executed
param inputReferences	the references of the input files (optional)
param outputNames	the names of the output files that shall remain after the execution (one is required as minimum).
param executionMachine	the IP or DNS name of a certain machine where the script is to be executed (optional)
returns	GridR job ID (later used for fetching the result)
operation name	executeRFunction
description	runs an R function on the grid
param inputReferences	the references to the input files. At least two references have to be provided, one for the file containing the R Code and one for the file containing the R workspace
param outputNames	the names of the output files that shall remain after the execution.The values have to contain at least the name of the result file that the function creates.
param executionMachine	the IP or DNS name of a certain machine where the script is to be executed (optional)
returns	GridR job ID (later used for fetching the result)
operation name	getResult
description	gets the result (references to output files) of an R execution on the grid
param gridrID	the ID of the GridR execution from that the results are to be fetched
returns	the references to the output files; null if result is not yet computed

Table 4.1.: Operations of the GridR service.

- the data mining script has to be available on the machine where the script is to be executed and the script needs to have the information on how to access its input files
- the runtime environment for the data mining script has to be available on the execution machine
- the information on how to start-up the script has to be available so that the execution on the execution machine can be started
- the result files have to be available after the execution

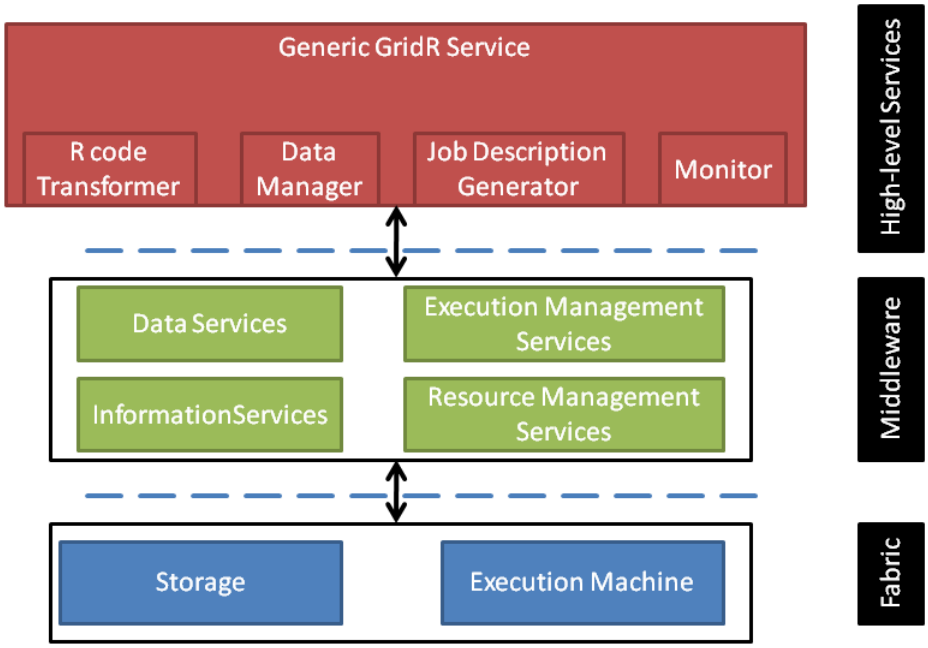

Figure 4.4.: Details of the GridR service: R code Transformator, Data Manager and Job Description Generator

To address these requirements, the GridR service is internally structured into the components R-code Transformer, Data Manager, Job Description Generator, and Monitor. Figure 4.4 visualizes the details of the GridR service, its internal components and the services of the OGSA architecture it depends on. In the following, we will describe the details of the internal components.

R-code Transformer: The R-code transformer is responsible for enabling the grid integration by adding code snippets to the script code to be processed. It attaches additional code sections to the R code for the handling of the input and output files. Figure 4.5 visualizes this idea. Code Listing 4.1 presents some code excerpt as example.

Data Manager: The Data Manager is responsible for handling the transfer of the input and output files as well as the transfer of the code of the script. It interfaces with the Data Management services of the grid middleware layer.

Job Description Generator: The Job Description Generator is responsible for creating the description on how the script has to be executed by the Resource and Execution Management services of the grid middleware. The job description generator submits the job via the grid middleware. As result, it gets a unique job ID which can later be used for querying the status of the job.

Monitor: The monitor is responsible for checking the status of the execution. With the help of the unique job ID it can get the status information from the grid middleware services. Once the execution is finished, it holds references to the result files.

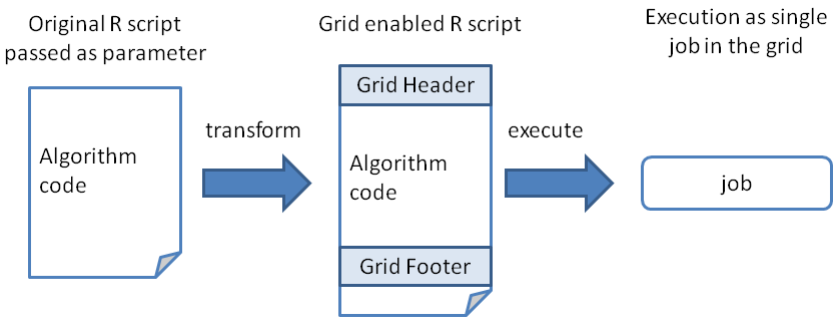

Figure 4.5.: Adding code snippets to the algorithm.

Listing 4.1: Example: R code attached to the script header for unzipping input files, loading data and passing the names of the input files to the script.

```
# handling of GridR input data
inputs <- c("inputfile1.csv","inputfile2.zip")
gridr.input <- list()
for(n in inputs){
  splits <- strsplit(n,".",fixed=TRUE)[[1]]
  if(length(splits) >= 2){
    suffix <- splits[length(splits)]
  }
  if(suffix == "csv"){
    gridr.input[[length(gridr.input)+1]] <- read.csv(n)
  }
  else if(suffix == "zip"){
    paste("unzip -o ", n )
    system(paste("unzip -o ", n ))
    gridr.input[[length(gridr.input)+1]] <- n
  }
  else{
    gridr.input[[length(gridr.input)+1]] <- n
  }
}
```

In addition, R needs to be available as Resource in the fabric layer. This is necessary, as the R execution environment is needed for executing the R scripts and functions. This also holds for the extensions to the R environment in form of R packages such as Bioconductor (see Section 2.3.4). The Information services of the grid middleware need to be configured so that they include the information on the R environment and the installed packages when exposing the capabilities of the execution machines attached to the fabric layer. This information can then be used for matching job requests to available resources by the resource brokering components of the grid middleware.

In the following, we present details on how the components and services interact during the execution of R scripts and functions.

4.4.3. Execution of Scripts and Functions

The GridR service provides an asynchronous way of executing R scripts and functions in the grid. The process of the execution of R scripts and functions with the GridR service is visualized in Figure 4.6.

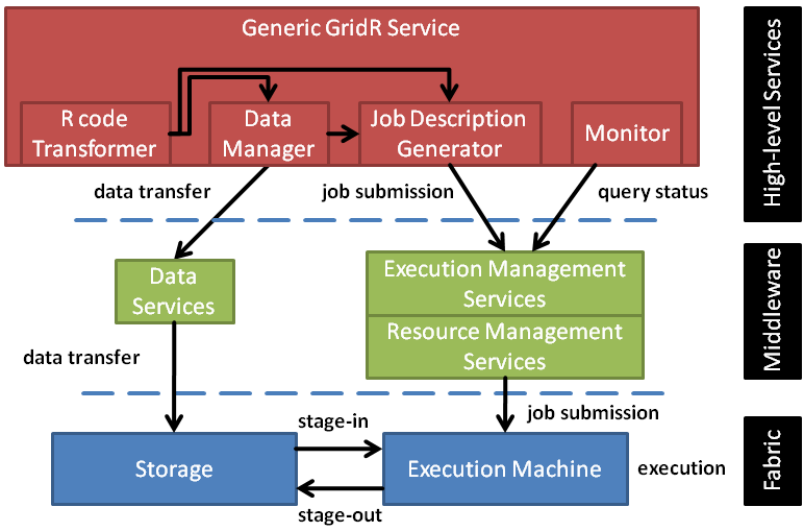

Figure 4.6.: Process of execution with the GridR service.

In detail, the process of executing an R script is realized as follows: The GridR service is called by a component from the client layer. The following steps are executed after the service call to the operation *rScriptExecution*:

- A unique GridR job ID is created for identifying and referencing the execution.
- An asynchronous server side process is started on the machine where the GridR service is deployed.
- The GridR job ID is returned as result of the service call.

The server side process consists of the following steps:

- The R-code Transformer transforms the R script's code provided as parameter by attaching a header for the handling of the input and output files.
- The R-code Transformer parses the script for R libraries needed for processing it, specified by the *library(..)* command of R.
- The Data Manager creates a local file containing the transformed R code.
- The Data Manager creates a directory in the grid storage via the Data Management service of the grid middleware.
- The Data Manager uploads the code file to the grid storage.
- The Data Manager creates references for the output files in the grid storage via the Data Management service, which can be used for retrieving the outputs after the execution finishes. This includes the reference to the R logging file ".Rout".
- The Job Manager creates a job description for the Execution Management services of the grid middleware. In detail, this includes the following specifications:
  - set the requirements for R libraries as constraint (for Information services).
  - set application to run to "R" including optional information on the actual R version as constraint (for Information services).
  - set the executable to "R CMD BATCH".
  - set R code file name as argument.
  - set input file references as files to stage in.
  - set output file references as files to stage out. This includes the R logging file name.
- The Job Manager submits the job to the Execution Management service of the grid middleware. The resulting job ID can be later used for querying the status of the job.

The Execution Management service performs the following steps:

- It maps the job including its requirements to available machines based on the information provided by the Information services of the grid middleware.
- It submits the job for execution on an execution machine in the grid.

On the execution machine, the following process is executed:

- Stage-in: during the stage-in phase of the job, the input files are transferred from grid storage to a local working directory on the execution machine according to the job description.

- Execution: In the execution phase, R is started in batch mode according to the command *R CMD BATCH* with the R code file as parameter, as specified in the job description. By the script header, the input files are mapped to the inputs needed by the R script and the output files are named so that they match with the job description.
- Stage-out: When the execution finished, during the stage-out phase of the job the output files, including the log file, are transferred to the grid storage according to the job description.

For retrieving the result of an execution, a client can perform a service call to the operation *getResult*:

- by the GridR job ID, connect to the GridR server process that manages the execution.
- by job ID, query the Execution Management service for the job status.
- depending on the status of the job, return information on the job's status, on errors or return references to the output files.

For executing an R function, the process is similar to the process of executing R scripts described above. The only differences for a service call to the operation *rFunctionExecution* are the following specifications of input and output files:

- input file: a file with R code, which is read, transformed, and then stored in the grid storage. The code includes a command for reading the workspace file.
- input file: a file containing parameters needed for the function to execute, prior stored as R workspace file in an R session on client side.
- output file: a workspace file containing the output parameter of the function, later read as R workspace file in an R session on client side.

Summing up, the GridR service is part of the high-level services layer, and is based on services of the grid middle-ware layer of OGSA. It enables the use of R as a data mining component by a service which can be seamlessly integrated with the other data access and data analysis services, e.g. in a workflow.

4.5. Developing Data Mining Scripts by Interactive and Rapid Prototyping

In Section 4.4, we described the high-level services layer of the architecture presented in Section 4.2 and its component GridR service. Now, we focus on the client layer. The client layer sits on top of the high-level services layer and includes client side components providing user interfaces for services in the lower layer. Such clients are, e.g., components of workflow environments or programming language components. Here, we will focus on a

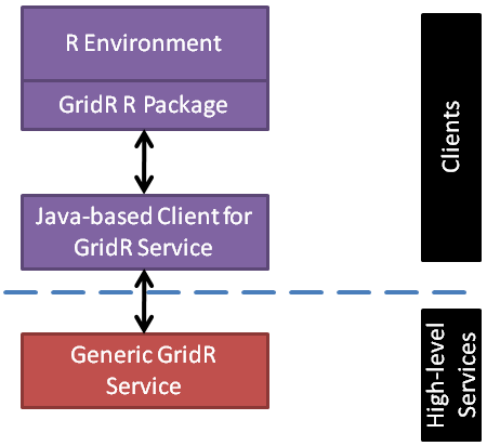

Figure 4.7.: GridR client architecture.

certain client component - the GridR client - which is part of our approach on interactive development of data mining scripts in the grid.

The GridR client provides the functionality of submitting the execution of R scripts and R functions from an R session on a client side machine to the GridR service, thus allowing for an execution in the grid environment.

The GridR client is structured into the GridR R package, which represents an extension to the R environment, and a Java-based client to the GridR service, which is attached to the R package. Figure 4.7 depicts the GridR client in the architecture of our approach. The R package includes a set of functions which make the GridR client's functionality available for the users. These functions are bound to a Java-based client for the GridR web service. The R to Java communication is performed via the RJava package [105].

The GridR client is used as an interactive tool and programming language interface for accessing grid resources and in particular for the remote execution of R code in the grid. More specifically, the task of executing R scripts and functions is submitted via the GridR service as a job to a remote machine in a grid environment by interactively calling the respective functions from an R session.

4.5.1. GridR Client Architecture

The R package part of the GridR client is internally structured around the components "RemoteExecution" and "Locking". The RemoteExecution component is responsible for the execution of R code as a job in the grid environment by transforming the R code to execute into a set of files and submitting it for execution to the GridR service (see Section 4.4) via a Java-based web service client.

The GridR service is responsible for creating a job description file in the respective job description language and for submitting the job to the Resource Management service of the grid. During this process, the Locking component takes care of the consistency of files and variables in the R session. Figure 4.8 visualizes the detailed architecture of the GridR client.

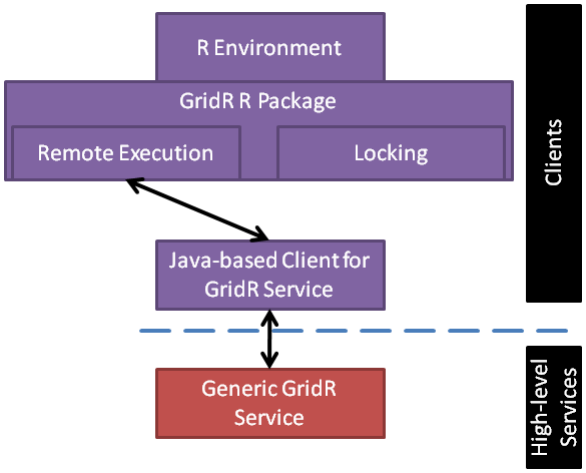

Figure 4.8.: Details of the GridR client: RemoteExecution and Locking.

In practice grid access is performed through the call of predefined R functions collected in an R package. The client side components are based on usual R functions (see Table 4.2 and Table 4.3), so no changes in the core R implementation are necessary.

Name	Action
grid.apply	Performs a remote execution of R functions; waits (callback) or sets a lock (grid.lock).
grid.check	Checks if the internal structure contains all variables and functions to execute a function f remotely. If not, the missing variables are returned.
grid.consistency	Checks if the internal structure has errors or if there are local files without a running job.
grid.init	Initialization of the grid environment.
grid.isLocked	Checks if a variable has a lock.

Table 4.2.: Public functions of the GridR client.

The functions are based on the following R functionalities:

- callbacks - functions that are executed automatically by R after the user has issued a command (this is used when checking for results).
- active bindings - a variable is replaced by a function call which is handling the locking system and allows working interactively with that variable. When the variable is read, the predefined function is called and returns the value associated to the variable (or an error code if the variable is locked). When a value is assigned to the variable, the function is called with the value as parameter for storage in an internal structure.

Name	Action
grid.callback	Performs a callback.
grid.waitForAll	Performs grid.callback for all jobs.
grid.readLock	Read-lock a variable.
grid.writeLock	Write-lock a variable.
grid.unLock	Removes the lock from grid variable.
grid.unlockAll	Unlocking of all grid variables.
grid.catchObjectNotFoundError	Error handling by parsing of error messages.

Table 4.3.: Internal functions of the GridR client.

- parsing of script and error code - checking for missing values, variables and functions in the code which is executed remotely as well as for errors in the result objects.

Thus, users can make use of the functionality of execution in the grid environment in a transparent way by passing the functions to be executed in the grid as input to one of those predefined functions (grid.apply) in their local code. Figure 4.9 shows the execution of a simple sum function with the GridR client. Details on the steps of execution are described in the following section.

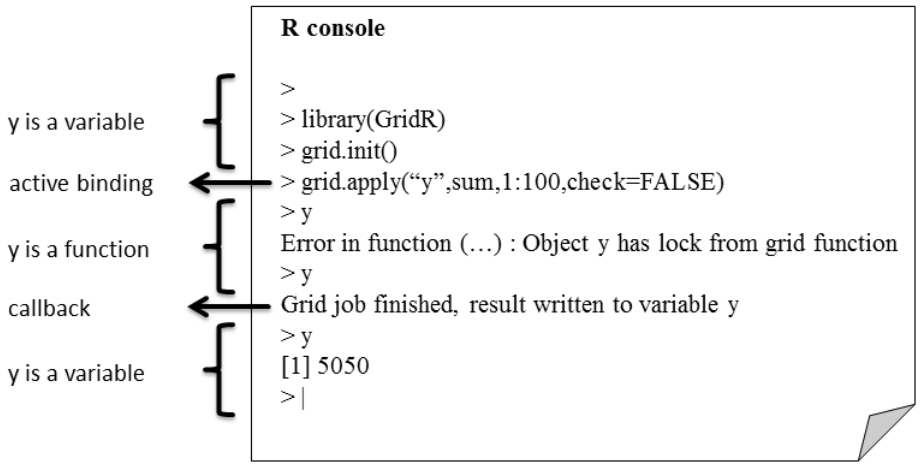

Figure 4.9.: Simple GridR Client Code Example.

4.5.2. Process of Execution

As described in Section 2.1.3, the difference between a script and a function execution is as follows: an R script can be executed stand alone, which means that it does not depend on objects from an active R session. During the execution of a script, a new R session is created, which is closed after the execution finished. R scripts might only depend on input files that have to be accessible from the script's code. In contrast, R functions depend on the workspace of an active R session. This means, as the computation of the R function is

supposed to be performed on a different machine than the client machine where the user's R session is running, that the workspace has to be made available for reuse on the remote execution machine in the grid.

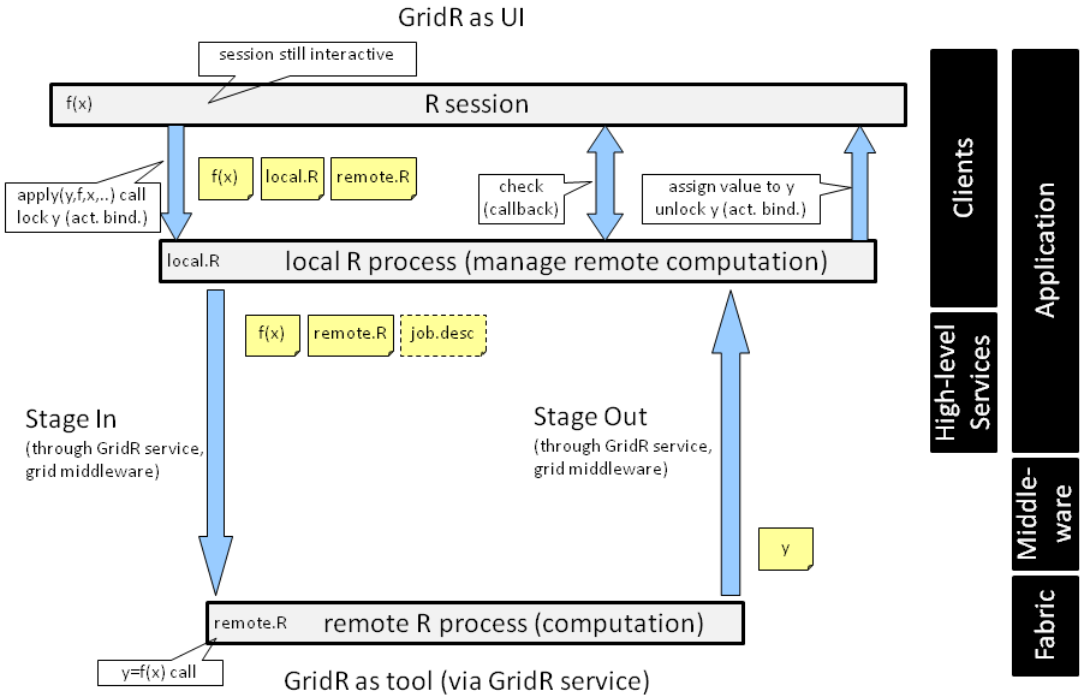

Figure 4.10.: Steps of executing an R function with the GridR client.

In the following the process of executing a single R function in a grid environment is described. Figure 4.10 visualizes the steps of executing an R function with the GridR client.

- **Function loading**. The GridR client functions are loaded from the GridR package into the workspace of the R client.
- **Grid initialization**. The grid environment is initialized by calling the function grid.init. This function sets the configuration for the GridR service (service location). To avoid the specification of settings on each submission, a configuration file can be used to pre-configure the settings.
- **Code writing**. The R code which is to be executed in the grid is written and wrapped as single R function in the local R environment.
- **Grid submission**. The grid.apply function is called, which launches the process of submission. At first, the function to be executed in the grid and the needed parameters are written into a file (uniqueID-fx). Then, the R script which is executed on the machines of the fabric layer of the grid environment on job startup is generated (uniqueID-RemoteScript.R). Next, an R file is created, which specifies the

"workflow" which is performed on the client side (uniqueID-LocalScript.R). After that, the R client executes this script, which results in connecting to the GridR service. During the GridR stage-in phase, all files needed for the job submission (uniqueID-fx and uniqueID-RemoteScript.R) are uploaded to the grid and a grid job is generated via the GridR service. The grid Execution Management services then take care of staging-in files to the execution machine, launching the processing and managing the handling of the results. During the remote execution, the created R script (uniqueID-RemoteScript.R) is executed on the remote machine, which reads the parameters and the function from uniqueID-fx, executes y=f(x) and writes the result or any errors into a result file (uniqueID-y.dat).

- **Waiting for result**. There are two ways of waiting for the result, a blocking one and a non-blocking one (specified by wait=TRUE or FALSE). While the remote execution is active and the R client waits for result (by checking if the file y.dat is created), the variable y is locked, or - if the blocking mode is used - the R session on client side is blocked until the result is available.

- **Result processing**. When the result file (uniqueID-y.dat) was created on the remote machine, it is, together with the other result files, transferred back to the client during the stage-out phase. In the GridR client, this file is loaded. The exit status is checked and - if the job was successful - the value is assigned to y and the variable is unlocked.

The execution of a script is more simple than the execution of a function, as there is no need to transfer the objects of the workspace on which the function depends. This means that there is no need for the file uniqueID-fx containing the function and workspace objects.

4.6. Parallel Processing

In this section, we introduce parallelization to our approach for flexible and interactive development of data mining scripts. The need for parallelization of R scripts results from the requirements collected in Section 4.1. As the architecture of our approach is structured into different layers, parallelization can be introduced in different ways, depending on what kind of functionality the certain layers provide (see also Figure 4.11):

1. **Parallelization in the client layer**. The GridR client can submit several script or function executions in parallel to the GridR service.

2. **Parallelization in the high-level service layer**. If the grid middleware provides services that allow for batch job submission, the GridR service can submit jobs for parallel execution.

3. **Parallelization in the fabric layer**. If resources such as cluster management systems are part of in the fabric layer, these can be used for executing jobs in parallel by the GridR client from within an R script or function.

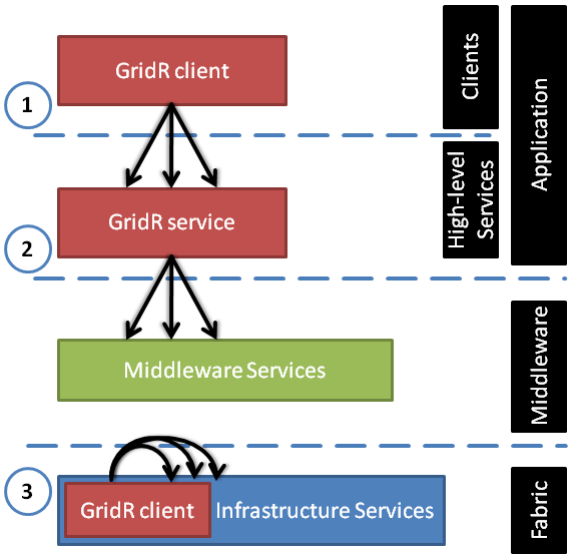

Figure 4.11.: Parallelization introduced in different layers of the architecture.

In the following, we first present how parallelization can be enabled in the high-level services layer (2) by extending the code transformation and job description generation mechanisms of the GridR service. Second, we introduce parallelization into the GridR client, which allows for parallelization in the client layer (1) as well as in the fabric layer (3).

4.6.1. Parallelization with the GridR Service

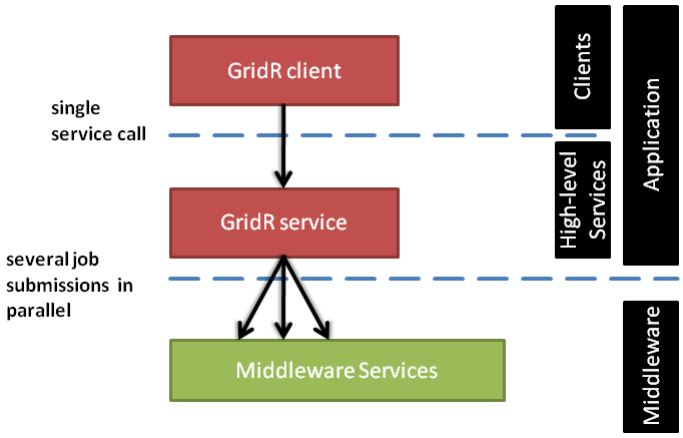

Figure 4.12.: Parallelization with the GridR service.

The motivation for the parallelization of R code is that a large set of advanced bio-statistics tasks are computationally very intensive but have a structure which is trivially parallelizable, e.g. there are elements of calculations in R scripts that can be run indepen-

dently of each other. Developers of R scripts are ultimately best placed to know which parts of the script can benefit from parallelization.

The parallelization with the GridR service is visualized in Figure 4.12. The GridR service can split the script into parallel or non-parallel sections that are or are not to be run in parallel as grid jobs with the same approach on automatically adding or transforming code snippets of the script that is provided as parameter, as described in the previous section. Figure 4.13 visualizes this idea.

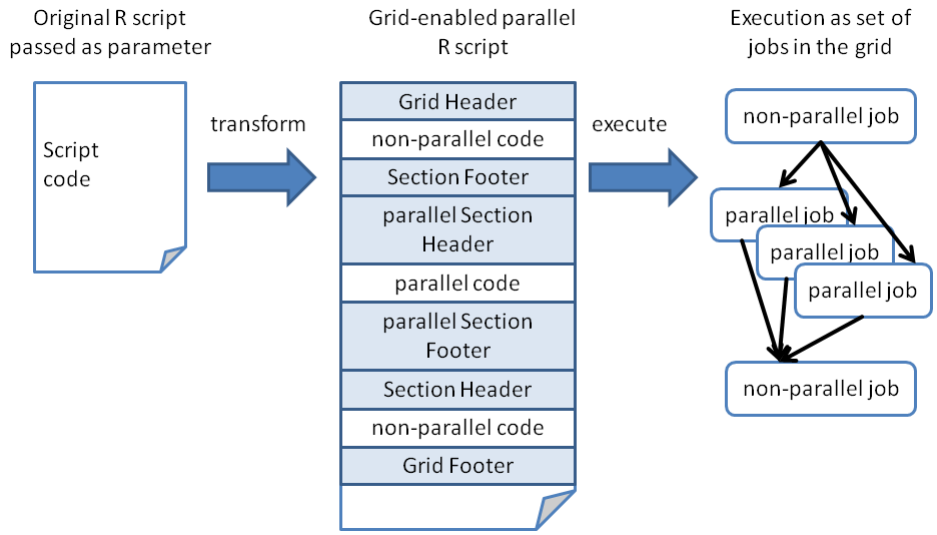

Figure 4.13.: Splitting the script code and adding code snippets.

Hence, the R script is no longer executed as a whole by a single grid job. Instead, it is executed as a specified number of sub-tasks resulting from splitting the original code into the respective number of smaller parallel and non-parallel sections. The individual script sections use files to interface with the previous or subsequent sections. The parallel GridR service thus attaches a header and/or a footer to each section for interfacing with other parts of the script, in a similar way as for the handling of input and output of the non-parallel version of the GridR service as described in Section 4.4.2. Headers and footers are responsible for loading data as R objects from the files that are staged-in to the execution machines and storing the data as R objects to the files that are staged-out from the execution machines.

In detail, the following code snippets are created for non-parallel sections:

- Header - loads the full workspace saved by the previous non-parallel section and the output object stored by the previous parallel sections
- Footer - saves the full workspace for the following non-parallel section and the objects needed as input by the following parallel sections.

For parallel sections, the following snippets are created:

- Header - loads the objects stored by the previous non-parallel section and the index

variable specifying the iteration number, which allows users to take iteration-specific actions.

- Footer - saves the output objects for the following non-parallel section.

Technically, the GridR service translates the parallelization information into a complex job description representing the workflow to be executed and submits it as job to the Resource Management services. For data transfer performance reasons, all non-parallel parts of the script can be executed on the same machine, which means that only the subsets of R objects required to perform the parallel sections have to actually be transferred to other machines for computation.

However, the sections of the code that can be computed in parallel have to be marked in a way that the information that is needed for the GridR service to set-up a parallel execution is fully specified. The developer of a parallel GridR script is offered a "directive"-like mechanism for the annotation of the parts of the script that can run in parallel. With the help of these annotations, the information needed can be specified.

Internally, a preprocessing component of the GridR service parses the script for extracting the user specified annotated information needed for the submission of grid jobs through the Resource Management services. This information includes the specification of the *inputs* (including functions if they are user defined) and *outputs* of the parallel sections and of an index variable which can be used to identify specific parallel computations. In addition, the *degree* of parallelization (the number of parallel tasks) and two pointers marking where parallel sections of the code *start* and *end* are also determined during the preprocessing phase. In order to avoid having a different code version for standalone and GridR parallel execution of the script, the directives needed to parse the R code and make a parallelized version of it are passed to GridR as R comments.

The number of iterations has to be known before execution. This number is used to spawn a corresponding number of grid jobs, each executing a single computation. This mechanism is illustrated with the example in Code Listing 4.2, which shows the generation of 3 parallel jobs that compute a single iteration of the for-loop each.

Listing 4.2: Example of a parallel GridR script.

```
double <- function (x) {2*x}
add <- function(a,b) {a+b}

x=1

result1=double(x)
result2<-list()

#GRIDR-PARALLEL-START; index=i; degree=3; input=result1,result2,add;
    skipNextLines=1
for (i in 1:3) {
        result2[i] = add(result1,i)
}
```

```
#GRIDR-PARALLEL-END; output=result2; skipPrevLines=1

result3=0
for (i in 1:length(result2)) {
       result3=result3 +result2[[i]]
}
```

The script is split into 3 sections including headers and footers. Code Listing 4.3 shows the first non-parallel section, Listing 4.4 the parallel section, and Listing 4.5 the final non-parallel section.

Listing 4.3: First non-parallel section of the parallel GridR script.

```
# GridR script code section 0
double <- function (x) {2*x}
add <- function(a,b) {a+b}
x=1
result1=double(x)
result2<-list()

# GridR section 0 footer code
save.image("codeSection_0_workspace.RData")
save(result1,result2,add,
   file="codeSection_0_outputForParallelSection_1.RData")
```

Listing 4.4: Parallel section of the parallel GridR script.

```
# GridR section 1 header code
load(file="codeSection_0_outputForParallelSection_1.RData")

# GridR script code parallel section 1
       result2[i] = add(result1,i)

# GridR section 1 footer code
save(result2,file="codeSection_1-0_outputForNonParallelSection_2.RData")
   # 1-0, 1-1 and 1-2 for 3 parallel jobs
```

Listing 4.5: Last non-parallel section of the parallel GridR script.

```
# GridR section 2 header code
load("codeSection_0_workspace.RData")
result2_tmp <- list()
load(file="codeSection_1-0_outputForNonParallelSection_2.RData")
result2_tmp[[1]] = result2[[1]]
load(file="codeSection_1-1_outputForNonParallelSection_2.RData")
```

```
result2_tmp[[2]] = result2[[2]]
load(file="codeSection_1-2_outputForNonParallelSection_2.RData")
result2_tmp[[3]] = result2[[3]]
result2[[1]] = result2_tmp[[1]]
result2[[2]] = result2_tmp[[2]]
result2[[3]] = result2_tmp[[3]]

# GridR script code section 2
result3=0
for (i in 1:length(result2)) {
       result3=result3 +result2[[i]]
}
```

4.6.2. Parallelization with the GridR Client

The GridR R package allows for client-side parallelization as well as for server-side parallelization, which will be both presented in the following sections.

Client side parallelization

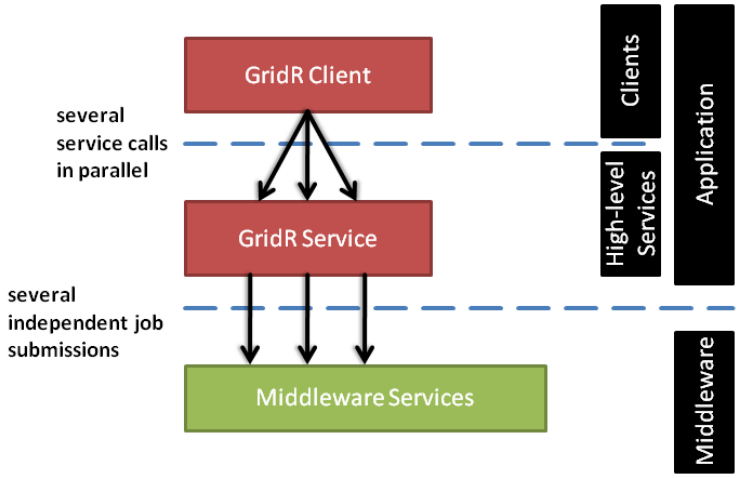

Figure 4.14.: Parallelization with the GridR client on client side.

Parallelism can be expressed in the code at the client layer in GridR. The client side parallelization with the GridR client is visualized in Figure 4.14. It provides the functionality of submitting R code packaged in a function as a job for execution in a grid environment. The jobs are submitted in the background to the GridR service, which forwards them to the Resource Management service. The Resource Management service is then responsible for executing them in parallel.

The following code example demonstrates a cross-validation task that is computed in parallel:

```
library(GridR)
grid.init()
X <- as.data.frame(array(rnorm(300),c(100,3)))
Y <- X[,1]+X[,2]-2*X[,3]+rnorm(nrow(X))
YX <- cbind(Y,X)

n_folds=10
err <-0
vars=paste("tmp",1:n_folds,sep="")
for(i in 1:n_folds) {
  grid.apply(vars[i],cv_single_fold,i,n_folds,X,Y,YX, wait=FALSE)
}

grid.waitForResult(vars)
for(i in 1:n_folds) {
  err <- err+get(vars[i])
}
err=err/n_folds

cv_single_fold <- function(i,n_fold,X,Y,YX) {
  n <- nrow(X)
  YXtrain <- YX[which(1:n %% n_fold != i-1),]
  YXtest <- YX[which(1:n %% n_fold == i-1),]
  m <- lm(Y~V1+V2+V3,YXtrain)
  p <- predict.lm(m,YXtest)
  err <- mean((p-YXtest[,1])^2)
  return(err)
}
```

Figure 4.15 visualizes the approach of client side parallelization with the example. Parallelization is achieved by generating a number of jobs at the client side, submitting them via the GridR service to a resource management system and processing them on the execution machines in parallel. In detail, on the client side an R script is executed that submits a number of jobs (the execution of a single cross-validation fold inside a for-loop) via the GridR service to the Resource Management service, which then are executed, e.g. by the GT4 grid middleware or the Condor cluster management system. The jobs run in parallel in the execution environment and compute the same R function (with different parameter settings).

Server side parallelization

In addition to client side parallelization, as described in the previous section, the GridR client allows for server side parallelization. The server side parallelization with the GridR client is visualized in Figure 4.16. Instead of submitting a number of independent tasks that are computed in parallel, a single job is submitted that is processed as parallel job. As in a sequential execution, the client just submits an R script *calc()* containing the algorithm

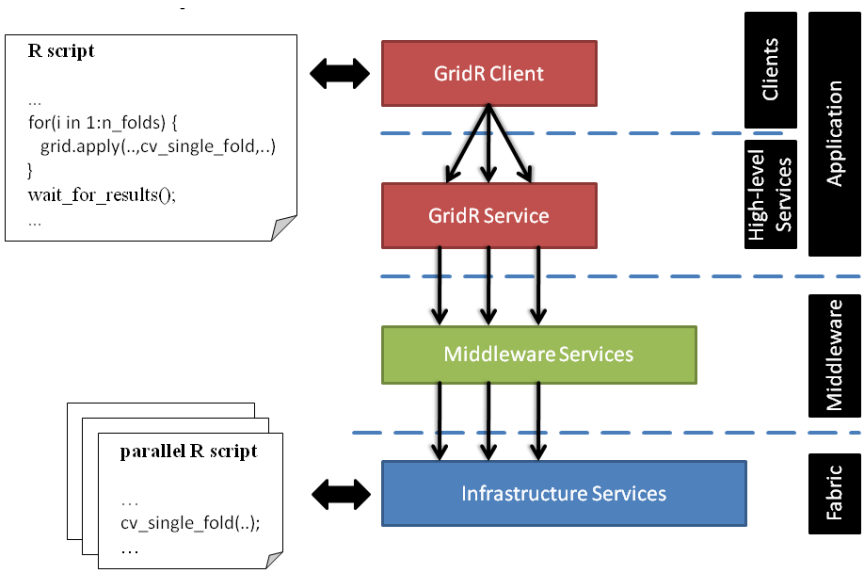

Figure 4.15.: Example of GridR client side parallelization with the GridR package.

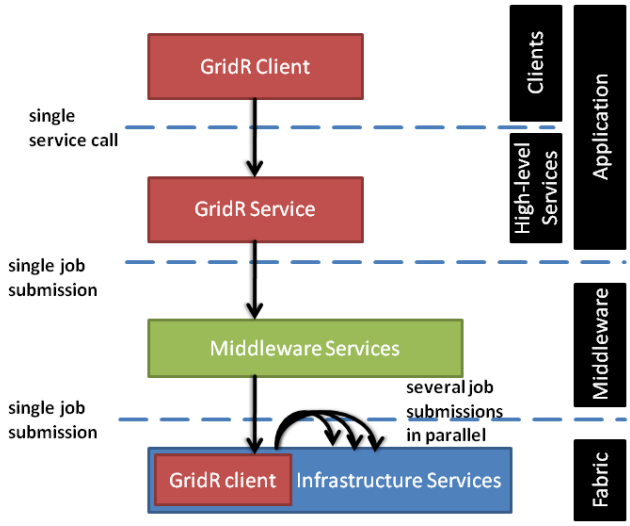

Figure 4.16.: Parallelization with the GridR client on server side.

that performs the analysis as a single job, but wrapped by another function *wrap()*. In addition, this function also has a parameter indicating the desired parallelization degree that has to be specified. The script containing the wrapper function which is executed on the server side is responsible for the generation of sub-jobs as well as the result aggregation. Generation of sub-jobs means that from the execution process on the server side another job-submission process is started. This can be achieved easily by instantiating the GridR client on the server side. This instance of the GridR client submits a number of sub-jobs to a resource management system which process the actual computation tasks on the execution machines.

With this approach of instantiating the GridR client on the server side, where the server takes the role of a client during the processing of jobs, users are enabled to set up complex parallel processes flexibly by even involving different resource management systems or environments (grid, cluster) as well as to make use of recursion. With this approach it is still possible to modify the R script specifying the algorithm independently from the parallelization because the latter is packaged into a separate wrapper function. It would be even possible to set up a collection of predefined wrapper scripts for some tasks (e.g. result aggregation) which would allow combining wrappers with different algorithms depending on the scenario.

Figure 4.17 visualizes the approach of server side parallelization as an example of submitting a single job from the client side to the grid and submitting a number of parallel sub-jobs from the grid execution environment to a cluster. Instead of generating a number of jobs at the client side, the client submits just a single job via the GridR service to the Resource Management service, e.g. the GT4 grid middleware. In the execution environment, the job is processed and executes the wrapper function, which internally starts launching a number of sub-jobs that are computed in parallel to another resource manager, e.g. the Condor cluster management system. By this, the functionality for client side parallelization can be used on server side in the execution environment even in a recursive way.

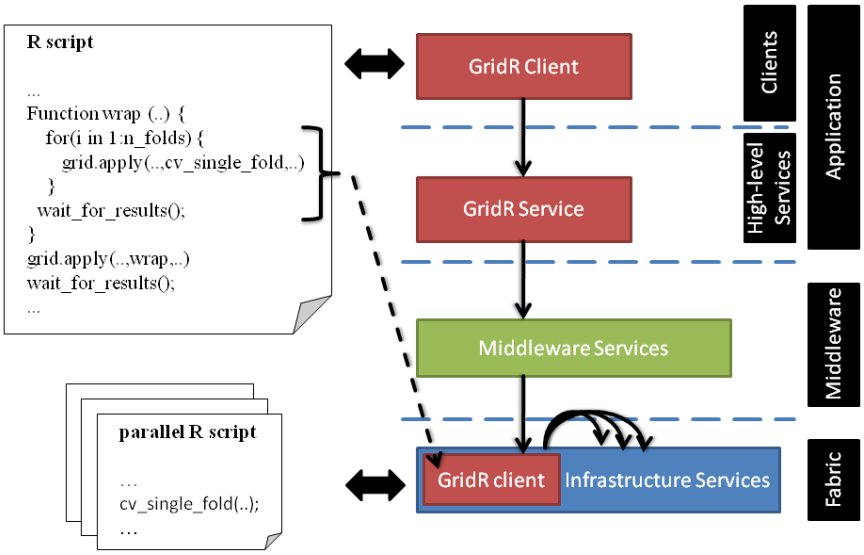

Figure 4.17.: Example of GridR server side parallelization with the GridR service.

4.6.3. Discussion

We have introduced parallelization into 3 different layers of the architecture. The decision on which layer to chose for the parallelization of a given scenario depends on the configuration and structure of the grid environment and the requirements of the scenario. Parallelization in the high-level service layer can be specified by adding annotations to

the R code to parallelize. However, the degree of the parallelization has to be known in advance, which might not always be possible for a given scenario. Parallelization in the client layer is achieved by applying functions with the GridR client, but this results in more communication between the client and the grid environment. Thus, the performance of the system might be reduced. If the application of functions with the GridR client is wrapped in an R script executed at server side, parallelization can be moved to the fabric layer. However, this is not possible if the grid jobs are executed independently in an isolated environment, which might be the case for highly secured grid systems.

4.7. Reference Implementation in ACGT

In this section, we show how the presented architecture of our approach on flexible and interactive development of data mining scripts is applied in practice. The Advancing Clinico-Genomic Clinical Trials on Cancer (ACGT) project [34], which was presented in Section 2.4.2, aimed at providing an open environment for supporting clinical trials and related research through the use of grid-enabled tools and infrastructure. The ACGT system [133, 150] serves as an example for a complex distributed data analysis environment in the medical domain that is designed according to OGSA (see Section 2.2.2). Our approach has been implemented in the context of the ACGT project. In the following, we will give an overview over the technical architecture of the ACGT system and describe how the GridR service and the GridR client have been integrated.

4.7.1. ACGT Technical Architecture

The ACGT project [34] aimed at addressing the requirements and needs of the biomedical community by providing a secured, integrated data management and data mining environment in support of large multi-centric clinical trials. The ACGT system represents a complex architecture supporting clinical trials based on grid technology.

From the technological point of view, ACGT offers a modular environment in which new data processing and data mining services can be integrated as plug-ins as they become available. ACGT also provides a framework for semantic integration of data sources (e.g., clinical databases) and data mining tools, through the use of a specifically developed ontology and of a semantic mediator. In the ACGT framework various elements of the data mining environment can be integrated into complex analysis pipelines through the ACGT workflow editor and enactor, itself embedded in a user-friendly web portal.

In terms of the technology infrastructure the ACGT platform is based on the following state-of-the-art technologies and standards: Service Oriented Architectures, the Grid and the Semantic Web.

The ACGT requirements in terms of data management, efficient utilization of computational resources, and security are matched by the adoption of a grid infrastructure. The adopted architecture builds upon the grid fabric and it is further enhanced by the deployment of web services and semantic web technologies. These technologies, although initially separated, are currently converging in a complementary way and the ACGT platform is a case which demonstrates the feasibility and the added-value of such a convergence and integration.

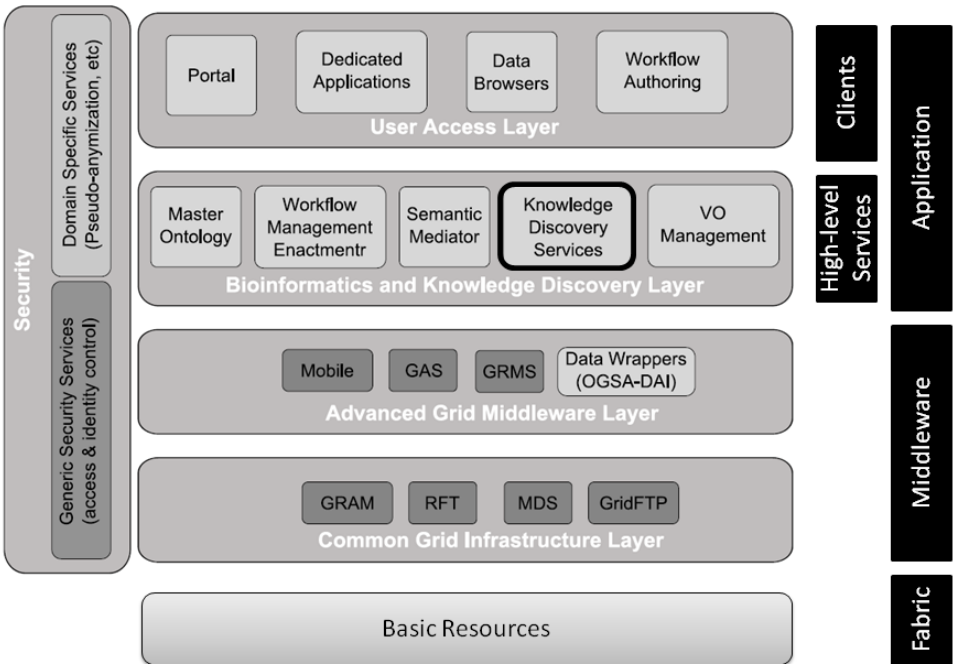

Figure 4.18.: The ACGT layered architecture (based on [150]).

The OGSA-based architecture that was adopted for ACGT is shown in greater details in Figure 4.18. A layered approach has been followed for providing different levels of abstraction and a classification of functionality into groups of homologous software entities [133]. In this approach the security services and components are pervasive throughout ACGT in order to facilitate user management, trust bindings and access-rights management and enforcement. The grid and domain-specific security mechanism used ensure the proper implementation of security requirements like pseudonymization and anonymization.

Apart from the security requirements, the grid infrastructure and other services are located in the first two layers: the Common Grid Layer and the Advanced Grid Middleware Layer (bottom of Figure 4.18). In particular the Grid Authorization Service (GAS) is the central entity for managing access authorization rules in the context of a Virtual Organization.

The middle layer consists of the Bioinformatics and Knowledge Discovery Services. These services are the "workhorse" of ACGT and the corresponding layer is where the majority of ACGT specific services lie. The set of services that have been developed in this context can be roughly classified as follows:

- Data access services. These services are responsible for the retrieval of data shared in the context of a clinical trial. This category includes the Data Wrappers which are adapters for existing clinical and imaging databases exposing database contents to other ACGT components, Microarray services that provide access to BASE repositories [42], and finally Semantic Mediator Services that offer uniform access to distributed and heterogeneous clinical and biomedical databases.

- Services for the Semantics-aware use of the platform. In this category, the Ontology Services provide a conceptualization of the domain, by the mean of the Master Ontology for Cancer [23], and for constructing complex queries for the Mediator Services based on the SPARQL query language [5].

- Service Enactment, which includes the basic grid mechanisms used for the submission and execution of jobs in the grid, and the higher-level Workflow Enactment Services that support the management and execution of complex biomedical workflows.

- Metadata Repository services, which ensure the persistence and proper management of the metadata description of the services available to the users.

- Data Analysis and **Knowledge Discovery Services**, which are a number of data mining and knowledge discovery tools and services that fulfil the data analysis requirements of ACGT, with GridR [151] as one of the most prominent tools. The approaches presented in this chapter are part of the Knowledge Discovery Services and their integration into the overall architecture.

Figure 4.19.: The ACGT Portal.

The upper layer is where the user-accessed services, such as the portal and the visualization tools, reside. The portal is the main entry point for ACGT (shown in Figure 4.19). The majority of the tasks a user needs to do in the context of a clinical trial, such as the analysis of biological data, execution of services, enactment of published workflows, training and learning activities, etc., are supported by the ACGT portal and its portlets. Through the portal the users are able to design new scientific workflows that combine data retrieval and data analysis tasks in order to implement a scenario. These activities are supported by the ACGT workflow editor (shown in Figure 4.20), a web-based drawing and workflow designing tool. The workflow editor provides a graphical browser which facilitates the discovery of the existing ACGT services and their composition. The ACGT workflow editor is an example of the "Software as a Service" approach, eliminating the need to install and run the application on the users' desktop machines, and also featuring better integration with the grid and Service Oriented Architecture of the platform (e.g. the server side execution of the workflows, a central repository of all the workflows to better support sharing, etc.). Users can develop complex analysis pipelines by interconnecting services registered in the metadata repository, and store them as new services for later reuse.

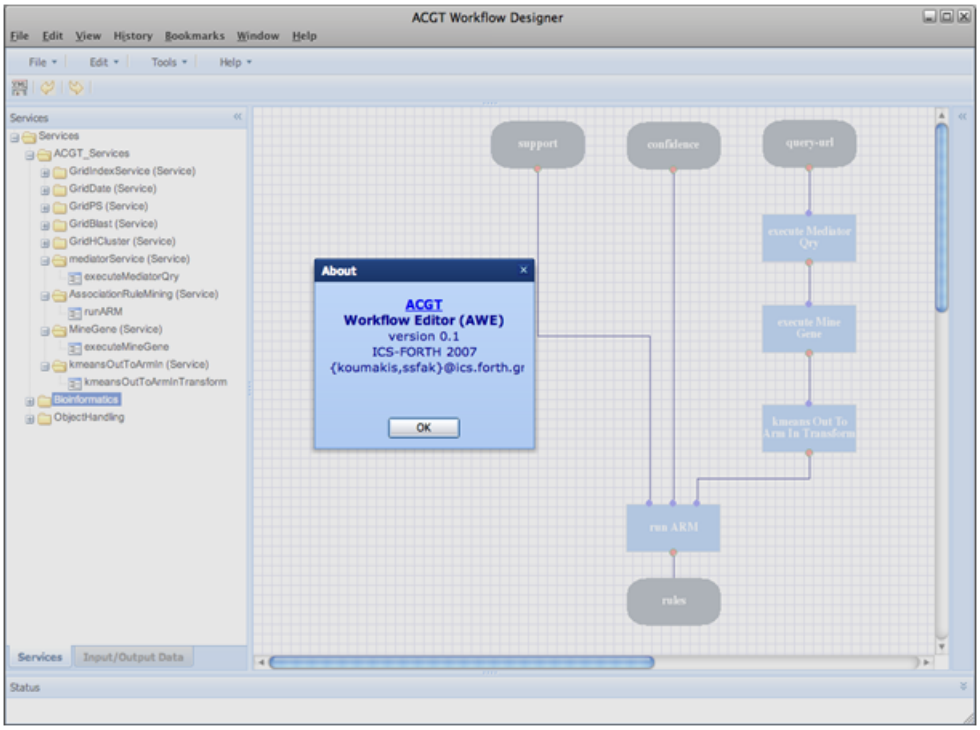

Figure 4.20.: The ACGT Workflow Editor.

4.7.2. Integration of GridR into the ACGT Grid Environment

GridR [151] is an analysis tool based on the statistical environment R that allows using the collection of methodologies available as R packages in a grid environment. The aim

of GridR in ACGT is to provide a powerful framework for the analysis of clinico-genomic trials involving large amount of data (e.g. microarray-based clinical trials). In ACGT GridR plays a dual role: On one hand it can be used interactively, giving the users access to the ACGT environment, on the other hand it is deployed as a data-analysis service exposing a web service interface for the execution of scripts incorporated in scientific processes and workflows. Thus, scripts can be used as atomic components in processes, which will be addressed in Chapter 5.

GridR in ACGT consists of the GridR service, the GridR client and the GridR R environment. The GridR R environment is basically a standard R environment with some additional packages installed, which is installed on the machines which will execute the R functions remotely.

The GridR client is provided as an R programming language interface that supports the access to services of the ACGT environment. This means that R users and developers have access to distributed resources in a transparent fashion, the complexity of the grid being hidden from the user. The sourcecode of the GridR client is partially published at the official R package repository CRAN[1] [37].

In the ACGT platform, the GridR service is implemented as a GSI-secured grid service based on the Globus Toolkit 4 libraries [48] and on the Gridge Toolkit [101]. In detail, the GridR service includes clients to the Gridge Data Management System (DMS, a virtual file system for data organization in grid environments), and to the Gridge Grid Resource Management System (GRMS, which is responsible for grid resource management and scheduling of grid jobs). The interface to the DMS is based on files. This implies that all input and output data have to be passed to and from GridR by physical files. As described in Section 4.4.2, the GridR service attaches a header to each script which makes the contents of input files accessible in the R session on the execution machine and holds information on the output file or directory names that the user can use to export data from the session.

Figure 4.21 presents details on the architecture presented in Section 4.4.2 in the context of the ACGT system.

R scripts can be integrated into workflows by using the GridR service, hence giving the possibility to perform highly complex and flexible analyses. At the workflow level the GridR service is considered as a single service that accepts very complex inputs - algorithms specified by R scripts. Thus, the GridR service can be used as task in ACGT workflows designed with the ACGT workflow editor. Upon enactment of such workflow tasks, the details and the complexity of the mechanisms involved in the execution are hidden from the user.

For the discovery of GridR scripts, the GridR service is integrated with a metadata repository. The scripts can be described by metadata that can be registered in the metadata repository (Repo) in order to publish the script in the analysis environment. This includes mainly information on the input and output parameters of the script, which are handled in the header and footer snippets that are attached to the script, as well as the script code itself. The description of individual scripts can be used to provide a convenient way of searching and reusing scripts that are developed with our method.

[1] `http://cran.r-project.org/web/packages/GridR/index.html`

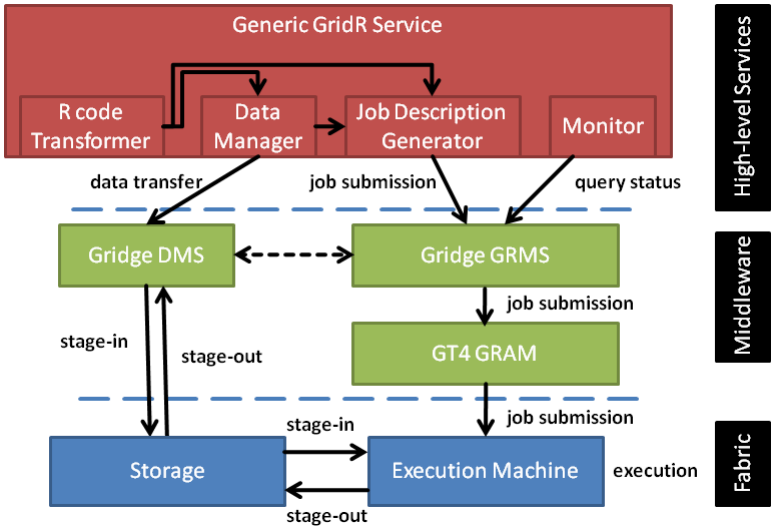

Figure 4.21.: The GridR service architecture in ACGT.

The R scripts are considered as another kind of reusable entities that are shared between the ACGT users and that are discoverable on the basis of their metadata descriptions, e.g. the algorithm they implement, the data types of their inputs, their quality properties (e.g. performance), etc.

The workflow depicted on Figure 4.22 is an example of a specialized workflow which interfaces two GridR scripts with a number of data sources (specifically: a set of mediator queries to and a set of static files) to produce the various types outputs, e.g. plots and matrices.

4.8. Case Studies

In this section we will present case studies that demonstrate the applicability of our approach in practice. In detail, we show a data analysis scenario from bioinformatics implemented with GridR and a scenario from an industrial application that is parallelized using GridR.

4.8.1. Integration of Scripts and Interactive Development - The Farmer Scenario

The scenario used in this case study is an example of a use case of the bioinformatics community. It demonstrates the GridR package and shows that a state-of-the-art scenario can be implemented with our approach presented in Sections 4.4 and 4.5. This case study is based on [150].

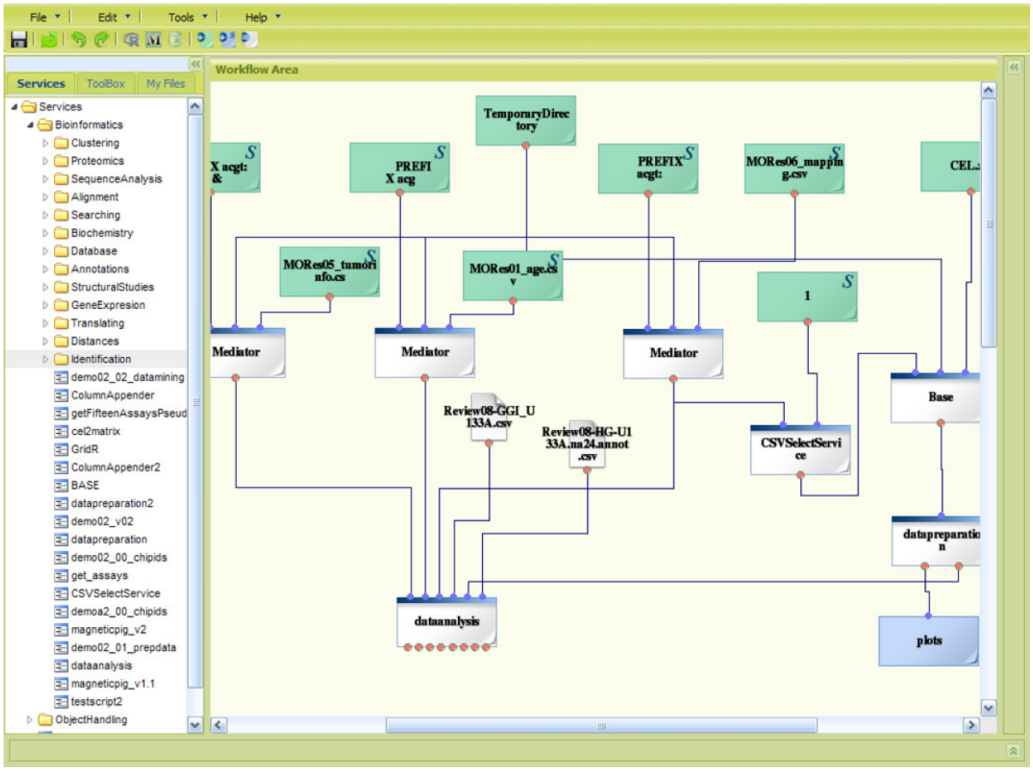

Figure 4.22.: A complex genomics data analysis workflow, represented inside the workflow editor portlet. (Two of the elements of the workflow, "datapreparation" and "dataanalysis", are GridR scripts.) The tree explorer on the left of the picture allows the user to select services registered in the metadata repository and to add them to the workflow.

The Farmer Scenario

In order to illustrate the working principles of GridR in terms of integration into the grid environment in an interactive way, we have selected the article by Farmer et al. [44], a simple clinical research project available from the literature and for which all data were available online at the GEO repository [56] (series accession number GSE1561). This also provides a validation of GridR in a realistic usage.

In the Farmer scenario, microarray data (Affymetrix U133A gene expression microarrays) obtained from breast cancer tumour samples of 49 patients are used to associate subtypes of breast cancer to patterns of gene expression and molecular signatures. R and BioConductor [55] packages are used to load, normalize and analyse the data. The present work is validated by showing that the results of the original paper can be reproduced using GridR.

The validation of GridR actually implements only a subset of the analysis steps presented in the original article, namely the principal component analysis (PCA). However, additional steps related to the quality control of the arrays are shown, which were not presented in the original paper.

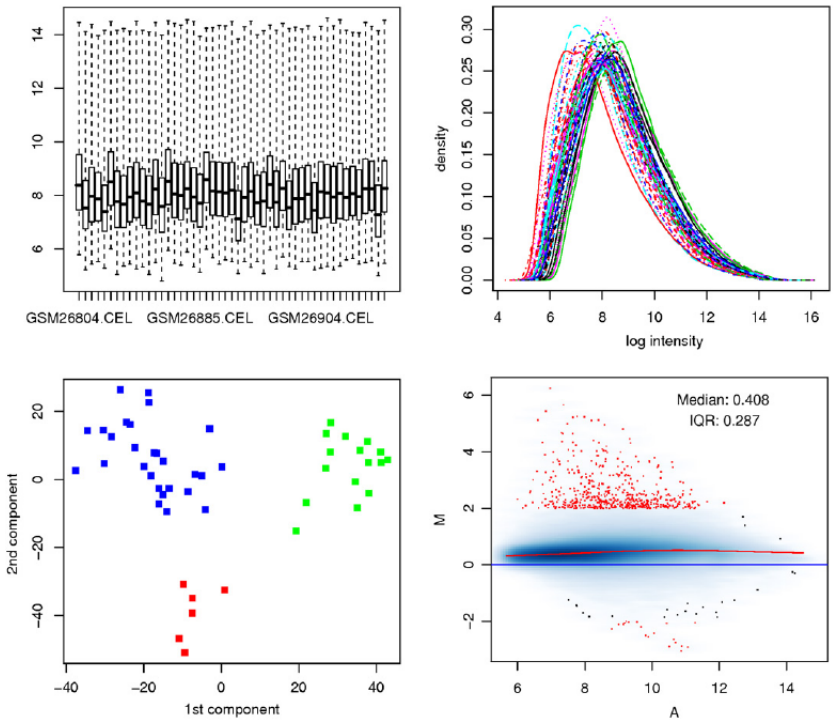

Figure 4.23.: Farmer scenario plots (from [150]).

The given scenario can be described from the technical point of view as follows: A researcher wants to perform interactive, grid-enabled data mining in the R environment. On his local machine (client) he develops algorithms using R as user interface, which he wishes to use on a clinical data set.

In clinical context, the data sets to be analysed are usually so large (about 800MB in the present scenario with a small number of patients) that a transfer to the client might be ineffective or not possible. Besides, execution machines in the grid environment might have bigger computational power than client machines. It is thus often more efficient to just ship the algorithm to the execution machine (the best being if the execution machine is the machine where the data is located in order to minimize transfer time), execute it remotely on this machine and transfer the results back to the client. This is the strategy implicitly implemented in GridR.

The Experiment based on GridR

As described above, the implementation of GridR is validated on the basis of the Farmer scenario by implementing some typical analysis steps of a microarray experiment [150]. In the present case, R was used in conjunction with the BioConductor packages *affy*, *affyPLM* and *marray*, which are specialized packages for microarray analysis, to build the individual components of validation. Besides loading the expression data matrix and associated clinical data, those components contribute in:

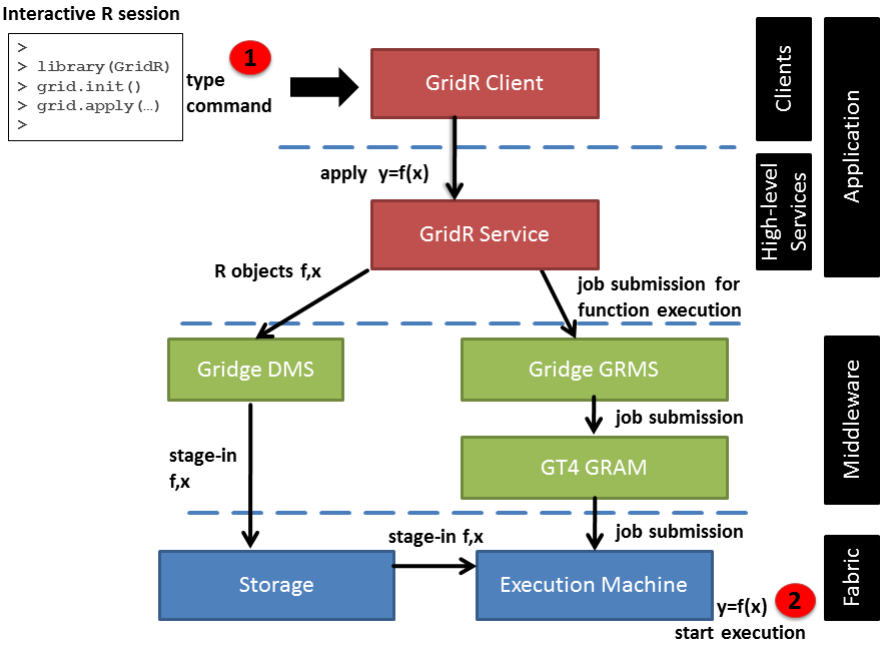

Figure 4.24.: Function Execution in the Farmer scenario (job submission: (1) type commands in the interactive R session, (2) execution on a machine in the grid is started).

- Producing figures required for the quality control of the chips.
- Producing "MvA plots" to obtain an overall view of the fraction of differentially expressed genes.
- Using a variance filter to pick unique probeset per gene and performing a principal component analysis to verify that samples with similar subtypes group together.
- Extracting symbols of genes most correlated to molecular markers relevant to the analysis (androgen receptor, AR, and estrogen receptor, ESR1).

The analysis steps are wrapped into functions for remote execution with GridR. Technically, the evaluation for computing the scenario with GridR in the ACGT testbed involves the GridR client at client side, the GridR service, a GRMS-Server installation from the Gridge toolkit [101] on a central machine in the grid environment that is responsible for resource management and orchestration of the execution machines, GT4-based execution machines, and the GridR R environment as well as the packages needed for the specific scenario installed on the execution machines. Figure 4.23 shows the plots related to those steps; in particular the plot illustrating the PCA is seen to be identical to that in [44]. Figure 4.24 visualizes how the function execution is submitted from an interactive R session as job for execution in the grid, and Figure 4.25 shows how the results are transferred back to the client session.

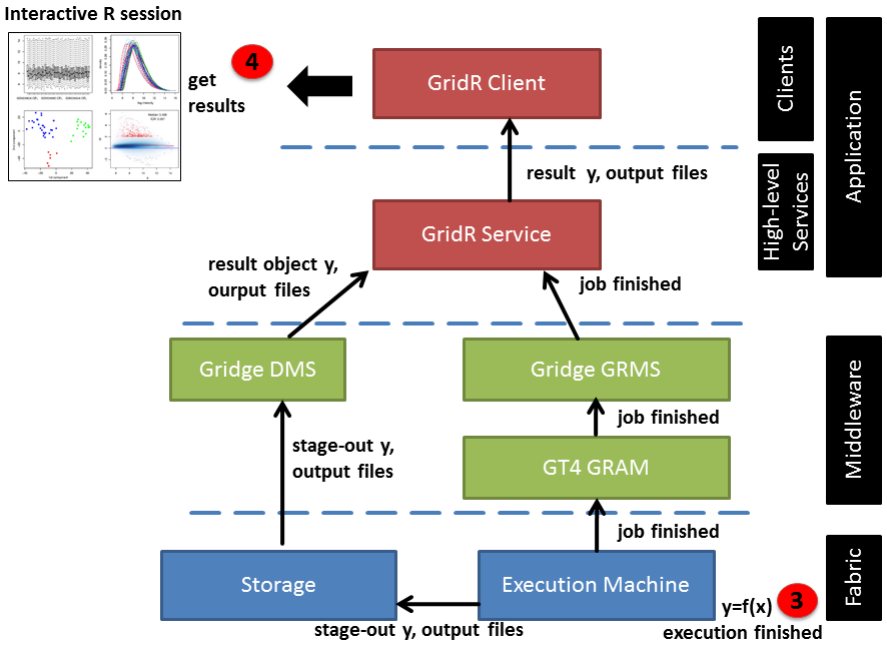

Figure 4.25.: Function Execution in the Farmer scenario (result transfer: (3) execution on a machine in the grid is finished, (4) get results in the interactive R session).

The present scenario is a proof of concept by performing a shipping of the algorithm. The infrastructure accessible to GridR may render executable R scripts that may not be executable on typical workstations. For instance, the normalization of microarray sets with a few hundreds of chips may require up to several tens of gigabytes of memory. This one-time procedure is typically performed on a single specialized server in the grid environment.

Summary

In the biomedical and biostatisticians community R is widely used and turned out as de facto standard. GridR, as one of the important analysis tools in the ACGT environment, enables users to run experiments in the grid and profit from the grid enabled environment for R. The presented case study showed the execution of R-based data mining scripts in the context of a state-of-the-art analysis scenario from bioinformatics. It becomes clear that the availability of GridR will be of great use to clinicians and clinical-data analysts interested in computationally intensive data mining, such as resampling techniques, full cross-validation of classifiers or meta-analyses.

The benefits for the users are twofold, based on the duality of using the R as a client tool and executing the R scripts on the grid. Firstly, the R environment is a popular tool among biostatisticians and the scientific community at large. Providing R with a simple way to access the resources made available in the context of ACGT is beneficial for the scientists actively using of the infrastructure. Secondly, the grid-enabling of the R execution layer ensures that the analysis tasks are executed in an efficient and secure way,

relieving the client side of heavy computational load and of the need to download all the input data sets.

This case study showed that it is possible to execute a typical bioinformatics scenario in the grid. With the Farmer scenario were able to prove that our approach supports the quick and easy execution of scripts based on the standard tool R in an interactive way. In addition, we showed that no modification on the grid environment was necessary to execute the scenario and that the details of the underlying technology were hidden.

4.8.2. Supporting Basic Script Parallelization - An Industrial Case Study

The scenario used in the script parallelization case study is based on an industrial application in the area of mobility mining. The application shares common components with bioinformatics scenarios, e.g. the Kaplan-Meier survival analysis, and has a demand for parallelization. The scenario shows how our approach of parallelizing R scripts with the GridR client (see Section 4.5) can be implemented in an industrial application. This case study is based on [141].

We apply the parallelization in a GPS-trajectory mining scenario that computes the reach and gross contacts of outdoor poster advertisement campaigns based on GPS-tracks of a set of test persons. Reach is defined as the percentage of a population exposed to a campaign within a certain period of time (e.g., a week). Gross contacts are the total number of contacts a campaign achieves. Both measures serve as common currency for comparing the performance of campaigns in the advertisement business. The scenario provides the foundation for price calculations for all outdoor advertisement throughout Germany and demands for scalability. In our case study users are enabled to execute parallel R scripts. The execution is mapped transparently to the execution environment in such a way that neither the user nor the R-programmer of the data mining script is affected.

In the following, we will present details on the GPS-trajectory mining scenario, the parallelization of the scenario with GridR and the experimental results.

Scenario

In Germany, the outdoor advertisement industry records a yearly turnover of more than 800 million Euro. The Arbeitsgemeinschaft Media-Analyse e.V. (ag.ma) - a joint industry committee of around 250 principal companies of the advertising and media industry in Germany - authorized the AGMA project, which provides the foundation for price calculations in outdoor advertisement throughout Germany. In 2006/07 the ag.ma commissioned a nationwide survey about mobile behaviour and appointed Fraunhofer IAIS to calculate the reach and gross contacts of poster networks [43].

Figure 4.26 on the left shows a campaign of 321 billboards on arterial roads in Cologne. The right hand side displays the development of reach over a period of seven days. Reach is a time-dependent measure about the publicity of a poster network. It states the percentage of people which see at least one poster of the campaign within a given period of time, e.g. one week. The campaign reaches about 50 percent of the Cologne population on the first day, after one week about 90 percent of the population have seen a poster of the campaign.

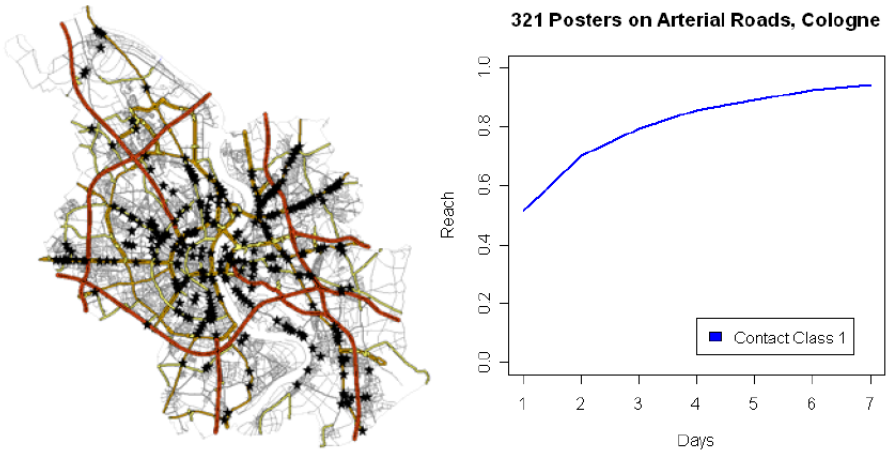

Figure 4.26.: Reach of a poster network in Cologne (from [141]).

The basic input data of the AGMA application are trajectories and poster information. Nationwide, the daily movements of about 30,000 people have been surveyed using GPS technology and telephone interviews. This data amounts to about 21 million tuples where each tuple represents a section of a trajectory which has been mapped to the street network. Poster information indicating geographic location, type and visibility is available for approximately 230,000 posters. In order to determine reach and gross contacts, the intersections of a given poster network and the trajectory data have to be calculated. However, the number of test persons is rather small compared to the whole population of Germany and the trajectories do not span the full street network. Therefore, we introduce variance to the data by performing a geographically restricted simulation of the trajectories. The resulting simulated trajectories serve as basis for the computation of reach and gross contacts. However, the trajectories suffer from in-completeness in terms of missing measurements due to defective GPS devices or the forgetfulness of test person, which easily interrupt the series of measurement days. These deficiencies are treated in the modelling step by applying the Kaplan-Meier survival analysis technique [11]. Details of the analysis technique itself are not subject of this work. A detailed documentation of the study is already available at the ag.ma website (in German).

The scenario is implemented using the statistical software R [103], as R directly supports statistical analysis including the Kaplan-Meier method. At the beginning, the script retrieves input data by triggering several database queries that read a (random) network of posters and the test persons' movement data. Afterwards, a 100-fold simulation of trajectories is performed, and reach and gross contacts are calculated. In previous tests, we determined a number of 100 simulations to achieve stable results. Finally, the results are stored in a text file.

In addition to calculations regarding a particular poster campaign, the advertising industry is also interested in mean network ratings. This requires the repeated execution of the script with randomly selected poster campaigns of specified size, and subsequent averaging of results. Depending on the size of the poster network, stable results can be achieved by using 30-100 repetitions.

The input parameters allow for a variety of combinations. For instance, the geographic location of a poster network can span from a single city up to combinations of cities to federal states or whole Germany. Also, the target population can vary. For instance, it is interesting to know the amount of contacts contributed by people living in the surrounding area of a large city. Other parameters are the size of the campaign and the poster type. Depending on parametrization, the execution time to calculate reach and gross contacts of a single campaign varies between several minutes and a few days. Thereby, the main influence factors are the number of target test persons and the size of the poster network. Clearly, the large number of parametrizations prohibits advance computation of all possible settings, so that computations must be performed on demand. This requires an execution in reasonable time, which applies in particular to mean computations, where 30-100 repetitions of a setting must be executed. Therefore, scalability plays a crucial role in the application.

Parallelization with the GridR client

The scenario we selected for experimental evaluation calculates mean network ratings for a (rather small) number of poster types and network sizes in 12 cities, amounting to a total of 92 parametrizations. Each parametrization was repeated 100 times for averaging. If computed on a stand-alone machine, the expected runtime of the experiment would amount to about one year. For being able to compute the scenario at all, the application has to be speeded up by the parallelization of subtasks of the application's algorithm. In our scenario, the algorithm can be parallelized in theory because it contains a loop with independent steps based on randomly fetched data. But, it is not directly obvious if a parallelization will improve the runtime in practice because the data is stored in a central database which could be a bottleneck.

In the following, we present a way of how to make use of parallel computation using the GridR client. As described in Section 4.6, parallelism can be expressed at different layers in GridR. We chose server side parallelization with the GridR client, as we want the scenario to be computed on a computing cluster in the infrastructure layer. R code packaged in a function or script is submitted for execution in the distributed environment. In detail, the jobs are submitted in the background to a resource management system via the GridR service. The jobs are then executed, e.g., on a Condor cluster [130], which can process them in parallel.

The application scenario is as follows (visualized in Figure 4.27): The GridR client contacts the Resource Management services of a distributed grid environment via the GridR service and submits jobs (computing the reach of poster networks). The jobs consist of the execution of a wrapper script and are processed on the dedicated execution machines. In our application scenario, the wrapper script submits n subtasks to a Condor cluster, each taking the analysis script for computing the reach as input. These subtasks compute the AGMA scenario in parallel. They draw a random sample of posters along with the trajectory data from a central database and compute the poster reach. The wrapper script waits for the appearance of all results and then averages them. In the AGMA scenario, averaging of *n=100* parallel runs varying the poster network was required to achieve the desired stability in the result.

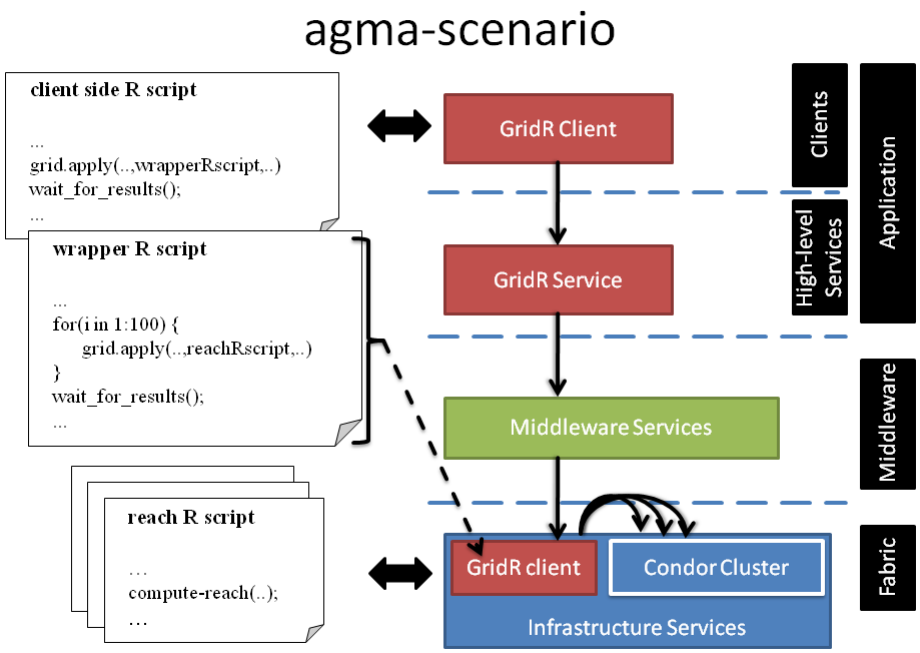

Figure 4.27.: Parallelization in the AGMA Scenario with the GridR client.

Summarizing, GridR together with the presented technique of parallelization enables users to submit a single job that is automatically split up in a number of parallel tasks on the server side. As an advantage, the code that is executed on the client side becomes smaller as there is no need for result checking and result aggregation on the client side. Furthermore, communication overhead is reduced because the splitting into parallel tasks and the result aggregation are performed on the cluster side. Consequently, performance is increased as computing clusters as, e.g., a Condor pool, are specially setup for allowing high speed communication between the execution machines.

Experiments

We conducted artificial experiments with a reduced computational load and real-world experiments to evaluate our system. The goal of the artificial experiments was to test whether it is useful to parallelize the application at all, having in mind that each parallel task has to get the trajectory data from the central database, thus inducing a parallel load on the database server. The big real-world application then delivered the data proving the feasibility of our system by computing the reach of 92 poster campaigns in 12 German cities.

The setup of our experiments was as follows: The experiments were processed on a PC cluster managed by Condor. In total, the cluster consists of 30 AMD Opteron 2.2 GHz machines running Condor on Linux. Each machine holds 2 CPUs, 8 GB memory and two local HDDs with 120 GB. The database machine is a Intel Dual Core 2x2.6 GHz with 4 GB

Simulation loop size k:	1	20	40	60	80	100
Computation time:	143	799	1496	2197	2897	3569
Database access:	75.5%	12.1%	6.5%	4.4%	3%	2.7%

Table 4.4.: Local execution of a task with varying simulation loop size k (in seconds).

# parallel tasks n:	5	10	20	30	40	50	60
Cluster runtime:	140	150	330	320	450	540	320
Summarized runtime:	625	1251	2502	3754	5005	6257	7508
Speed-up:	4.4	8.3	7.5	11.7	11.7	11.6	23.4

Table 4.5.: Parallel execution of n tasks with 75.5% database access (in seconds).

memory, 2x250 GB HDD (1 x System-HDD, 1 x DB-HDD), running Oracle 10.2 on Linux. All machines are connected by a 1GBit network connection. Throughout the experiments it could not be guaranteed that all resources of the pool were free and accessible for the full period of computation.

Artificial experiment. Each of the parallel tasks first draws the trajectory data and randomly chosen posters from the central database server and then locally computes the reach of the poster campaigns. As all parallel tasks compete in the database access, the ratio between the time needed for the database access and the local computation has a direct effect on the expected speed-up in a parallelization. We conducted experiments in which the size k of the inner simulation loop of the local computation was varied from 1 to 100, with 100 being required in the real-world experiment. The results are shown in Table 4.4. The chosen task computed the reach of a poster campaign with 321 posters, retrieving from the database the trajectories collected by 535 persons over a week in Cologne (about 430,000 street segments, roughly 7 MB). As we can see from Table 4.4, even if the database access takes 75.5% in the (hypothetical) worst-case of minimal local computation (k=1), only 2.7% of the computation time are spent with the access to the database system in the parameter setting required for the real world application (k=100), thus giving a high chance for achieving speed-up in parallelization.

Based on this positive result, we conducted an experiment to find out whether the central database server was a bottleneck in the parallel execution. We studied the hypothetical worst case in which 75.5% of the computation time was spent on data-base access (k=1) and varied the number n of parallel executions for different poster networks. The results are shown in Table 4.5. Table 4.5 depicts the results in the case in which n parallel tasks are started simultaneously and actually access the data at the same time. The cluster runtime denotes the job computation time on the cluster, the summarized runtime is the sum of the execution times of the individual tasks, and the speed-up is defined by the ratio between the summarized and the cluster runtime. The computation was parametrized with k=1 to let each local task spend 75.5% of the time with the database access. Given these conditions, the results are promising: the database system did not dramatically slow down the parallel executions. In the real-world application, the percentage of database usage is 2.7% only and the parallel runs are much more interleaved. The artificial experiments

Poster type	Advertising pressure	# Poster	# Test persons	Average runtime per task	Summarized runtime	Cluster runtime	Speed-up
CLP	normal	740	533	8326	832566	21488	39
BB	high	483	533	4619	461939	20104	23
BB	medium	321	533	3482	348169	8315	42
BB	low	241	533	2832	283218	7751	37
C	high	161	533	3353	335288	12069	28
C	medium	121	533	2751	275071	9092	30
C	low	97	533	2362	236245	10641	22
ML	normal	102	533	1712	171173	8243	21

Table 4.6.: Parametrization and results showing runtimes (in seconds) and speed-up for the computation of values for the city of Cologne - 8 jobs with 100 tasks.

correspond to the campaign depicted in line 3 of Table 4.6, in which we achieved a much higher speed-up.

Real world experiment. We tested the runtime behaviour for the computation of average campaign ratings in 12 cities. In each city, poster networks of the type column (C), billboard (BB), city light poster (CLP) or mega light (ML) were drawn respecting different campaign sizes. Each parametrization was averaged over a group of 100 tasks, amounting to 9200 tasks in total. Table 4.6 shows an excerpt of the job parametrizations and execution times for the city of Cologne.

As stated earlier, the runtime of a task depends upon the number of test persons available for the city and the size of the campaign, which is derived from the advertising pressure. Thus, the scenario provides a variety of input data. Figure 4.28 left shows the average task runtime depending linearly on the volume of input data, which is defined as the number of test persons multiplied by the number of posters.

Figure 4.28 right displays the relationship between the sum of individual task runtimes and the total runtime on the cluster for all 92 jobs. On average, we obtained a speed-up of 45, which is a plausible result for the execution of 100 tasks on a cluster of 60 CPUs. On evaluation of the results, we detected four anomalous jobs (marked with triangles in Figure 4.28 right). These did not terminate properly because some of their tasks failed. The extreme outlier on the top left results most likely from external jobs, which competed for cluster and database resources.

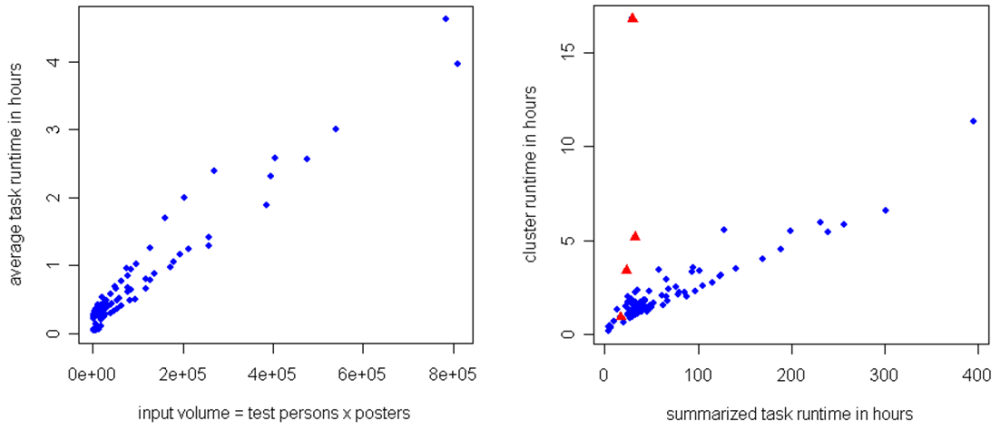

Figure 4.28.: left: Runtime behaviour vs. input volume; right: Cluster runtime vs. sum of individual job runtime.

Summary

In this case study we have applied techniques for the parallelization of R-scripts in grid environments using GridR as described in Section 4.6. The considered application enables marketing departments from all over Germany to compute the reach of out-door poster campaigns based on trajectories of test persons. As the R-based computation of the reach requires substantial computational efforts, the application requires access to a pool of execution machines that executes the analysis tasks and holds the data. With GridR, we enabled the parallel execution of R scripts. Using this technique in our application, a client can submit and initiate parallel computations of R scripts. In the considered application, we managed to compute the complete scenario of 12 cities in a few days compared to the hypothetical sequential execution time of roughly a year.

The case study showed that it is possible to execute R scripts efficiently with GridR and that our approach is applicable to other domains than bioinformatics.

4.9. Wrap-up

In this chapter we have presented an approach for flexible and interactive development of data mining scripts in grid-based analysis environments. The approach is based on a grid service with complex inputs and outputs that allows for providing data mining algorithms implemented as scripts as parameters (the GridR service). It reduces the complexity of integrating and executing data mining scripts in grid environments, as it does not require the registration of each individual script separately in the grid registry. In addition, we presented an extension to the R environment that allows for interactively developing data mining script including the development and composition of data mining components (the GridR client). Users are enabled to interactively develop data mining scripts directly in the grid environment. Furthermore, we presented how parallelization can be introduced into our approach. Several case studies showed that the approach solves problems in practice, that is supports complex scenarios, and that it is generalizable to other settings in industry.

Data mining components and scripts are usually part of larger processes, realized as executable workflows. In the next chapter, we present an approach on supporting the reuse of existing data mining processes including components and scripts.

5. Data Mining Process Patterns

The goal of this chapter is to introduce an approach for the reuse of existing data mining based analysis processes in bioinformatics scenarios. Data mining components and data mining scripts, which have been discussed in Chapters 3 and 4, are usually parts of larger analysis processes. These are typically realized in the form of executable workflows. In contrast to data mining scripts, which already enable users to compose components, workflows provide a higher level of abstraction than a scripting language, which reduces complexity for users when composing components and scripts.

Reuse of analysis processes becomes much more important in the area of bioinformatics due to complex process chains that have to be set-up for today's analysis solutions. However, the support for reusing existing data mining based analysis processes lacks today. On the one hand, deployed executable workflows often cannot be reused directly, as they are customized to a certain scenario and as the information on how the workflow was set-up and on which requirements have to be met for executing the workflow is often not available. Although workflows are already shared and reused in the area of bioinformatics, Goderis et al. identified "barriers that keep people from effectively processing the available workflow knowledge" [60], e.g. bottlenecks in knowledge acquisition about workflows. On the other hand, abstract process descriptions, e.g. based on the standard data mining process model CRISP [121], are reusable but require a lot of effort for creating an executable workflow out of this information. Thus, the reuse and integration of existing solutions is not often or only informally done in practice due to a lack of support and bottlenecks to reuse [60], which leads to a lot of unnecessary repetitive work.

In this chapter, we focus on the reuse of existing processes including its components rather than on workflow planning or on recommending components and workflows for reuse. This means that we assume that the process and the components that are going to be reused are known.

In the following we present our approach on supporting the reuse of existing data mining processes by formally encoding both technical and high-level semantics of these processes in so called *data mining process patterns*. Data mining process patterns facilitate the integration and reuse of data mining in analysis processes by providing a description of a process at a level of abstraction between the CRISP model as most abstract process and an executable workflow as most concrete process. The pattern approach is based on encoding requirements and prerequisites inside the process and a task hierarchy that allows for generalizing and concreting tasks for the creation and application of process patterns.

This chapter first motivates the need for a new approach on reusing data mining processes in Section 5.1. Second, it introduces the requirements for supporting the reuse of existing data mining processes in Section 5.2. Third, we present related work in Section 5.3. Fourth, Section 5.4 describes how to modify the CRISP model to focus on the special

case of reuse of existing solutions. Then, Section 5.5 introduces our approach on reusing analysis processes based on data mining process patterns. We present how data mining processes can be abstracted to data mining process patterns and how data mining patterns can be specialized in order to reuse them for a certain analysis problem. Subsequently, we present how the pattern concept can be used for modelling data requirements in Section 5.6. After that, we present three case studies. In Section 5.7 we evaluate the concept of data mining process patterns in the context of a bioinformatics scenario. In Section 5.8 we present how to create a pattern describing the general process of meta-analysis from a multi-center multi-platform scenario. In Section 5.9 we show how to integrate data mining process patterns into business processes. Finally, Section 5.10 summarizes and wraps-up. This chapter is mainly based on [114, 140, 146, 147, 148].

5.1. Motivation

Currently, existing data mining processes can be reused at different levels. An abstract level of reuse is passing through a new CRISP process while being inspired by existing solutions (e.g. by personal experience or reading respective documentation and scientific papers), while the reuse at implementation level (e.g. by copy-and-paste of existing code from scripts and excerpts of workflows) is a concrete level of reuse. In this section we argue that there is a need for an approach in-between.

The first approach (abstract level) for reuse is following a new CRISP process (see also Section 2.1.2) based on information and outcomes of an existing CRISP process. CRISP describes in an abstract way how data mining processes are performed and guides in instantiating this abstract process for a given problem by a breakdown from generic to specialized tasks.

It is known that the CRISP model lacks in the deployment phase [116] and misses phases important for engineering projects [89]. Based on experience in software engineering, [89] proposes a model for data mining engineering that includes engineering-related phases which are missing in CRISP. They identify the open issue that available process models specify what to do, but not how to do it. [120] also identifies the lack of guidance towards implementing particular tasks of data mining methodologies and introduces a framework for the implementation of the business understanding phase of data mining projects. In addition, it was detected that many redundancies and inefficiencies exists when following the CRISP model in parallel to approaches that work on already modelled processes, as e.g. in the field of Business Process Management (BPM) [146].

Summing up, the approach of following CRISP does not suffice, as it is often too general. There is no support to create executable workflows from a CRISP process - the user has to follow all CRISP phases and concretion steps.

The second approach (concrete level) is to utilize reuse at implementation level. Data mining processes can be reused by making use of available data mining workflows. There exist several frameworks and tools for the development, composition, deployment and execution of workflows based on data mining components and scripts, e.g. RapidMiner [93] and Weka [156] in the area of data mining, Taverna [72], Triana [32], Kepler [86] and Galaxy [61] in the area of scientific workflows or jBPM [79] and YAWL [129] in the area

of business processes (see also Section 2.2.3). However, deployed executable workflows are often too specific and too detailed for being reusable efficiently. The workflow tasks, which include data mining components and scripts, are fully specified and parametrized so that they are directly executable by a workflow engine. In contrast, the information on the requirements and prerequisites that have to be met in order to process these tasks as well as interdependencies between the tasks are not or only implicitly encoded in the workflow. What is left to be answered is how to describe the processes implemented as workflows in a way that allows for an efficient reuse.

For example, in an analysis of a large set of real-world data mining workflows [21, 114] we detected that the changes of a workflow during the lifetime of a data mining project are made to the same extend at the preprocessing and at the modelling part, which implies that understanding and representing the semantics of the data is a very important step. The changes for the preprocessing part consisted to 50% and for the modelling part to 75% of manual parameter optimizations. The challenge is to reuse such kinds of manual fine-tuning. A data mining process is a complex process that requires a lot of manual optimizations and is not always transferable due to the dependency to the data. A copy-and-paste approach, by taking over the data mining part from another analysis process, will only work if the analysis process into which the data mining is integrated has exactly the same properties as the original one. What is needed is a way to specify tasks of the process at the correct level of abstraction for enabling reuse.

In the area of data mining there exist approaches on supporting the design of workflows by ontologies which describe the workflow objects, e.g. data, meta data or components [69, 137]. In the area of business processes, several workflow patterns have been identified that describe the control-flow of workflows [117] (see also Section 2.2.3). In the area of scientific and bioinformatics workflows, there exist efforts in describing workflows based on different levels of abstraction [159, 29] and on supporting reuse by enhancing workflow discovery [59]. In our approach we propose to combine parts of these concepts, providing process patterns that allow for the description of tasks at different levels of abstraction and include support of ontologies for describing data.

As visualized in Figure 5.1, the approach should support the description of the process at different levels of abstraction between the CRISP model as most general representation and executable workflows and code as most concrete representation. For doing so, we will introduce the concept of *data mining process patterns* and a *meta-process* that describes the steps needed for applying these process patterns.

5.2. Requirements

In many cases data mining processes could in principle be reused without the need for detailed technical knowledge. However, in practice the reuse is too complicated or not efficiently supported.

We aim at supporting reuse in an easy way. However, the exact meaning of easy depends on the capabilities of the users, as reuse of different parts of a process presupposes different kinds of previous knowledge. This includes, e.g., knowledge about the details of a component, knowledge on the composition of components in a script or in a workflow, domain

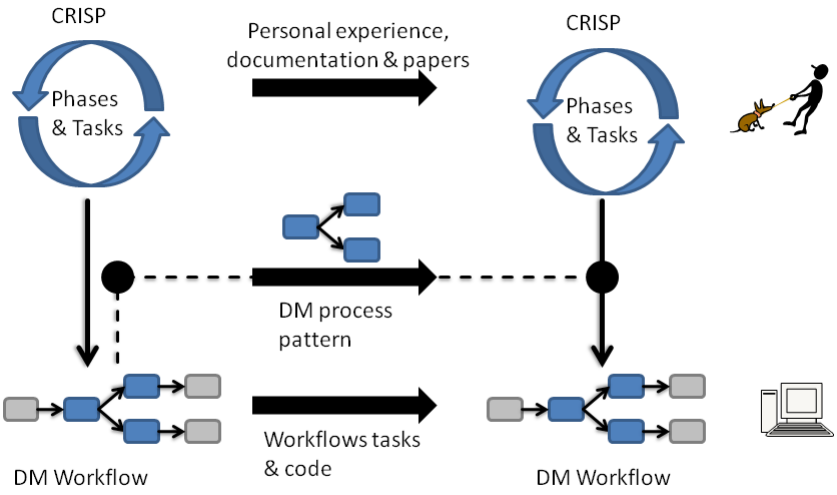

Figure 5.1.: Different strategies of reusing data mining.

knowledge about the data, or knowledge about the integration into higher processes.

We focus on users with a lot of domain knowledge in bioinformatics, but less technical knowledge. Motivated by the way how bioinformaticians work (as described in Section 2.3), we categorize the technical knowledge according to the kinds of changes that are needed to reuse a process. In detail, we distinguish the ability to

- **use a component**: the execution and parametrization of components are usually supported by user interfaces, e.g. in a graphical way or via command line.
- **develop a component**: the development of new components needs expertise in programming and software development and is supported by development environments.
- **use a script**: the execution and parametrization of scripts needs expertise in scripting languages, which are usually supported by command line interfaces.
- **develop a script**: the development of new scripts which can include components needs expertise in scripting languages, which are usually supported by environments for text based script composition.
- **use a workflow**: the execution and parametrization of workflows are supported by GUI based workflow environments.
- **develop a workflow**: the development of new workflows which can include components and scripts needs expertise in workflow composition which is usually supported by drag-and-drop based workflow environments.

User groups are defined by their capabilities, e.g. it might be the case that clinical end-users are only able to develop workflows but cannot develop scripts or components due to missing expertise in scripting and programming.

The goal that we want to achieve is to define a description for data mining processes that allows for the reuse of these processes including data mining components and scripts. The processes can vary from general processes as described by CRISP as most abstract processes up to executable workflows as most concrete processes. Thus, tasks may only be reusable at a level of abstraction that requires manual work to process or further specify them. In addition, tasks may only be reusable under certain assumptions. Thus, the prerequisites for reusing them need to be covered by the process description. All in all, the method for the description needs to fulfil the following requirements:

1. it has to allow the description of processes at different levels of abstraction between the CRISP model as most general representation and executable workflows as most concrete representation.
2. it has to cover the description of the prerequisites that have to be met in order to execute the process.
3. it has to include the description of manual tasks that need to be performed for executing the process (e.g., manual quality checks based on plots).
4. it has to cover actions required by the user to specialize abstract tasks (e.g., parametrization, the creation of a sub-workflow, etc.).
5. if it is in principle possible to reuse a process by generalizing a certain task or set of tasks, it should be possible to describe this information in a way that allows for easy reuse with as few previous knowledge as possible.

Our approach to address these requirements is to define *data mining process patterns* that describe processes including pre-requirements at different levels of abstraction and to define a *meta-process* that allows end-users without deep knowledge in data mining to reuse data mining processes based on these patterns.

5.3. Related Work

In the area of workflow and process modelling, there exist the two main fields of scientific workflows and of business processes. Today, data mining is part of processes and workflows in many business and scientific application domains, and in particular in bioinformatics and healthcare. Scientific workflows are used, e.g., for prevention, diagnosis, therapy, prognosis, etc., but also for other problems in healthcare such as resource planning or fraud detection [78].

Data mining processes are implemented in various ways, as stand alone software, in toolkits, as workflows, etc. (see also Section 2.1.3). Processes involving data mining introduce additional dependencies among tasks as well as a combination of automated and manual tasks [70]. The standard data mining process model CRISP represents the underlying abstract process model for many data mining processes (see also Section 2.1.2). Modern process frameworks provide great support for flexible design, deployment and management of workflows. However, data mining needs a lot of domain knowledge and thus is difficult to handle for non-experts.

In the area of bioinformatics, setting up and executing scientific data analysis processes that include data mining as described in Section 2.3 require a lot of manual work. Typically, such processes consist of a phase involving mainly manual tasks - including data search and access, data understanding and data preparation, and a semi-automatic analysis phase based on scripts or workflows - including further preprocessing, data fusion and the actual analysis.

Often, bioinformatics processes are modelled in form of executable workflows in different workflow environments. These environments are mainly based on data-flow oriented workflow languages. Existing environments for scientific data analysis processes are, e.g., Taverna [72], Triana [32], Kepler [86] or Galaxy [61] (see also Section 2.2.3). The workflow environments allow for modelling the steps of executable workflows, but lack in providing the encoding of requirements or (manual) tasks at different levels of abstraction, which is important for reuse.

In the area of scientific workflows it is often distinguished between abstract and concrete workflows [39]: "*Abstract* workflows capture a layer of process description that abstracts away from the tasks and behaviour of a *concrete* workflow." In [29] the authors introduce another level of abstraction above the abstract workflows - the conceptual level. A conceptual workflow aims at capturing the user intentions when designing a process. In addition, they define the concept of patterns as reusable fragments that are woven into the process dynamically when specializing conceptual or abstract workflows. In [159] the authors present a hierarchical workflow structure representation that contains four levels of representation: abstract workflow, concrete workflow, optimal workflow and workflow instance. The advantages they see are that users with different levels of experience might create workflows at these different levels, that (semi) automatic transformation of workflows is enabled and that (partial) reuse of workflows is defined at different levels of abstraction. However, both approaches do not specifically address the encoding of requirements and manual tasks.

In [10] the authors propose a framework for the reuse of scientific workflows, which are also based on reusable process patterns that include abstract tasks. However, their work focuses rather on technical tasks like copying, job execution and monitoring than on data mining specific tasks.

In the context of business processes, there exist huge efforts in research and implementation of business process management (BPM). These efforts result in lots of methods and approaches for process modelling, process instantiation, process execution, etc., and various implementations of business process management systems. Modern SOA-based Business Process Management (BPM) environments, e.g. based on standards like BPEL [4] and BPMN [153], provide flexible and user friendly environments and tools for designing, deploying and managing business applications. BPM principles, methods and tools support the creation and management of business processes by graphical modelling, (automatic) transformation into executable workflows, easy deployment and easy inclusion of external services. Common among a majority of such BPM systems is that processes are modelled in a control-flow based modelling language.

In [117] a set of workflow patterns describing the control-flow perspective of workflow systems is defined. According to Atwood [15], such patterns have plenty of advantages: BPM processes serve as both the specification and the source code. The modelled processes

become the solutions deployed and provide a simple communication tool between end-users, business analysts, developers and the management. Workflow patterns provide a proven and simple technique to shorten the learning curve and improve productivity and quality of the processes designed as they are simple to understand, learn and apply immediately.

Given that these powerful BPM environments and the CRISP model exist, one could assume that it is very straightforward to efficiently reuse data mining processes. However, in practice still many redundancies and inefficiencies exist [146].

In [83] the authors outline how a system can be built that supports users in the design of data mining workflows out of distributed services for data understanding, data integration, data preparation, data mining, evaluation and deployment. The support they aim at includes checking the correctness of workflows, workflow completion as well as storage, retrieval, adaptation and repair of previous workflows. The authors present a data mining ontology (DMO), in which all services including their inputs, outputs, preconditions and postconditions are described. They propose to build a support system based on the DMO, which would also allow for meta-learning for algorithm selection [69]. Based on the ontology from [69], workflow templates have been defined [84] that can mix executable tasks and tasks that need to be refined into sub-workflows. The workflow templates contain only tasks that are described by concepts at the upper level of the ontology. Such ontologies represent a good way for describing tasks and components that are part of workflows, but lack in covering the description of manual steps and actions that need to be performed to abstract and specialize tasks. The workflow templates are useful for describing automated workflows and allow, in combination with the ontology, for precondition checks for individual tasks, but do not cover manual tasks that need to be performed for executing the process.

Enabling the reuse of existing solutions for similar scenarios has the potential of making the development of analysis process much more efficient. In principle, processes are reusable in different scenarios, just by performing changes on certain components of them. But, it is not obvious how to exactly do this, as this knowledge is typically not formalized. Thus, data mining can only be reused efficiently and successfully in this context if the user is supported during the task of constructing and reusing processes. Data mining based analysis processes in bioinformatics need to be described in a meaningful way. For doing so, it is necessary to know which characteristics and parts of the process have to be described and which have not.

5.4. Analysis of the CRISP Model for Reuse

In the following, we will describe the CRISP phases and tasks in detail and present how these differ in the case of reusing existing solutions compared to executing CRISP from scratch. Depending on the aims of the tasks of the individual CRISP phases, the tasks are either considered to be part of a data mining process pattern if they are reusable, considered to be part of the meta-process if if they are related to following the procedure of reusing a data mining process, or considered to be obsolete for reuse. Figure 5.2 visualizes the mapping of the CRISP tasks.

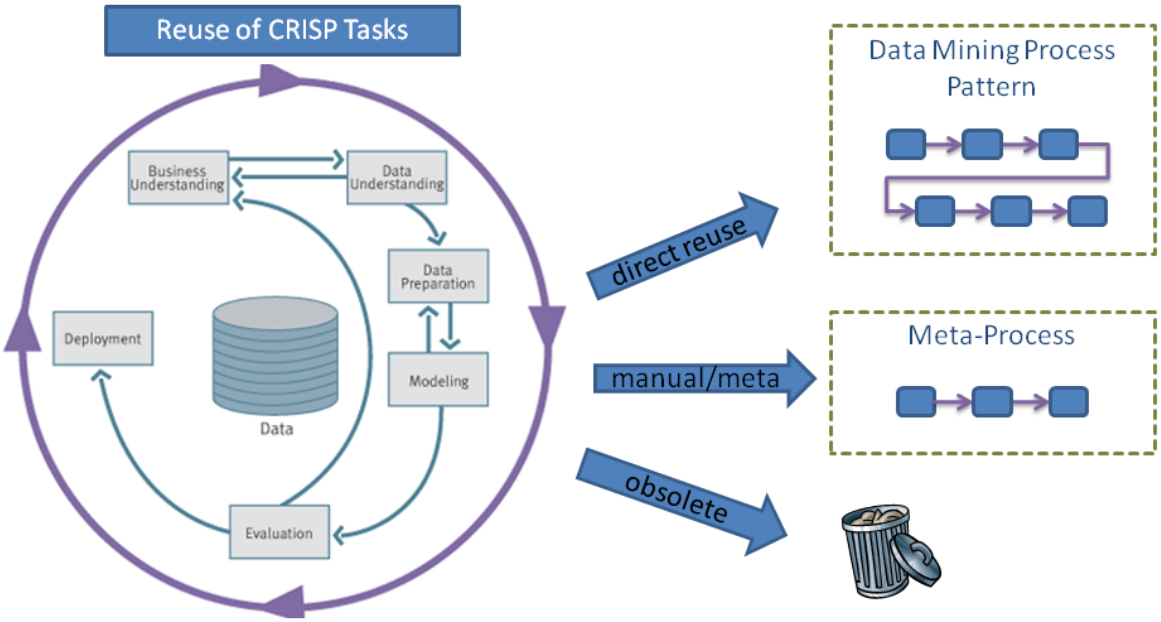

Figure 5.2.: Mapping CRISP tasks to data mining process patterns and the meta-process.

In the following, we give details on the generic CRISP tasks (based on [30]) and their mapping to the process patterns and the meta-process:

- Phase **Business Understanding**
  This phase focuses on understanding the project objectives and requirements from a business perspective, converting this knowledge into a data mining problem definition and a preliminary plan to achieve the objectives.
  - *Determine Business Objectives*: The task Determine Business Objectives is a general task that sets the goal of the overall process. Business Objective means here answering the research question of the bioinformatics scenario. We arrange this task at the start of the meta-process, as it provides the information needed for the choice of a process pattern.
  - *Assess Situation*: The task Assess Situation involves the set-up of an inventory of resources, a collection of requirements, assumptions, etc. In our scenario, this task does not apply as the important information is already available through the existing process.
  - *Determine Data Mining Goals*: We transform the task Determine Data Mining Goals into a task that checks if the data mining goal is matching and arrange it at the beginning of the data mining process pattern, as the data mining goal is already specified in a data mining pattern.
  - *Produce Project Plan*: The task Produce Project Plan is outside of the scope, as the project plan is following the procedure for the reuse.

- Phase **Data Understanding**
  This phase is based on an initial data collection and includes activities in order to get familiar with the data, to identify data quality problems and to discover first insights into the data. The description of data and data requirements will be discussed in detail later in Section 5.6.
  - *Collect Initial Data*, *Describe Data* and *Explore Data*: The tasks Collect Initial Data, Describe Data and Explore Data are considered as obsolete as we assume the data to be available through the modelled analysis process.
  - *Verify Data Quality*: The task Verify Data Quality is mapped to a task at the pattern level.

- Phase **Data Preparation**
  This phase includes the activities to construct the final dataset from the initial raw data, which can then be fed into the modelling tools. Data preparation tasks include, e.g., table, record and attribute selection as well as transformation and cleaning of data. They are likely to be performed multiple times and not in any prescribed order.
  - *Select Data, Clean Data, Construct Data, Integrate Data, Format Data*: These tasks are all preprocessing tasks at the pattern level.

- Phase **Modeling**
  This phase deals with selecting and applying various modelling techniques including the calibration of their parameters to optimal values. Typically, there exist several techniques for the same data mining problem with different specific requirements on the form of the data. Therefore, stepping back to the data preparation phase is often necessary.
  - *Select Modelling Technique, Generate Test Design, Build Model, Assess Model*: These tasks are part of the patterns.

- Phase **Evaluation**
  This phase involves evaluating the outcome of the modeling phase, the built models that appear to have high quality from a data analysis perspective, from the business perspective. It leads to a decision on the use of the data mining results.
  - *Evaluate Results*: The task Evaluate Results involves a matching with the business objectives. Thus, we arrange this task at the meta-process.
  - *Review Process*: The task Review Process is implicitly contained in loops of the meta-process (changing the specification of tasks of a data mining pattern or choosing another pattern).
  - *Determine Next Steps*: The task Determine Next Steps does not apply as the next steps are defined by the meta-process.

- Phase **Deployment**
  This phase deals with organizing and presenting the knowledge gained in a way that the customer can use it, e.g. by applying models within an organization's decision

making process. In addition to the data mining expert, the customer or end-user responsible for the higher processes is involved in this phase.

- *Plan Deployment*: The planning of the deployment by the task Plan Deployment does not apply, as in our context the deployment is always an executable process. Thus, we transform it into a task for deploying the process which is arranged at the level of the meta-process.
- *Plan Monitoring and Maintenance*: The task Plan Monitoring and Maintenance does not apply as well, as monitoring and maintenance are handled by the process environments anyway.
- *Produce Final Report*, *Review Project*: The tasks Produce Final Report and Review Project are outside of the scope, as we are not interested in such a kind of deployment.

Figure 5.3 visualizes the mapping of the generic CRISP tasks.

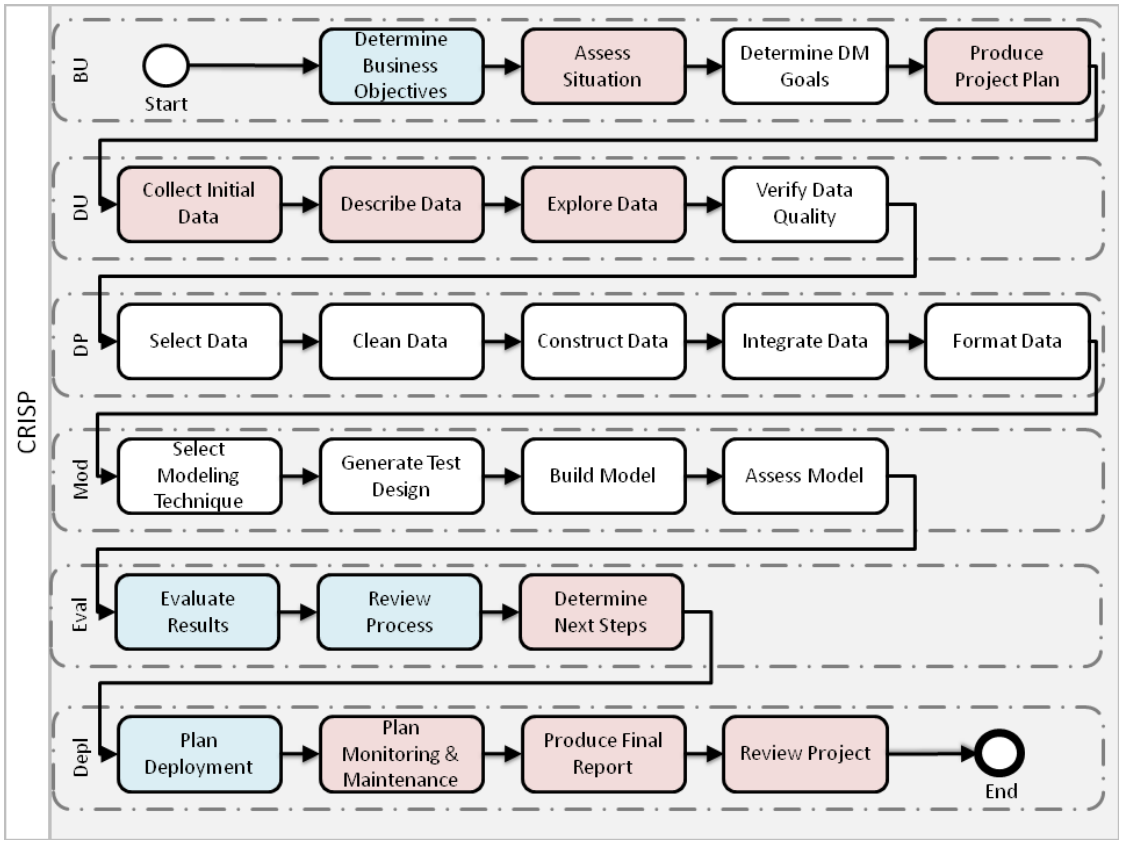

Figure 5.3.: Mapping of the CRISP tasks. White tasks are part of patterns, blue tasks are mapped to the meta process, red tasks are obsolete.

5.5. Data Mining Process Patterns for Data Mining based Analysis Processes

Currently, approaches on describing data mining based bioinformatics processes have certain lacks in supporting the requirements for reuse presented in Section 5.2. To fulfil these requirements, an approach is needed that addresses the characteristics of data mining processes as described by CRISP, allows for describing abstract processes, executable workflows and abstractions in-between, and covers the description of requirements, prerequisites and manual tasks.

We aim at an approach for supporting the reuse of existing data mining based analysis processes that have proven to be successful, and hence want to develop a formal and concrete definition of the steps that are involved in the data mining process and of the steps that are necessary to reuse it in new analysis processes. Thus, we focus on the reuse rather than on the data mining problem itself and consider a solution for the data mining problem to be available.

We propose to provide an approach inspired by the workflow pattern concept from [117] for the reuse of data mining processes - process patterns that represent templates for different data mining problems. These process patterns have to include the definition, description and requirements of the data mining process, but are independent of the application scenario. In addition to the workflow templates presented in [84], the process patterns include also the description of manual tasks and of actions required by the user to specialize abstract tasks. The goal of these data mining process patterns is to provide a flexible representation for different levels of generality.

For the abstraction of data mining based analysis processes, as it is also proposed by [29] and [159], it is necessary to analyse which parts of such processes can be reused and if it makes sense to reuse them. We consider the generic CRISP model as basis for this analysis. Our approach splits up into two main steps: the abstraction of existing analysis processes from a given scenario to create data mining process patterns, and the specialization of data mining process patterns to executable analysis processes for new scenarios.

In this section, we will introduce our approach on the reuse of existing data mining based analysis processes. First, the concept of data mining process patterns is presented in Section 5.5.1. Second, Section 5.5.2 gives details on the abstraction of process patterns from workflows and how executable workflows can be created from process patterns by the specialization of tasks.

5.5.1. Definition of Data Mining Process Patterns

In the following, we will present our approach for the specification of data mining process patterns at different levels of generality.

The CRISP methodology includes a four-level breakdown, which describes the instantiation of the CRISP process model in order to get a CRISP process (see Figure 5.4). The 6 CRISP **phases** consist of several **generic tasks** which cover all possible data mining applications. Out of these tasks, **specialized tasks** are created which specify specific actions for a certain situation. Finally, the **process instances** represent a record of actions

and decisions of an actual data mining engagement. This top-down approach is a manual process which is not automated.

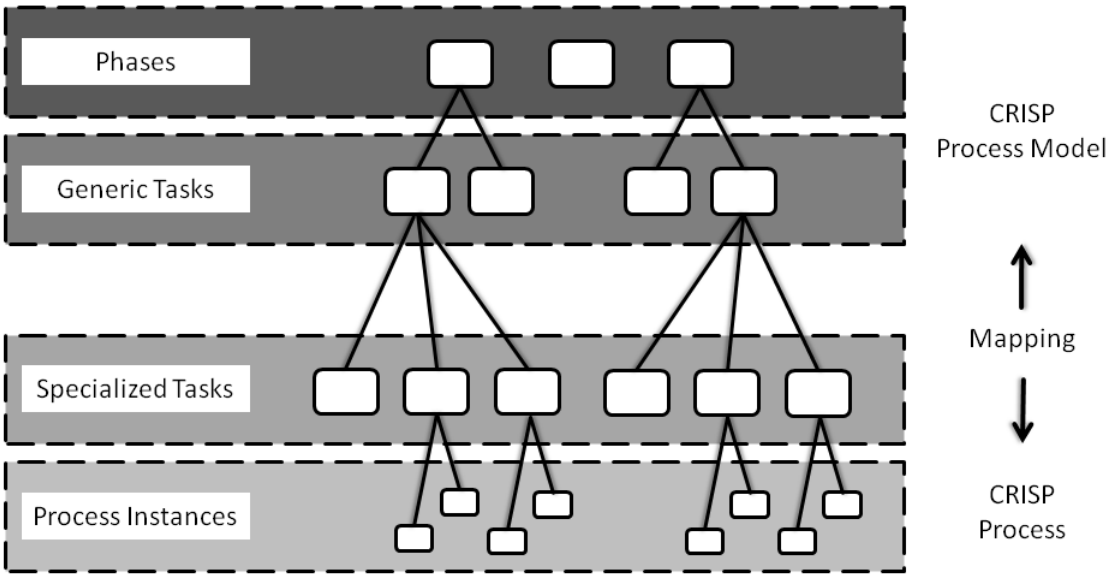

Figure 5.4.: Four level breakdown of CRISP (based on [30]).

In the context of the CRISP breakdown, an executable data mining workflow (as defined in Section 2.1.4) is a process instance that includes automated steps only. This means, that for existing workflows there is only information available at the process instance level. Information from the higher levels is either lost or only implicitly contained in the modelled workflow. However, we need to take into account that reuse may in some cases only be possible at certain level. E.g., on the one hand tasks like checking pre-requirements could only be formalized as manual tasks, but on the other hand there could be a detailed data mining workflow available where only some parameters for the modelling need to be specified, which is already formalized and only needs to be a little adapted. Thus, parts of the existing processes need to be abstracted in order to reuse it. Such an abstraction has to be done at different levels.

In the following, we will define tasks and task levels.

Definition 7 ***(Task)*** *A task is a named element. A set of tasks is a finite set of named elements.*

A **task** represents a step of a process and can be visualized by a Core Flow Object **Activity** or **Gateway** as defined by BPMN (see also Section 2.2.3).

We will define data mining process patterns in a way that the CRISP breakdown is partially pre-defined, where some of the tasks could be defined on a detailed level (process instance), but others at higher levels (specialized or generic). Thus, in order to allow for a description of processes that support reuse at the level of the general CRISP model, of executable workflows, and of abstractions in-between, we define the following different levels of granularity for tasks:

Definition 8 ***(Task Levels)***

- ***executable level****: A task is described at the executable level if there exists a description of the task that allows to execute it automatically. Tasks at the executable level consist of a mapping to an existing component and a set of already specified inputs. The inputs can be either directly defined in the configuration of the component, provided by results of previous tasks or provided as inputs for the overall process. A task at the executable level is called* ***executable task****.*

- ***configurable level****: A task is described at the configurable level if there exists a description of the task that specifies a mapping to an existing component and a set of configurable inputs that are needed by the component. The tasks have to be processed manually in terms of specifying the missing input. A task at the configurable level is called* ***configurable task****.*

- ***structural level****: A task is described at the structural level if there exists a description of the task in form of a graph $G = (V, E)$ comprising a set V of sub-tasks together with a set E of directed edges, which are 2-element subsets of V, and a textual description on how to further specialize the sub-task(s). A task at the structural level is called* ***structural task****.*

- ***conceptual level****: A task is described at the conceptual level if there only exists a textual description of the task. A task at the conceptual level is called* ***conceptual task****.*

Figure 5.5 describes how we visualize the different levels. Conceptual tasks consists of a textual description that includes information on how to further specialize the task. E.g., if a task of a process is not reusable, it needs to be replaced such that the process becomes reusable. Thus, there might be a need to develop a new atomic data mining component or a data mining script to be able to reuse the process. The description of such tasks refers to the conceptual level. In addition, prerequisites for the process and manual tasks, e.g. a task for checking if the plots as results of an analysis are satisfying, can be described by conceptual tasks. The user needs to manually process such conceptual tasks.

Structural tasks consists of a partially formalized description that pre-structures the task by a graph of sub-tasks and gives information on how to further specialize the task. Tasks for organizing components, for developing or adapting a workflow or for developing scripts from available existing components are described at the structural level. E.g., a data preprocessing task which consists of the two steps normalization and filtering, but the components for these steps are not yet specified, is a structural task. Structural tasks have to be processed manually by the user.

Configurable tasks are already bound to an existing component, but cannot be executed as there is further input needed. Tasks for parametrization of existing component, scripts or workflows, are configurable tasks. E.g., a data fusion task which needs the identifiers for the records of two tables to join provided as parameters is a configurable task.

Executable tasks can be directly executed, which means that they can be used to describe the tasks of an executable workflow. Tasks for executing components, scripts and

workflow are described at the executable level. No user interaction is needed for processing these tasks. E.g., a task for executing an analysis that is fully specified by an R script is an executable task.

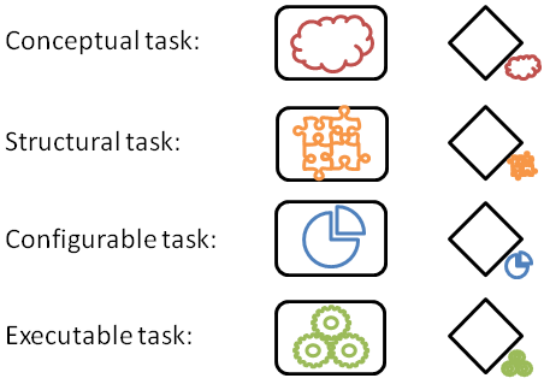

Figure 5.5.: Visualization of tasks levels.

Tasks from the different levels from Definition 8 can be specialized to tasks from lower levels in the following way:

Definition 9 ***(Specialized)***

- *A structural task is specialized from a conceptual task if the textual description of the conceptual task is manually processed, a graph of connected sub-tasks is generated and a new textual description on what is necessary to further specialize the structural task is created.*
- *A configurable task is specialized from a conceptual task if the textual description of the conceptual task is manually processed, a component is created or an existing component is selected, and a set of inputs is created that need to be configured.*
- *An executable task is specialized from a conceptual task if the textual description of the conceptual task is manually processed, a component is created or an existing component is selected, and the inputs for the component are provided.*
- *A configurable task is specialized from a structural task if the textual description of the structural task is manually processed, a component is created or an existing component is selected for the sub-task the configurable task refers to, and a set of inputs is created that need to be configured.*
- *An executable is specialized from a structural task if the textual description of the structural task is manually processed, a component is created or an existing component is selected for the sub-task the configurable task refers to, and the inputs for the component are provided.*
- *An executable task is specialized from a conceptual task if the inputs for the component of the task are provided.*

Thus, the task levels represent a **hierarchy of tasks**, where the executable tasks are described at the most detailed level and the conceptual tasks are described at the most general level. E.g., a *Clean Data* task could be specified as human task (conceptual task), as component that deletes records with missing values (executable task) or replaces them by a user defined value (configurable task), or as separate data mining process for the prediction of missing values (structural task).

It depends on the knowledge and capabilities of the bioinformaticians, at which level they are able to perform manual tasks for the reuse. The different levels of tasks are technically supported by different kinds of tools:

- Conceptual tasks for the development of data mining components are supported by software development environments.
- Conceptual and structural tasks for the development and composition of components in the context of data mining scripts are supported by scripting environments.
- Structural tasks for the composition of components and scripts, and the development and adoption of workflows are supported by workflow environments.
- Configurable tasks for specifying parameters of components, scripts or workflows are supported by the components itself, the scripting or workflow environments, or by additional systems such as a web site that guides through the parametrization.
- Executable tasks that specify the usage of components, scripts and workflows are supported by the runtime environments necessary for components, by the scripting environments or workflow environments.

As described above, conceptual tasks can also include the description of requirements and preconditions. Encoding the requirements and preconditions allows a faster identification of problems related to (re)using the data mining process. We distinguish between requirements that need to be met in order to reuse a process pattern (which are checked once) and requirements that are needed to execute the analysis process (which are checked at each execution of the process). When specializing conceptual tasks describing requirements, the former are omitted after the process pattern has been selected, while the latter are specialized to executable tasks that check the requirements. In Section 5.6, we will describe how to deal with data requirements in detail.

Next, we will define a task graph as a set of tasks that are connected to each other.

Definition 10 ***(Task Graph)*** *A* ***task graph*** *is a directed graph $G = (V, E)$ comprising a set V of vertices together with a set E of directed edges, which are 2-element subsets of V. The vertices consist of tasks.*

In BPMN, the edges of a task graph can be visualized by the Connecting Element Sequence Flow, such that a task graph can be visualized as a BPMN process. In Section 5.4 we described which tasks of the CRISP process are mapped to the process pattern level. From these tasks we can construct a CRISP task-graph visualized in Figure 5.6.

Based on the definition of a task graph, we can now define what it means that a task graph is specialized from another task graph.

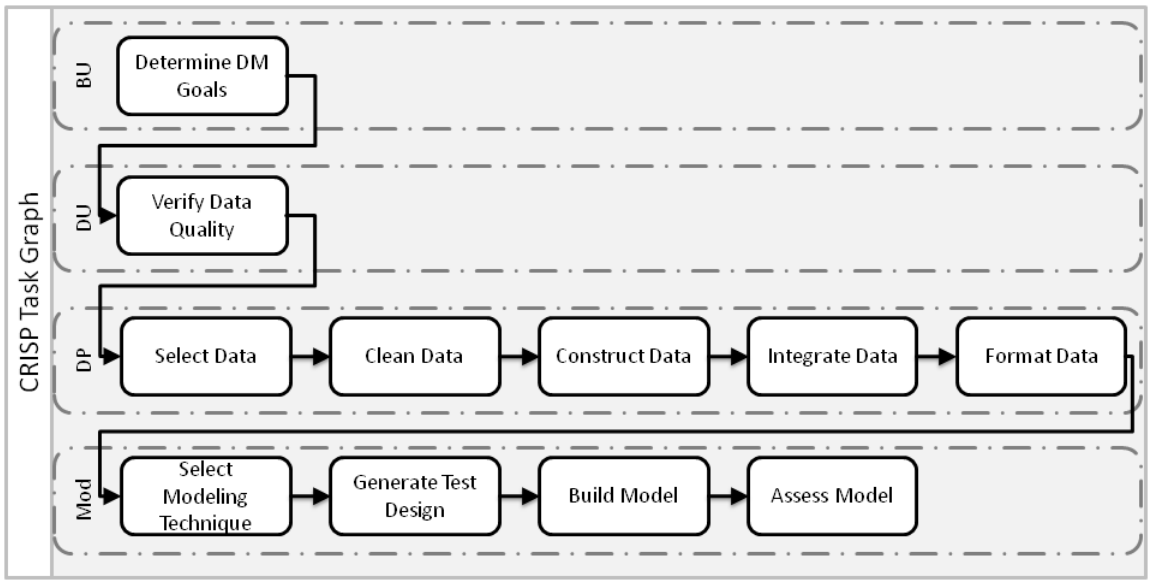

Figure 5.6.: The CRISP task graph.

Definition 11 ***(Specialized Graph)*** *A task graph G_s is specialized from a task graph G, if G_s can be constructed from G by either*

- *replacing a task of G by a specialized task, or*
- *replacing a structural task of G by a task graph which contains only tasks from the configurable or executable level, or*
- *following a finite sequence of the steps as described above.*

Now, we can give the definition of data mining process patterns.

Definition 12 ***(Data Mining Process Pattern)*** *The CRISP task graph is a* ***data mining process pattern****. Every specialization of this process pattern for an application according to Definition 11 is also a data mining process pattern.*

Definition 13 ***(Executable Data Mining Process Pattern)*** *An* ***executable data mining process pattern*** *is a pattern whose tasks are specified to the executable level.*

An executable data mining process pattern contains enough information to transform it into an executable process in a process environment. Further graphical elements from BPMN can be used for defining process patterns in more details. Figure 5.7 presents an example of a process pattern modelled in BPMN, based on the adapted CRISP process from Section 5.4 as most general pattern (see Section 2.2.3 for details on BPMN). The generic CRISP tasks are visualized as tasks in a pattern that splits up into a pool for manual tasks including checking of requirements, and two pools for model building and model application that could be executed automatically. The process is meant to be

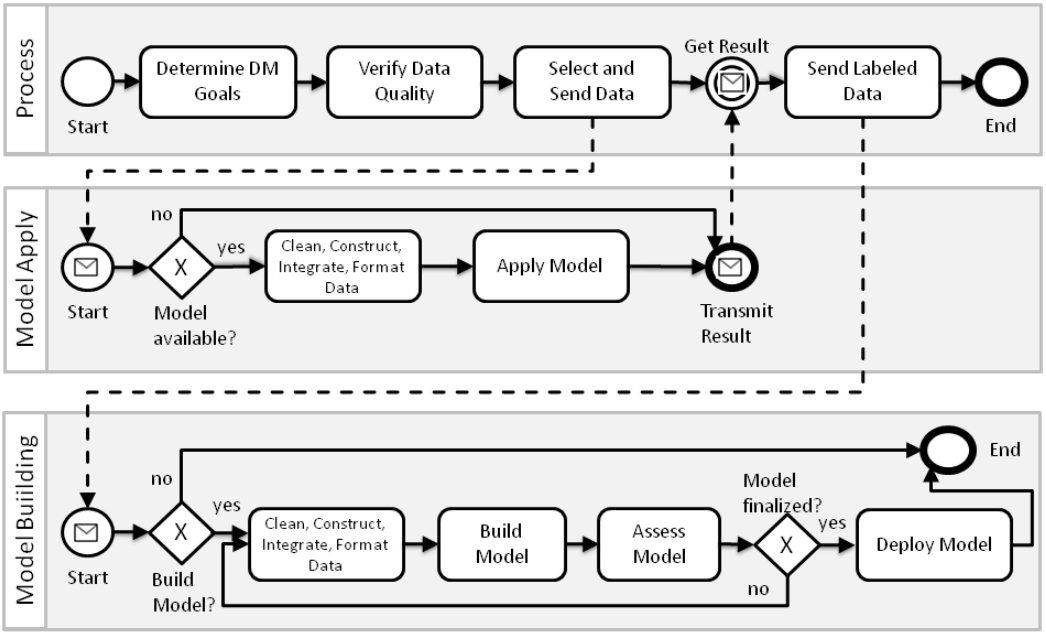

Figure 5.7.: A general pattern modelled in BPMN based on a CRISP process.

executed for new data records individually and starts with a task for checking the data mining goal in the requirements pool (which could be, e.g., prediction based on supervised learning). Then, the data quality of the new data record is checked. After that, the data record is sent as input to the model apply pool. First, it is checked if the model to apply already exists. If not, the sub-process of the model apply pool ends and the process continues in the requirements pool. If the model exists, the data record is preprocessed. After that, the model is applied to the data record. Finally, the result is transmitted and the process continues in the requirements pool. The next task is to send labelled data as input for the model building pool. There, it is decided whether to (re)build the model. E.g., there could exist a rule which decides to (re)build a model if 100 new labelled data records have been sent. If the model is not build, the process ends. If it has to be build, the data is preprocessed, and the model is build and evaluated. If the quality of the model is sufficient, the model is deployed and the process ends. If it is not sufficient, the process steps back to the preprocessing task.

The presented pattern was described on a very abstract level. Examples of a more concrete data mining process patterns will be given later in Sections 5.7, 5.8 and 5.9.

5.5.2. Reuse with Data Mining Process Patterns

Data mining process patterns are designed to support the reuse of data mining processes. Figure 5.8 gives an example of how the reuse of data mining processes is supported. User A holds a workflow that solves a certain analysis problem. To support the reuse, he creates a data mining process pattern from his workflow. This is done by abstracting tasks that are not reusable directly according to the task hierarchy and to model the assumptions

and prerequisites. As he is the only one who knows all assumptions and details of his workflow, he is the right person to perform the abstraction. Other users do not have detailed knowledge on this, so it is harder for them to collect the correct assumptions and to abstract the tasks. User B, who wants to reuse the solution of user A, takes the pattern, checks the prerequisites and assumptions, and creates a workflow by specializing the abstract tasks according to his specific needs.

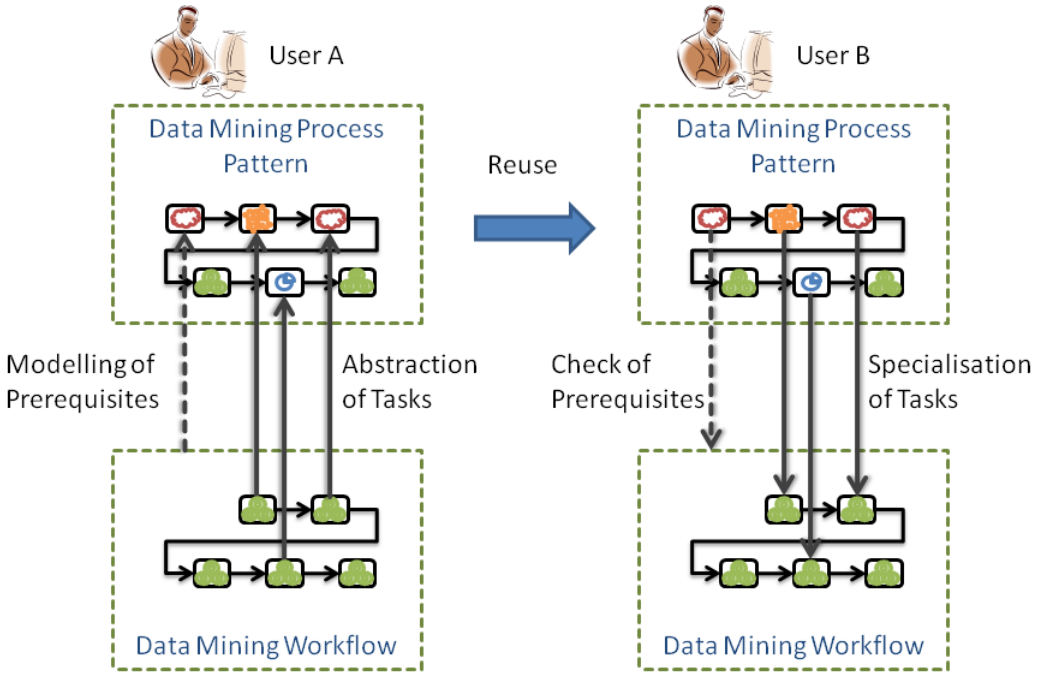

Figure 5.8.: Procedure of reuse with data mining process patterns.

In the following, we will give details on creating process patterns (responsibility of user A) and how to apply process patterns (responsibility of user B).

Creating Process Patterns

Data mining process patterns are created by abstracting parts of an existing data mining workflow. Similar to the specialization presented in Section 5.5.1, the abstraction is done according to the presented task hierarchy. Executable tasks are abstracted to configurable tasks by explicitly modelling the parameters of the underlying component, script or service, e.g. the number of clusters for a clustering component. This means that the task is reusable, but the parametrization is not. Abstraction to a structural task is done by defining the order of tasks in the process while leaving out information about the details of the tasks. E.g., it can be modelled that a quality control task is necessary before a clustering task, or that a data normalization task has to be performed at a certain step of the process, but the actual tasks are not bound to any components, scripts or services. This means that the connection and order of the tasks are reusable, but the components, scripts or services not. An abstraction to a conceptual tasks is done by textually describing

Task	User Level	Integration Level
use component	configurable or executable	-
develop component	conceptual	configurable
use script	configurable or executable	-
develop script	conceptual or structural	configurable
use workflow	configurable or executable	-
develop workflow	structural	structural

Table 5.1.: Task levels from user's and technical point of view.

what needs to be done at a certain step in the process, but components, scripts or services including their connections are not reusable. E.g., a component could be usable just for a certain data type.

We can argue that the choice of the different levels of the task hierarchy presented in Section 5.5.1 makes sense if we map those to the capabilities of users described in Section 5.2. We distinguish between the user's point of view and the point of view from the technical integration of new developed components, scripts and workflows. Tasks that involve using existing components, scripts and workflows are always configurable tasks if they are parametrized, or executable tasks if the parameters are already specified. For the development of components, scripts or workflows, this is different. Although the task for creating a component is at the conceptual level, from the technical point of view the integration of the component is a configurable task, as the component needs to be described by metadata passed as parameter to a service that grid-enables the component (see our contribution from Section 3.3). This is similar for scripts. The task for creating a script is at conceptual or structural level, but from the technical point of view of integrating the script it is a configurable task, as the script can be passed as parameter to a single service (see our contribution from Section 4.4). Thus, by the solutions presented in Chapters 3 and 4 the complexity of the integration is kept at a low level and requires less knowledge from the users. The development of workflows remains at structural level. Table 5.1 presents the task levels from the user's and from the technical point of view.

Process patterns can also be created based on information from a data mining paper [148]. Data mining solutions are often worked off when creating a publication about the solution. This work could be used for the process of generating a data mining process pattern. Papers on data mining solutions consist of a lot of information on requirements, approaches, related work, literature, examples, configurations, pseudo-code, results, summaries, etc. Within this, typically a lot of information is included which can be used for creating a data mining process pattern. Information that is not useful for the creation of a process pattern has to be ignored. This includes, e.g., information on related work, literature and examples, as this information is related to other patterns. In addition, summaries and other redundancies have to be left out, as the process patterns follow a more formal, structured approach. Experiments and results are also not important for creating a process pattern, as the process pattern is only focused on the process of the data mining solution. The remaining parts of the paper, which hold the information useful for the pattern, can be transformed into tasks of the data mining process pattern depending on

their relation to the CRISP phases as well as how precise they are described with respect to the process pattern. Hence, the structure of a data mining process pattern is based on the information contained in the paper and on the level of abstraction in which it is presented. Figure 5.9 visualizes the approach.

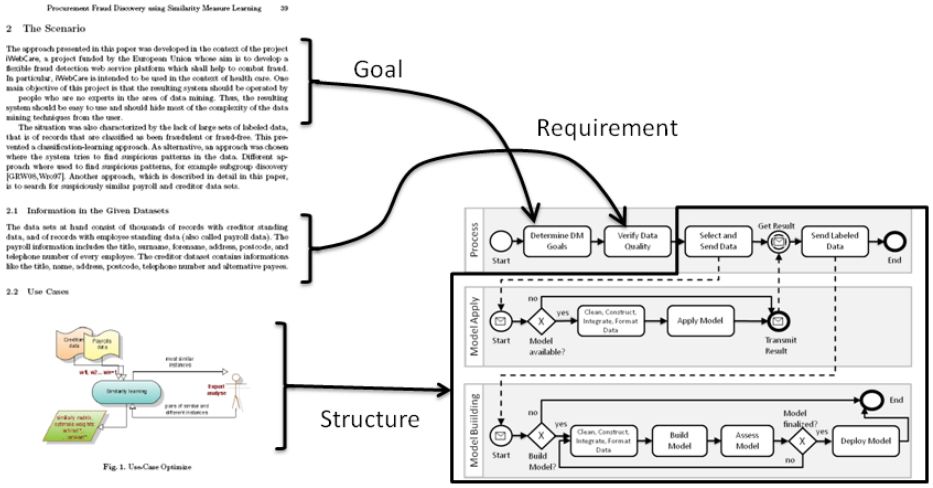

Figure 5.9.: Mapping the information of a paper to the generic CRISP pattern.

The approach of creating a process pattern out of information contained in a data mining paper can be summarized as follows:

- Remove information on related work, literature, examples, results, summaries and other redundancies.
- Transform descriptions of tasks and requirements as regards content into conceptual tasks.
- Transform detailed descriptions of tasks and requirements, instructions and configurations into structural or configurable tasks.
- Transform code and pseudo code into structural, configurable or executable tasks.
- Use figures and use-case diagrams for the arrangement of structural tasks, e.g. lanes, pools and groups of tasks.

Examples of process patterns will be given in Sections 5.7, 5.8 and 5.9.

Applying Process Patterns

In the previous paragraph, we presented details on the abstraction of data mining based analysis processes for a given problem to data mining process patterns. Now, we focus on the specialization of data mining process patterns to new analysis processes.

We describe the steps needed to use a pattern for a given analysis process in a meta-process for applying process patterns. Figure 5.10 visualizes the meta-process and its steps. In detail, the process consists of the following steps:

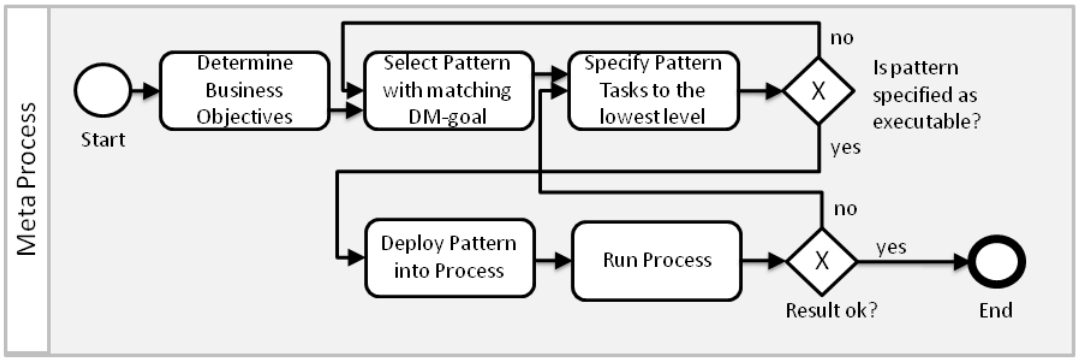

Figure 5.10.: The meta-process for applying a process pattern.

- *Determine Business Objectives:* The first task is to define the business objectives of the application.
- *Select Pattern with matching Data Mining Goal:* After that, a data mining pattern with a data mining goal is selected that addresses these business objectives.
- *Specify pattern tasks to the lowest level:* The tasks of the selected pattern are specified to the executable level according to the task hierarchy. The concretion of a process pattern is not unique and it is not guaranteed that the concretion of a pattern to the executable level is always possible. As the process pattern becomes more specialized by each concretion, it possible to either specify all tasks to the executable level or to detect that it is not possible to specify all tasks to this level in finite steps.
- *is pattern specified as executable?:* If it is observed that the process pattern cannot be specified as executable (all tasks are described at executable level), the meta-process steps back to the task of choosing a new process pattern. If the process pattern is executable, the meta-process steps on.
- *Deploy process pattern into a process:* Assuming that an adequate process environment exists, the process pattern is deployed into an executable process. As the pattern is described as executable, it includes the necessary information for creating an executable process or workflow. However, it is not guaranteed that the process or workflow is really executable in the execution environment, as checks for correctness of components, connections of components, types etc. are not possible beforehand.
- *Run process:* The process is executed in the process environment and performs the analysis. The result is either the result of the analysis or an error.
- *is the result ok?:* If the result is verified as satisfying by the user, the meta-process is finished. If it is not satisfying or if an error occurred, the meta-process steps back to the task of finding a new specification.

The meta-process of applying a process pattern has a set of data mining process patterns as input and an executable process as output. Although the individual steps of the meta-

process can be completed in finite steps, it is not guaranteed that the meta-process ends, as the concretion of pattern tasks can result in various solutions.

The steps of the meta-process still include a lot of manual work. However, tool-based support for applying process patterns is possible and might be implemented in the future. Such tools would guide the user through the steps of selecting patterns and specifying their tasks, helping to handle the different abstraction levels of the task hierarchy and to interface with data mining tools.

5.6. Data Requirements in Data Mining Process Patterns

The concept of data mining process patterns presented in Section 5.5.1 allows for the description of data mining processes at different levels of abstraction and covers, in addition to the description of the steps necessary for executing the data mining solution, also the description of requirements and preconditions that need to be fulfilled to apply the data mining pattern to a new problem setting.

In Section 5.5 we presented the task hierarchy, which included tasks at the conceptual level for the description of requirements. However, tasks at the conceptual level include a lot of manual work. It would be beneficial to describe tasks at lower levels to better support the users in reuse. In the following, we focus on the important special case of data requirements. We show that in this case we can transform conceptual tasks into configurable tasks if there is additional information available on the data in form of an ontology. Thus, we can further extend the support provided by the patterns by making use of external knowledge.

In the context of data mining, the main question that arises is if the data that is available throughout the analysis process makes sense for the data mining process pattern. For example, we could have a specific service S that executes a data mining component in a process pattern. Then, we could include a conceptual task in the pattern that states as text that the input must be of a specific format. This would allow to reuse S, given that the preconditions from the textual description of the conceptual task for the precondition check are met. For an application domain with well structured data it can be shown that this problem can be addressed by combining the concept of data mining process patterns with the concept of semantic mediation of data sources [140].

Data mining process patterns only enable the specification of requirements in general, but do not specifically address requirements and prerequisites to data. As described in Section 2.3, ontologies can help to model data. There exist numerous ontologies modelling specific areas of the biomedical domain. For example, Gene Ontology (GO) provides "structured, controlled vocabularies and classifications that cover several domains of molecular and cellular biology" [68]. The Foundational Model of Anatomy (FMA) ontology contains a model of the human body system from the molecular to the macroscopic levels [108]. The ACGT Master Ontology (MO) represents the domain of cancer and related clinical trials [23]. Thus, the data can be described by ontologies at the conceptual level. Such ontologies can be used for efforts on data integration based on semantic mediation components to provide uniform access to datasets [140]. Semantic mediation components can be used for querying data based on an ontology at the configurable and

executable level. By this, the integration of heterogeneous data sources can be facilitated.

Since the data mining process patterns need to include semantic preconditions about data, we propose to implement those semantic preconditions with queries to a semantic mediation component. This means to combine the concept of semantic mediation of data sources with the concept of data mining process patterns to support the reuse of data mining based analysis workflows in the areas of medicine and bioinformatics. The basic idea is that both data mining process patterns and semantic mediation components are based on a translation of a high-level, semantically rich representation of the workflow and the data to the actual executable code and input data. By combining these two approaches, both data and the data analysis can be formally represented at a higher level of abstraction while guiding the user in a stepwise concretion of the data description and pattern to an executable workflow (see Figure 5.11). We are not interested in the implementation of a semantic mediation component in particular, but in the idea of translating between a higher-level, semantically rich query language and an actual dataset or database. Furthermore, there already exist semantic mediation components, e.g., the ACGT semantic mediator [140] (see also Sections 2.4.2 and 4.7).

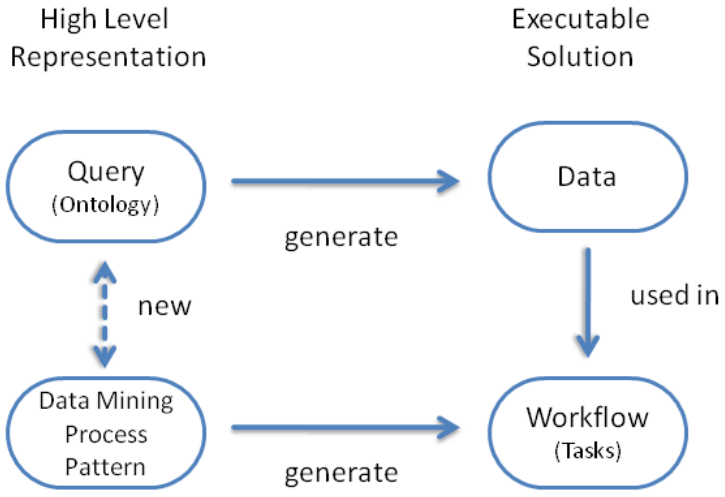

Figure 5.11.: Mapping high level knowledge about clinical data analysis to concrete solutions.

In the following, we will describe our approach on data semantic enriched data mining process patterns. We aim at data semantic aware data mining process patterns which represent patterns with data preconditions. These support the formalization of the description of the data input for a process pattern. The aim of this approach is to further support the reuse of analysis processes and thus to speed-up the development of new solutions. We focus on semantic rather than on syntactic issues. In detail, we extend the concept of data mining process patterns and the task hierarchy by tasks that describe data and data preconditions. These tasks can be realized as querying components in the pattern for checking data requirements or for data access based on the underlying semantics. By this we bind knowledge about medical data, as defined by medical ontologies, together with knowledge about medical data analysis, as formalized by data mining process patterns.

First, we will describe which data requirements in data analysis processes we want to address. Then, we will introduce our approach for describing data semantics via medical ontologies. Finally, we give an example on how our approach can be applied in a scenario in bioinformatics.

5.6.1. Data Requirements in Data Analysis Processes

When we describe the pre-condition for checking whether a certain data mining approach can be applied, it is very important to be able to describe for which data this approach makes sense. Therefore, we want to be able to automatically check if a dataset fulfils some pre-condition or not (and if it does, of course we then want to access the data later). So, assuming there is a single dataset and 100 possible process patterns. The goal is to be able to automatically filter out those process patterns that can be applied to the data in the first place. In particular, we need not only to describe the syntactic format (e.g. "only numbers"), but also we would like to be able to describe the content (e.g. "this approach makes only sense for gene expressions").

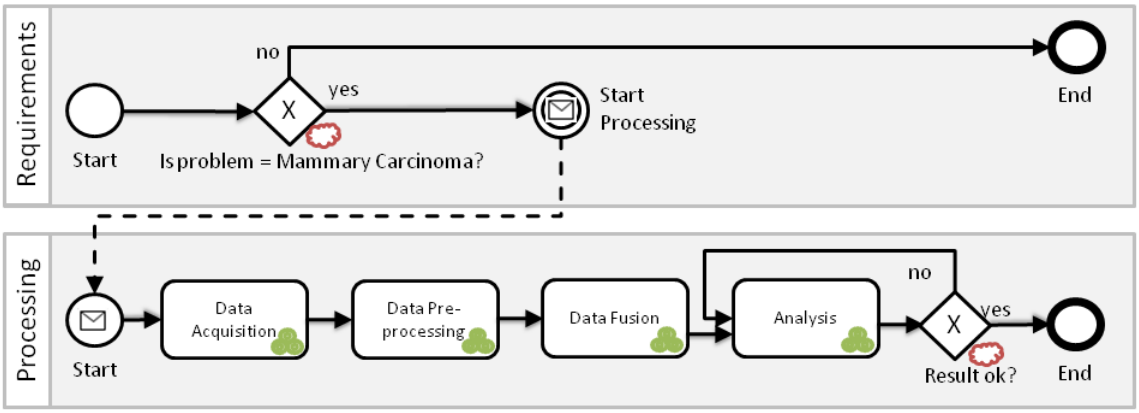

Figure 5.12.: Example of a data mining process for analysis tasks with preconditions in bioinformatics from [140]. The processing tasks are mapped to executable scripts based on the R environment.

Typically, workflows including data mining components can only work on specific data. Figure 5.12 presents an example of a data mining process which was developed based on scenarios from the ACGT project [34]. In addition, there are assumptions made on the data when applying a workflow. Thus, the key point is that we want to allow for the formalization and automatic testing of requirements. For data mining tasks we can characterize the type of data requirements and pre-conditions into semantic and syntactic. In our work, we focus on the semantic part. However, when using tools like an ontology based semantic mediator to query data sources in practice, the problem of syntactic requirements is already partially solved [140]. These are some examples for semantic and syntactic preconditions:

- semantic: the values for attribute x represent gene expressions.

Such requirements are addressed by using a medical ontology. Since ontologies arrange classes in a hierarchy, it would be feasible to check whether a class C inherits from the class *gene expression*.

- syntactic: all attributes have to be numeric, the label attribute has to be binary, missing values are not allowed for attribute x.

Such requirements are addressed by the underlying database schema of the ontology or could be formulated inside a SPARQL query [5]. When selecting an attribute of a class of the ontology, the data type is automatically defined by the underlying schema. Missing values could be addressed by a filter in the corresponding query.

5.6.2. Describing Data Semantics as Query Tasks

Basically, our idea is that - assuming we have a way to perform semantic queries - we could replace a pre-condition such as "the data needs to contain information about blood pressure" by a formal query that returns a non-empty result. By this, we use the semantic description of the data as a bridge between the high-level description of the content that we need and a low-level description of the actual data.

We assume that we work on relational data in table format. We further assume that there is an ontology O available for describing the domain specific semantics of datasets. In detail, this means that the semantics of a dataset are described correctly if each column of our data table is mapped to a concept of the ontology.

In Section 5.5.1 we introduced our task hierarchy. In the context of data requirements, the most specialized task for describing data is a task that is automatically executable and returns a non-empty dataset. The most general task is a task that only describes in natural language how the data has to look like to be able to apply the analysis. At a level in-between, there can be tasks specified which do not define a concrete dataset, but make use of concepts from an ontology. These could, e.g., describe the columns of a data table. Such tasks are more general, if the used concepts from the ontology are more general, and more concrete if more concrete concepts are used. From the description based on an ontology, a query can be generated that can be executed against some data source with the help of a semantic mediation component.

Definition 14 ***(Data Precondition Task)*** *A* ***data precondition task*** *t for data described by an ontology O is defined by a query q, which is specified according to the ontology O. A data precondition task is defined as fulfilled if the semantic mediation component returns a non-empty result on task execution. The query tasks are executable tasks for an existing semantic mediation component. They are arranged in a hierarchy as follows: query task a is more general than query task b, if the concepts of the ontology used within the query of a are more general than the ones of b. Query task a is more specific than query task b, if the concepts of the ontology used within the query of a are more specific than the ones of b.*

Using an ontology and a component for semantic mediation, queries can be written and can be translated and executed against some data source. In a data mining process

pattern, such queries can be modelled as query tasks based on already existing components for semantic mediation and query editing.

Definition 15 ***(Semantic Enriched Data Mining Process Pattern)*** *We define a* ***semantic enriched data mining process pattern*** *as a data mining process pattern from Definition 12 that includes query tasks from Definition 14.*

A data mining process pattern for a certain analysis task can be applied to a given dataset if all tasks of the pattern can be specified to the executable level, including the data requirement tasks. The latter have to be specified to an executable query that returns a non-empty result.

Generalizability of data requirements means to define the set of all datasets for which the analysis workflow is generalizable. Requirements that are specified as query tasks based on an ontology can be generalized by going up in the hierarchy of the ontology. The process of creating a process pattern from a specific workflow including data precondition tasks is extended by the following step:

- **Generalization of requirements:** Generalize the query of the data precondition task in a way that it matches to the data mining tasks formally (same data types) and as regards content (makes still sense to apply the DM to the data).

Users could be supported by tools which enable the creation of executable workflows from process patterns, e.g. a semantic mediator or a query generator, and which allow for the development of process patterns from a specific workflow, e.g. an ontology browser for navigating through the concepts of an ontology. In the ACGT project [34] (see also Section 2.4.2) the ACGT Semantic Mediator offers a uniform query interface to an array of underlying data sources. In the end, the end-user "sees" a new database with a new schema (the ACGT Master Ontology [23]) which covers the domain of the sources. He can submit queries in terms of this global schema. The query language chosen for the Semantic Mediator was SPARQL [5], which is a standard for querying RDF resources.

5.6.3. Example

In the following we present a scenario for the demonstration of our approach. The scenario provides evidence in the sense that it is possible to describe meaningful requirements by our approach.

The scenario consists of a workflow designed to work on a given dataset (dataset 'A') available through a semantic mediation component. The workflow contains queries that retrieve and analyse data from the dataset (see Figure 5.13, all tasks are specified to the executable level). In this case, the dataset contains data about mammary carcinomas (diameter, ER status and nodal status of the carcinoma of each patient). A second dataset (dataset 'B'), also available through the semantic mediation component, contains the same type of data for sarcomas (another class of neoplasm). This scenario shows how it is possible to automatically check the suitability of using the existing workflow with the new dataset by using the semantic mediation component functionalities and having an ontology as database schema.

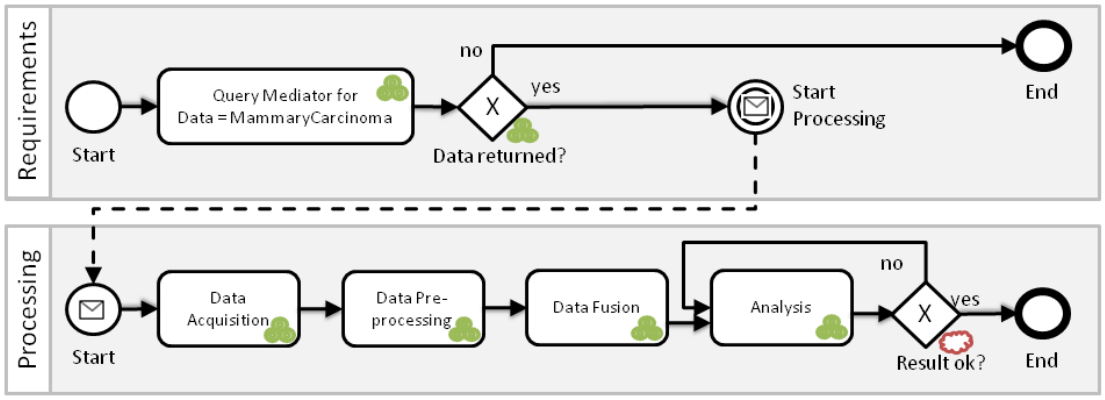

Figure 5.13.: A process with a query task.

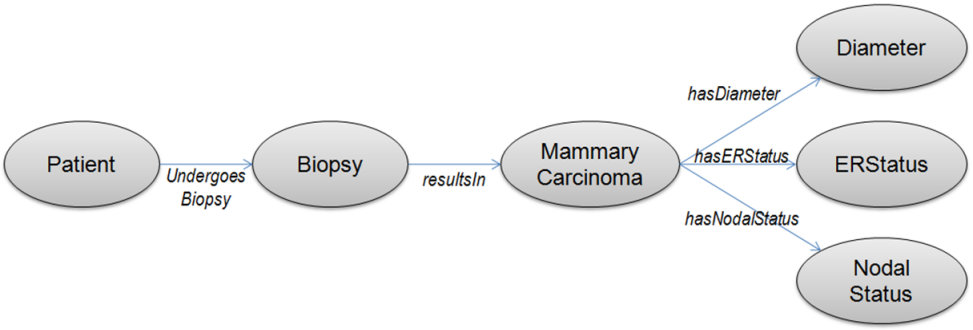

Figure 5.14.: Part of the ontology mapped to the schema of a dataset. Incoming queries containing this view will have translation to the dataset, and therefore will retrieve data contained in it.

The data which is accessible in each dataset depends on the mappings defined for that source. These mappings express in a formal manner pairs of views (in case of RDF models, paths composed by classes and relations) which have the same meaning. This way the semantic mediation component is able to correctly translate incoming queries in terms of the ontology into queries that can be answered by each data source. For our example, dataset 'A' is mapped to views in the ontology which express data of mammary carcinomas of patients. Figure 5.14 shows these views.

The query defined for the existing workflow includes these views, so when the workflow is executed the mammary carcinoma data is correctly retrieved. In fact, the query does not contain the exact previous view, but a generalization of it. The MammaryCarcinoma class in the mapped view is substituted for the Neoplasm class in the query. MammaryCarcinoma inherits from Neoplasm, therefore the mapped view effectively answers the query in the workflow. This process of generalization is what allows the reuse of workflows in a broader spectrum of datasets than it was designed for. Indeed, dataset 'B' is mapped to a view in the ontology which relates patients with data about their sarcomas. Just like with dataset 'A', the semantic mediation component is able to detect the inheritance relation between the views. The data in dataset 'B' will be used to answer to the query in

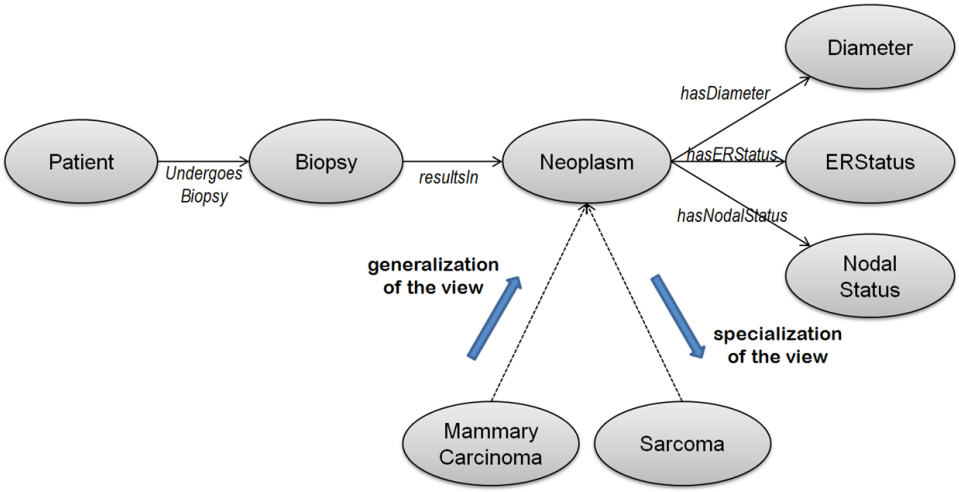

Figure 5.15.: Mappings for both datasets inherit from the query contained in the workflow. This generalization process, and the subsequent concretion carried out in the semantic mediation component allows automatically checking if another dataset is compatible with the workflow.

the workflow. Figure 5.15 illustrates this situation. The generalized pattern is visualized in Figure 5.16.

In fact, the semantic mediation component offers the possibility of checking whether a given query has a translation to a specific dataset. This is of course independent of whether the dataset contains any actual data, or is empty. The semantic mediation component allows verifying the possibility of using an existing workflow with different datasets, even when such sources have not been loaded with data yet. The only prerequisite is to have the dataset mapped to the ontology. Thus, by the approach of defining data requirements as query tasks to the semantic mediation component in the process pattern, the reuse of workflows is facilitated.

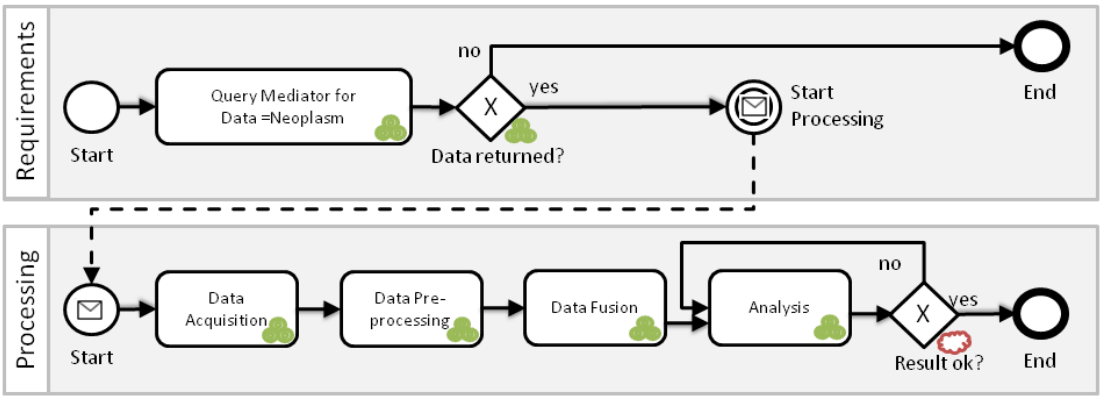

Figure 5.16.: Generalized pattern.

5.7. Case Study Process Pattern of a Clinical Trial Scenario

In this section we present a case study for the data mining process pattern approach based on the clinical trial scenario presented in Section 2.3.5. We will show how to create a process pattern from a data mining script by extracting configurable tasks and how to specialize the process pattern to create a workflow.

5.7.1. The Clinical Trial Scenario

The scenario presented in Section 2.3.5 deals with the analysis of tumour samples with associated gene expression data and clinical features [94, 112]. In the scenario, cancer samples are analysed to find gene signatures which may indicate whether a tumour is malignant or not, and whether the tumour will metastasise or not. The original scenario has been implemented as R scripts in the context of the p-medicine project [94, 112]. In detail, the scenario consists of 6 parts, which could be considered as individual components:

1. Prepare Experiment - import and normalization of genomic data.
2. Quality Control - generation of plots for quality control before and after normalization.
3. Build Environment Structure - generation of data structure including clinical data for the analysis.
4. Find Differentially Expressed Genes - finding the differentially expressed genes and generation of the volcano plot and heatmaps.
5. Create Risk Index - generation of a risk-index for the survival analysis.
6. Survival Analysis - generation of Kaplan-Meier plots for survival analysis based on the risk index.

For more details on the scenario we refer to Section 2.3.5. Please see Section A.3.1 in the Appendix for the original R code of the scenario.

5.7.2. Process Pattern of the Scenario

The first step of creating the process pattern is to identify the individual components which have to be described by tasks in the process pattern. In our case, we split the scenario into 9 R scripts. The code that covers the import and the normalization of the genomic data is represented by the *Prepare Experiment* group including the tasks *ReadExperimentData* and *NormalizeData*. The code that deals with the generation of the 3 plots for checking the data quality is represented by the Quality Control group including the tasks *QC Degradation*, *QC Intensities* and *QC LogIntensity vs. Density*. The code that covers the import of the clinical data and the creation of the data structure for the analysis is represented by the *BuildEnvironmentStructure* task. The code for finding the differentially expressed genes, creating the heatmaps, the risk index and the survival analysis are also represented by the respective tasks *FindDifferentiallyExpressedGenes*,

CreateRiskIndex and *SurvivalAnalysis*. Figure 5.17 visualizes the identification of the components in the script.

The code of the individual components is not directly reusable, as it is part of a standalone R script. The components, which should be reusable as individual R scripts, have to be abstracted to the configurable level to allow for reuse. However, this can be solved by adding headers and footers to the R scripts. The headers are responsible for loading the R libraries needed for each of the split script, which was done once at the beginning of the original script. Furthermore, the headers and footers take care of the data exchange between the split scripts by storing and loading the data of the R workspace. In addition to the headers and footers, it is necessary to specify parameters for the directories where the input data can be read from and where the output data has to be stored to in order to allow for reuse. By this, the components are transformed into configurable tasks. The parameters that have to be configurable are basically the folders in which the input data for the individual components reside and where the results should be written to. The only part that is at conceptual level is the processing of the decision on the quality control, as this has to be performed manually anyway. In the original scenario this was done manually by the bioinformatician. Figure 5.18 visualizes the process pattern of the scenario in BPMN. Please see Section A.3.2 in the Appendix for the R code of the components.

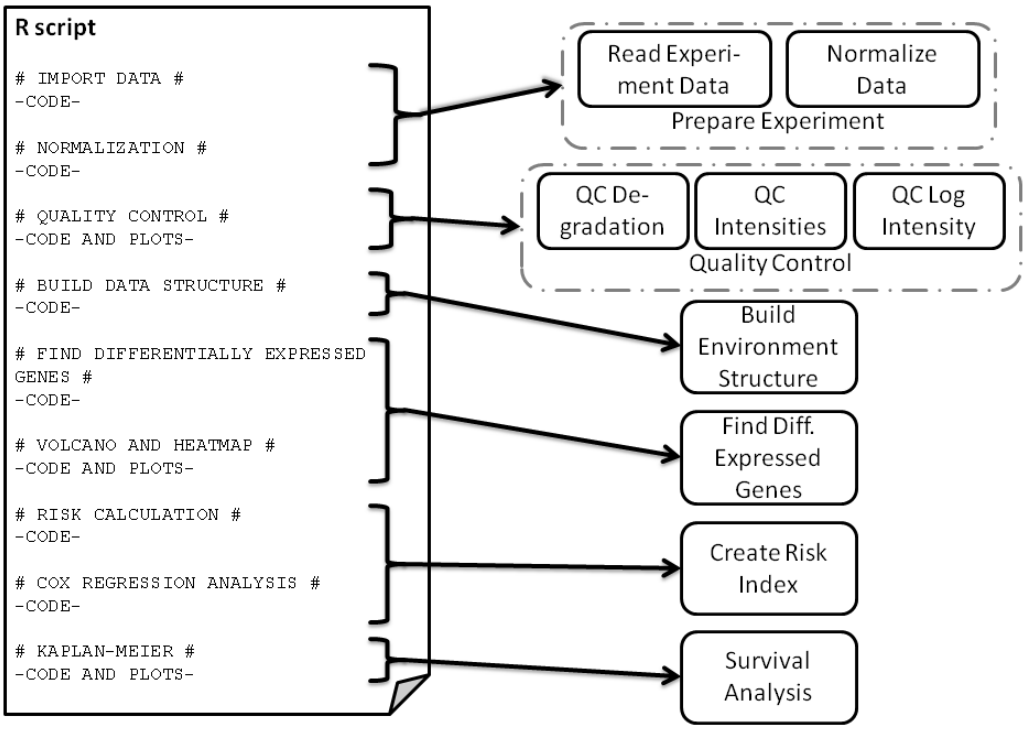

Figure 5.17.: Identification of components and groups of components in the script.

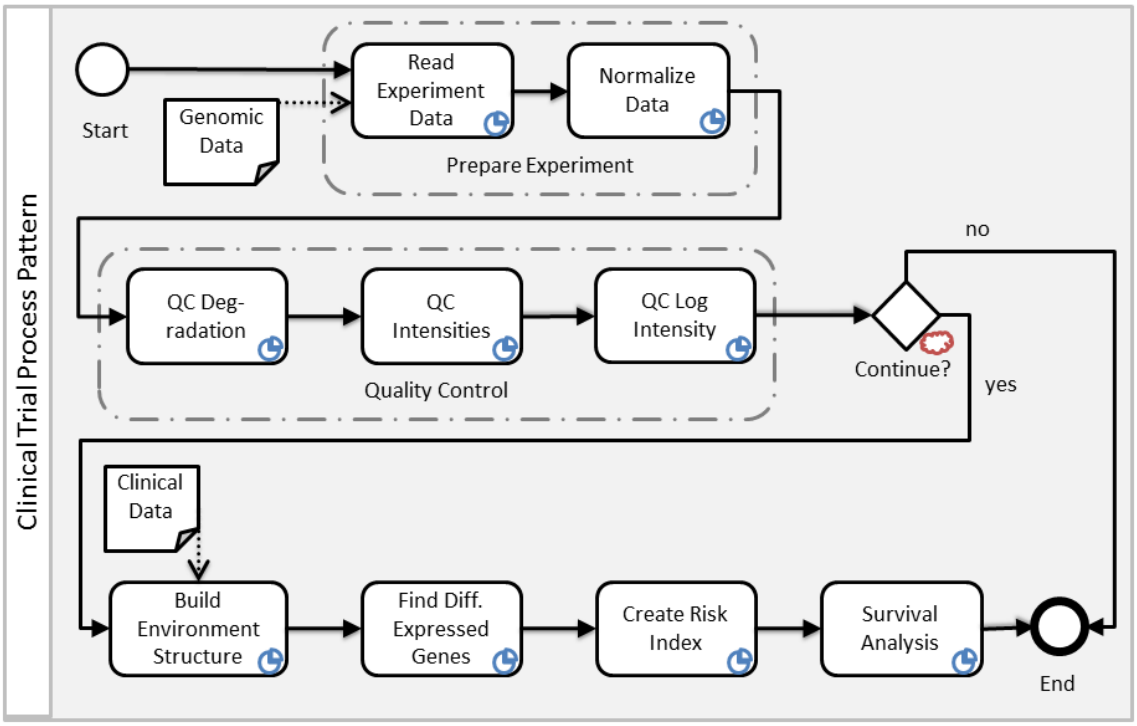

Figure 5.18.: The process pattern of the scenario visualized in BPMN. The components have been transformed into process pattern tasks.

5.7.3. Taverna Implementation

The process pattern created in the previous section can be specialized to create an executable workflow. In our case study, the scenario is implemented as workflow in Taverna [72]. In detail, the overall workflow consists of 6 nested workflows that are connected to each other. The nested workflows represent the tasks and groups of tasks of the process pattern. The complete workflow is depicted in Figure 5.19 for an overview and in Figures 5.20, 5.21 and 5.22 in more details. The R scripts representing the components are attached via the R-plugin of Taverna which allows for the execution of R scripts within a Taverna workflow. These are visualized in dark blue in the figures.

The process starts with the first nested workflow *Prepare Experiment*. It has two input parameters that are passed from the workflow input fields: the path to the input data and a path to which the output is written. The latter is passed to all other nested workflows and R tasks in the workflow, thus making the tasks to executable tasks. Inside the nested workflow two R scripts are executed: *ReadExperimentData* and *NormalizeData*. In the R script *ReadExperimentData*, the datasets, which are based on affymetrix arrays, are read in and imported into variables. The data is accessible under a path that has to be specified as input parameter. In the *NormalizeData* script, the data is normalized. The result returned from the nested workflow is the path where the results are stored. The pink tasks in the workflow are fields containing further information on the execution and completion of the R-scripts. The nested workflow *Quality Control* consists of 3 R scripts,

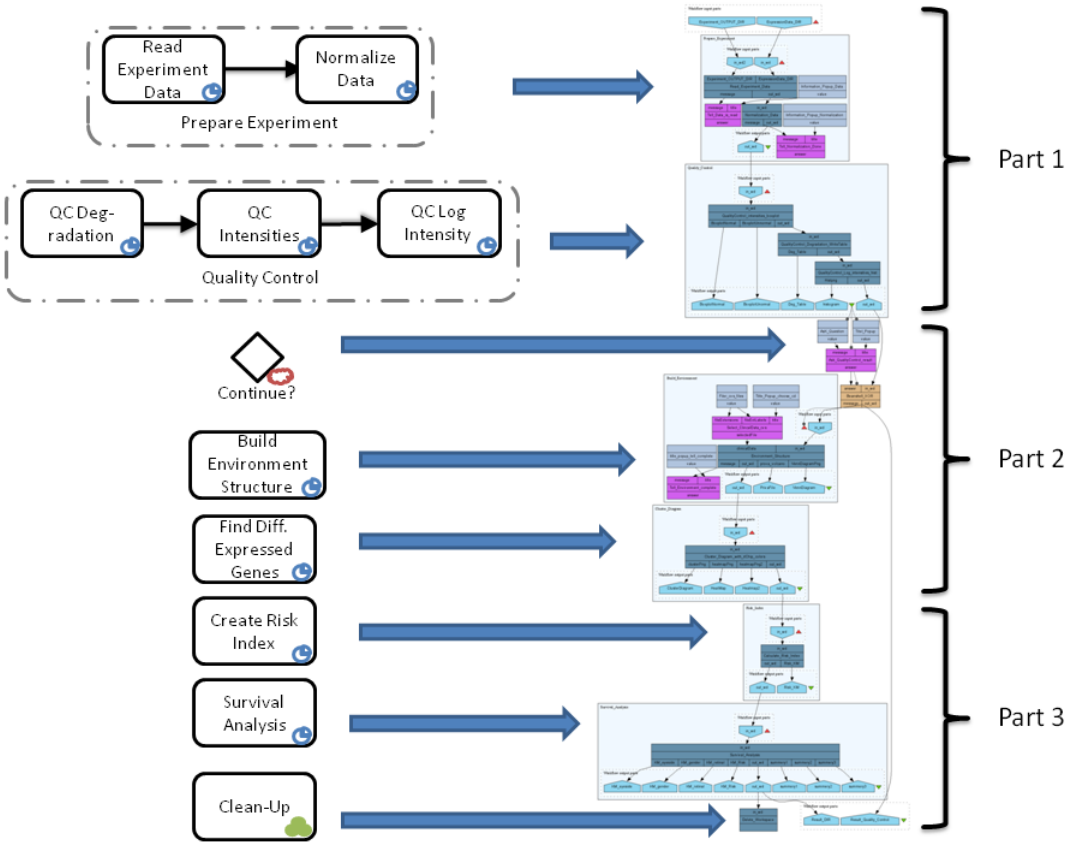

Figure 5.19.: Scenario implemented in Taverna (overview). The tasks and task groups of the process pattern are deployed into nested workflows and workflow tasks in Taverna. Further details can be seen in Figure 5.20 (Part 1), Figure 5.21 (Part 2) and Figure 5.22 (Part 3).

which check the quality of the imported and normalized data. The results are plots which have to be manually interpreted by the user.

After the completion of the quality control step, the user is asked if the the workflow is to be continued or not via an input field in the UI. This represents the conceptual task of the process pattern.

If the data quality was evaluated as sufficient, the next nested workflow that is executed is *BuildEnvironment*. The clinical data are read and the data structure for the analysis is created and visualized in plots. After that, the nested workflow *ClusterDiagram* is executed. It continues by finding the differentially expressed genes between established sub-groups of samples belonging to classes of interest and produces the volcano plots and heatmaps that provide information about which genes have an increased activity. Subsequently, the nested workflow *RiskIndex* is executed. It creates a risk index that is used for the survival analysis. Finally, the nested workflow *SurvivalAnalysis* is executed, where Kaplan-Meier plots are created based on clinical features and the risk index.

The output of the workflow is the directory containing the outputs of the R tasks and the indication on whether the quality control was successful. The workflow furthermore includes a clean-up task that removes intermediate results from the output directory.

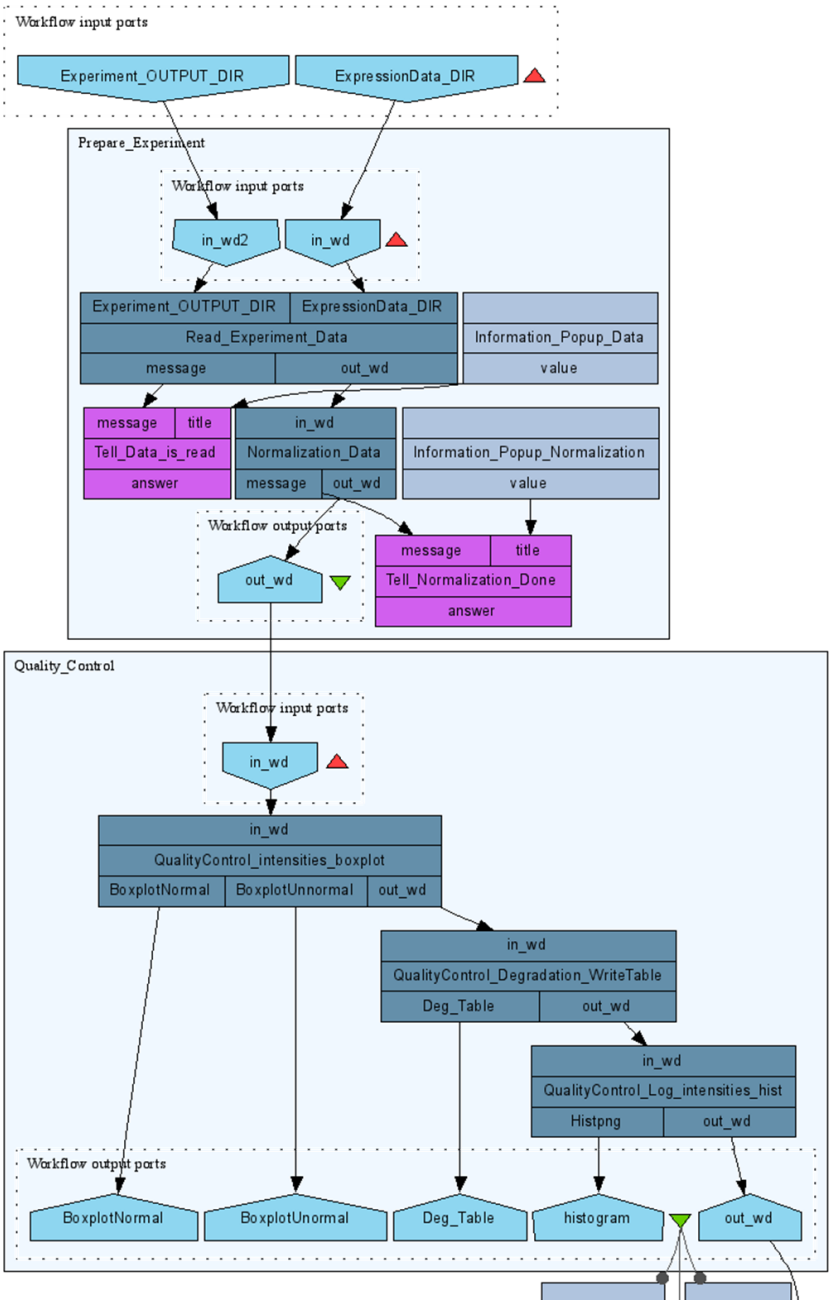

Figure 5.20.: Scenario implemented in Taverna (Part 1 - PrepareExperiment and QualityControl).

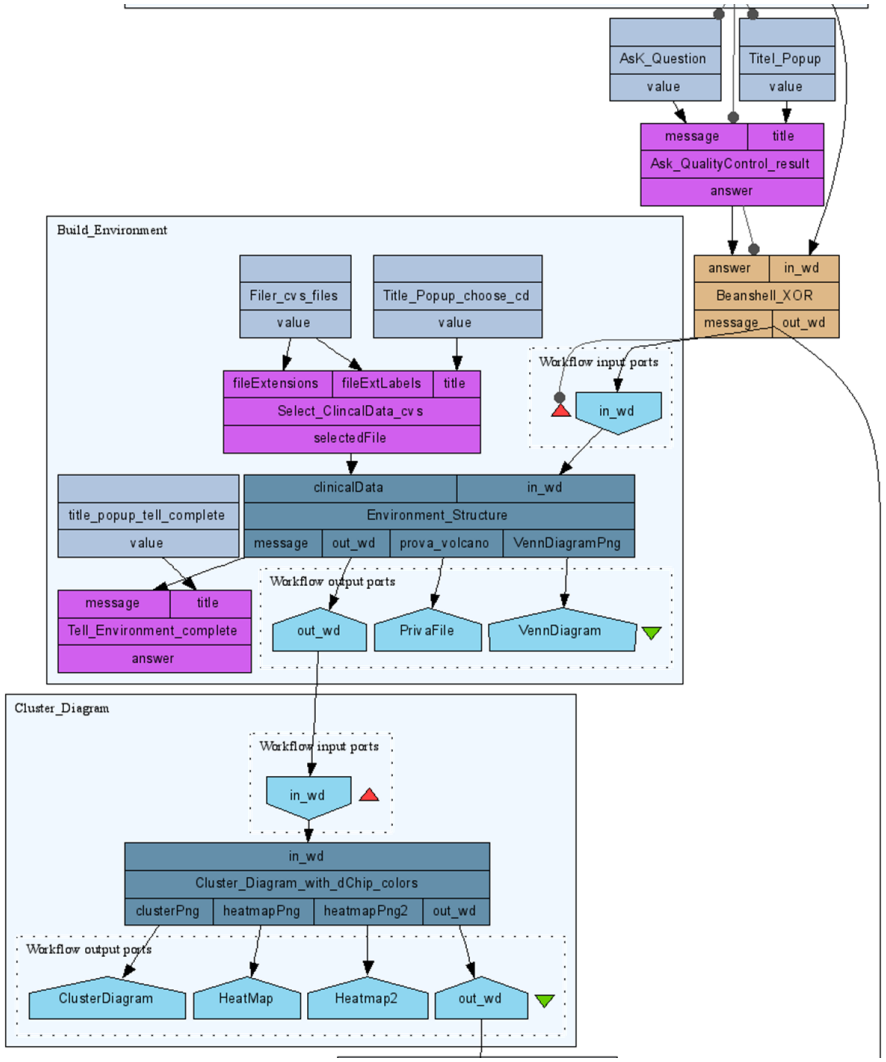

Figure 5.21.: Scenario implemented in Taverna (Part 2 - Conceptual task, BuildEnvironmentStructure and CreateClusterDiagram).

5.8. Case Study Process Pattern of a Meta Analysis Scenario

In the following, we will present a case study based on the multi-center multi-platform (MCMP) scenario of the ACGT project [40]. The aim of the scenario is to assess the variability in gene expression microarrays, and the reliability of the prognostic and predictive profiles obtained from this technology, when the arrays are performed using different technological platforms and at different organizations (centers). We demonstrate how the process of the MCMP scenario can by abstracted using structural and conceptual tasks to create a process pattern that describes the general way how bioinformaticians work in scenarios dealing with meta analysis.

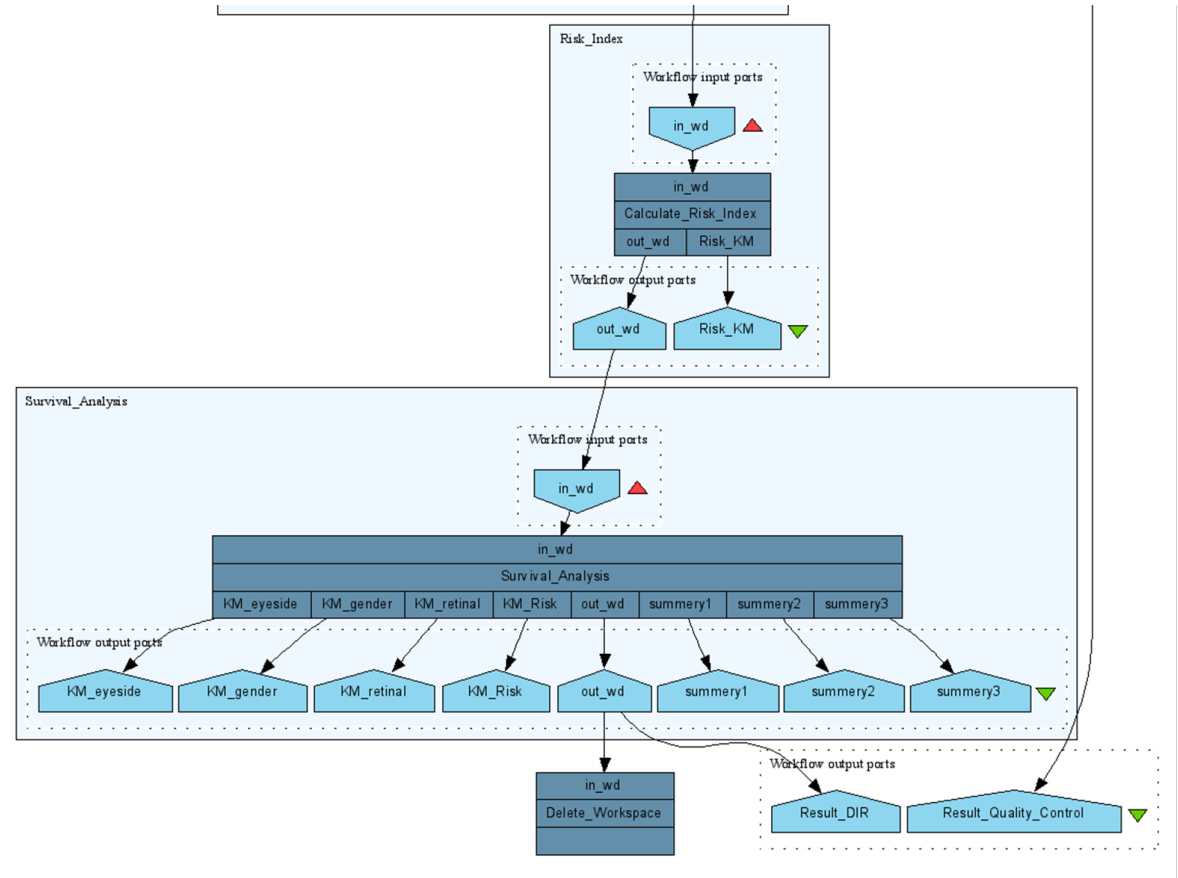

Figure 5.22.: Scenario implemented in Taverna (Part 3 - CreateRiskIndex and Survival-Analysis).

5.8.1. The Multi-Center Multi-Platform Scenario

Summarizing the research background of the scenario, it is assumed that biopsies are collected from patients registered in two organizations (multi-center) and that each organization is using a different microarray platform (multi-platform), namely Affymetrix [74] and Illumina [75], to measure genes expression in the samples. In addition, the classical clinical parameters associated to each patient are available in relational databases. Figure 5.23 gives an overview over the scenario. All private patient data were anonymized prior to their integration in the ACGT environment. The anonymized data is retrieved from the databases of each organization. After that, it is preprocessed, normalized and then combined. Based on that, the analysis is performed, e.g. to check the repeatability of the expression signal across the platforms. A more detailed description of the underlying study can be found in [40].

The workflow of the MCMP scenario has been implemented with the ACGT workflow environment [24] (see Section 4.7 for details on the ACGT environment). In detail, the scenario consists of the following:

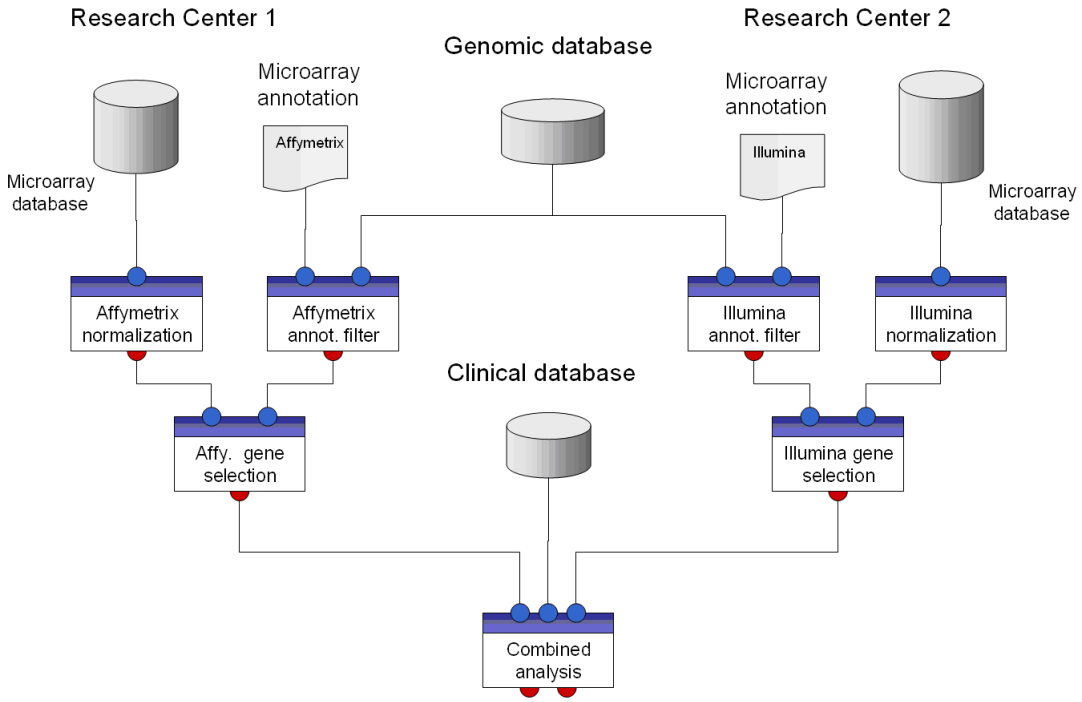

Figure 5.23.: Overview over the MCMP scenario.

- **Inputs:** Two microarray databases have been integrated into the ACGT data infrastructure (Center 1 with Affymetrix data and Center 2 with Illumina data). In addition, a simple SQL database describing the mapping between patient identifiers and Affymetrix and Illumina chip identifiers is available.

- **Data Import and Normalization:** The process begins by retrieving microarray experiments that are stored as files in the grid file system and preprocessing them in two parallel branches based on the platform used.

- **Filtering and Gene Selection:** A feature selection is performed in each of the branches to extract the most informative genes. The platform mapping and data preprocessing is done by adopting existing standards. Specifically, the probe sequences from each platform are mapped to the NCBI RefSeq database (`http://www.ncbi.nlm.nih.gov/RefSeq/`, genomic database). Probes for which a match is not found are filtered out. A probe is considered as having a match if it perfectly matches a RefSeq sequence and does not perfectly match any other transcript sequence with a different gene ID (`http://jura.wi.mit.edu/entrez_gene/`). The common set of matching probes between Affymetrix and Illumina is then considered for further analyses and the full annotation is retrieved based on the RefSeq ID.

- **Combined Analysis:** At the final step, the results of the two parallel sub-processes are combined in an analysis task that also uses the results returned by the Semantic

Mediator based on the Master-Ontology-expressed query for the patients' clinical data. The platform comparison is based on different criteria:

1. Reliability of gene expression measurement: comparison of gene expression as measured by Illumina and Affymetrix platforms in the 73 samples.
2. Reliability of patient classification: comparison of published gene expression classifiers and their performance in patient classification when using the two platforms.
3. Reliability of biological/clinical findings: comparison of biological content of the gene expression classifiers obtained using the two platforms.
4. Feasibility of a combined-platform trial: simulation of a combined platform study and assessment of feasibility of multi-platform studies.

Figure 5.24 visualizes a process pattern modelled in BPMN that can be abstracted from the MCMP scenario, where the tasks for gene filtering and gene selection have been abstracted to structural tasks (Data Filtering).

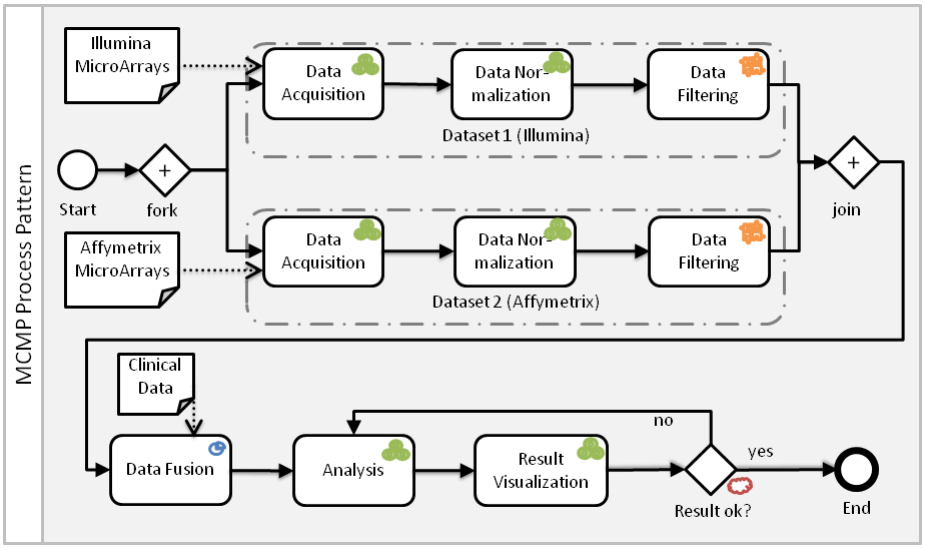

Figure 5.24.: A process pattern for the MCMP scenario.

5.8.2. Abstract Process Pattern for Bioinformatics Processes in p-medicine

As described in Section 2.3.1, there emerge processes of general interest in the area of bioinformatics which would profit from standardization and reproducibility. In the following we show that the pattern developed for the MCMP scenario can be further abstracted and extended to cover the description of a general process used for many bioinformatics scenarios - the meta analysis of public datasets.

In the following, we will focus on meta analyses of public datasets in the context of cancer-related bioinformatics scenarios as tackled by the p-medicine project [94]. These

analyses typically involve data coming from gene expression and some clinical or demographic data.

The genomics data can be acquired from public repositories, e.g. GEO [56]. Often, microarray data can be retrieved already pre-processed and normalized. However, database and data format, pre-processing and normalization methods are different depending on the database, on the time when the datasets were submitted or on the submitter of the dataset. Thus, in meta-analyses using public datasets these steps typically need to be re-performed so that datasets are coherent.

Clinical and demographics data are typically provided by the repositories that contain the gene expression data, via publication in peer-review journals, or by the authors of the datasets. This part of the data is usually less standardized due to lack of standardization in the clinical context. E.g., there exist different names for stages of diseases in different countries. Thus, the retrieval of clinical and demographics data can require pre-processing which might be difficult to perform automatically.

A meta analysis process is typically performed in the following way:

- **Data Acquisition step**: genomic, clinical and demographic data is acquired from different data repositories.
- **Data Pre-processing step**: genomic, clinical and demographic data are pre-processed to ensure that the datasets are coherent.
- **Meta-Analysis step**: depending on the specific question the analysis will generally involve calculating a statistic and its confidence limits in each dataset and summary statistics and confidence limits across all datasets.
- **Results step**: The results will typically consist of a table with numeric values of statistics and confidence limits in each datasets, and the summary statistics.

The meta analysis process is often iterative and can include various methods such as Bootstrapping, leave-one-out, k-fold cross-validation, depending on the specific aims and needs. The general schema of a meta analysis process is described in Section 2.3.1 (see also Figure 2.10).

The MCMP process pattern can be transformed to a process pattern for the meta analysis by further abstracting certain parts. The tasks for data normalization and data filtering, which are executed for each genomic dataset, can be abstracted to a single conceptual task for data pre-precessing for each dataset. Furthermore, the task for data filtering and analysis can be abstracted to a single conceptual task for the meta-analysis. Figure 2.10 visualizes a process pattern for the meta analysis scenario.

5.9. Case Study Integration of Patterns in Business Processes

In bioinformatics, the business processes that exist around the scientific analysis processes are typically not formally modelled inside a workflow. In the following, we will present a case study from the healthcare sector as example on how data mining process patterns can be integrated into business processes.

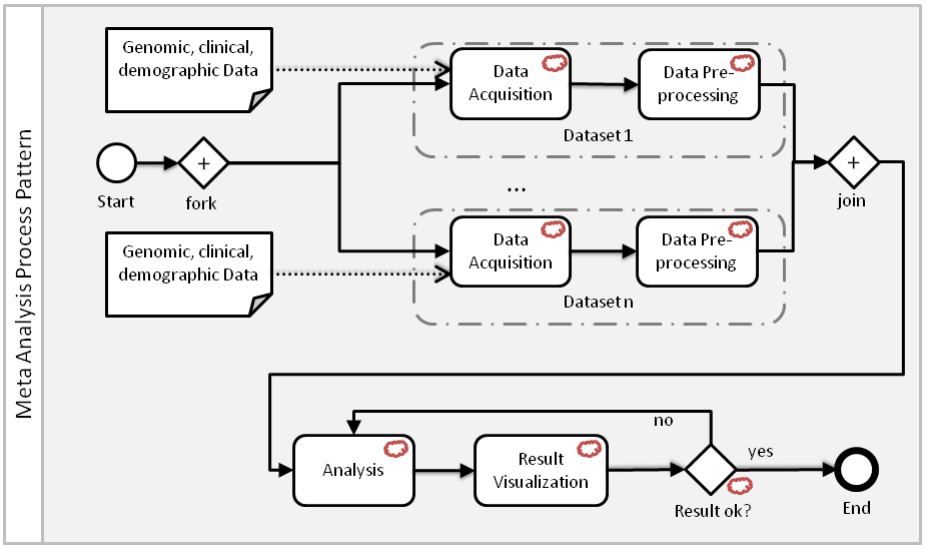

Figure 5.25.: An abstract process pattern for the meta analysis scenario in p-medicine.

5.9.1. Introduction

Motivating examples for integrating a data mining process into a business process are, e.g., the EC-funded projects RACWeB (`http://www.racweb.org/`) and iWebCare [126]. RACWeB aimed at improving the Western Balkans and EU countries' customs efficiency and transparency in risk assessment by enhancing the identification of risk profiles through the utilisation of data mining techniques. In iWebCare, a flexible fraud detection platform was developed in order to ensure quality and accuracy and minimise loss of health care funds in the Healthcare business. The basic idea behind both projects was to develop a web service platform where participating organizations can upload datasets and semi-automatically select an appropriate data mining process. Based on the analysis of the business processes of participating organizations, it was possible to develop generic data mining solutions that can be re-used in similar business processes. Because of the detailed business process modelling it is not necessary to follow the CRISP process step by step again a second time, as much of the relevant business knowledge is already contained in the business process model.

We choose a fraud detection scenario from [78] as case study, as it already includes well specified business processes as well as a data mining solution that allows for reuse. The approach for fraud detection is based on learning similarity measures for data records and is transferable for a generic class of fraud opportunities [113]. The application scenario is based on detecting procurement fraud, e.g. an employee of a company placing an order to another company which is owned by himself. This is done by computing a similarity between employees and company owners based on several features such as name, address or bank accounts. We do not go into more details of the data mining method, as this is not important for understanding our process pattern approach. Basically what is needed to apply this data mining solution to a problem is to first check if the problem is a procurement fraud problem, second to specify which attributes to be used for the similarity,

and third to connect the inputs and outputs of the data mining process. For all other steps ready-to-use code is already available.

In the following, we will present how to create a process pattern from the data mining scenario, how to integrate it into a business process and how to reuse it in another business process.

5.9.2. Creating a Pattern

In Section 5.5.2 we presented an approach on how to create data mining process patterns from papers. To demonstrate the approach, we take the paper [113] as example. Leaving out information like related work, literature, examples, results, summaries and other redundancies, the paper includes most of the information needed for the creation of the pattern, e.g. several prerequisites, data requirements, process steps, configuration information and pseudo-code. The second and third paragraph of Section 1 of the paper cover the prerequisites that the problem addressed is procurement fraud and that it is assumed that there is no large training dataset available. Instead, user feedback is used for constituting a relevant similarity of employees and creditors. Section 2.1 of the paper describes requirements on the data, e.g. the attributes of the payroll and the creditor data. In Section 2.2 of the paper, the steps of the process that involve the user feedback are described. Section 3 of the paper addresses a configuration issue on which of several similarity measures to involve in the computation of the similarity function. Section 4 of the paper presents pseudo-code on the algorithm that optimizes the overall distance measure. In addition, instructions on how to transfer a model to another dataset are given. Finally, Section 7 of the paper contains information on how to extend the framework by more similarity measures.

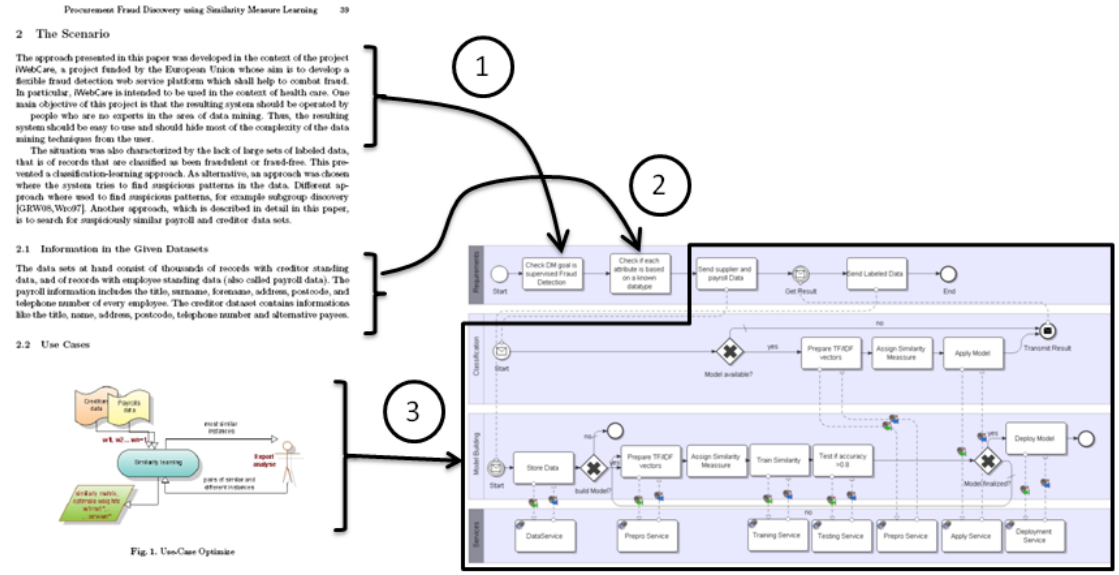

Figure 5.26.: An example on how to map the information of a paper to a process pattern (overview). See Figure 5.27 for details on the process pattern.

All this information can be used for the creation of the data mining process pattern. Figure 5.26 gives an example on how the parts of the paper can be mapped: The first paragraph can be mapped to the prerequisite tasks of the process pattern (1), which are evaluated first when integrating the pattern into a business process. These tasks cannot be specified more detailed as it is done in the paper. Thus, the tasks become tasks at the conceptual level in the process pattern. The second paragraph does not hold any useful information for the pattern. The third paragraph describes a requirement to the data and thus can be mapped to the task of verifying the data quality (2). The figure presents the interaction with the user in terms of manually analysing similarities and thus can be mapped to task of sending labelled data to the model building part of the pattern (3). The paper also contains pseudo-code. As experiments are provided as well, it can be assumed that there is existing executable code available. Thus, in case the code is accessible, this kind of information can be transformed into a tasks at the executable level in the process pattern.

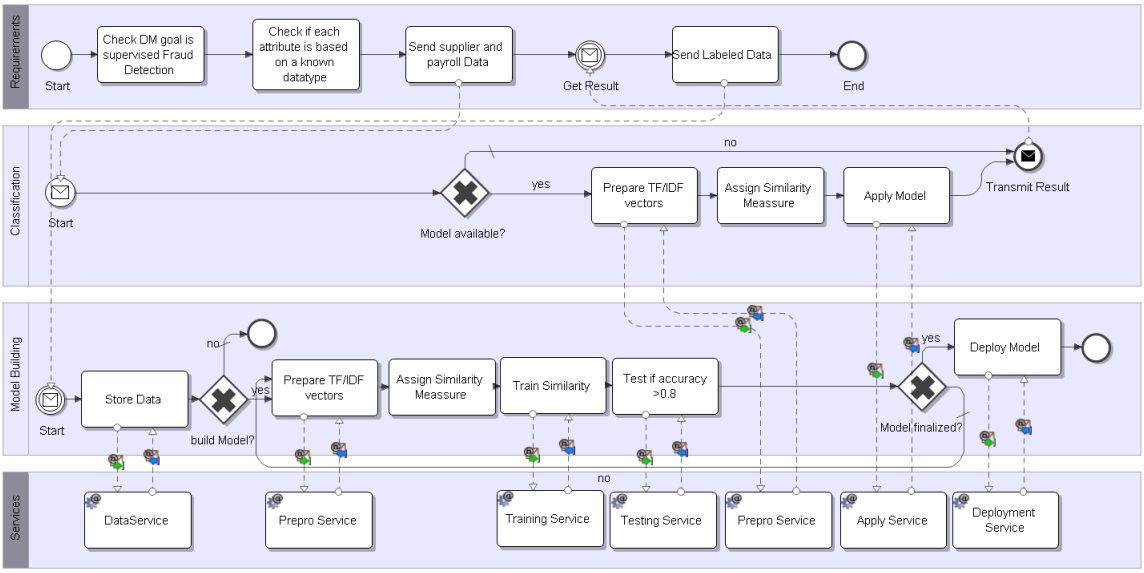

Figure 5.27.: An example of a data mining process pattern for procurement fraud visualized with Intalio BPMS 6.0.1 [76].

Figure 5.27 shows a detailed example of a pattern for the presented approach to procurement fraud detection. In the top pool *Requirements* the prerequisites for applying this process pattern are modelled. This includes checking the data mining goal (procurement fraud detection) and the data format as well as sending data, receiving the result and sending labelled data. The other pools *Classification* and *Model Building* contain the (partially already specified executable) tasks of the data mining process pattern and the respective services. It can be seen that the process is a data mining process pattern according to our definition from Section 5.5.1, which contains tasks of the different levels of the task hierarchy. E.g., the task *Check DM goal is supervised Fraud Detection* is a specialized task of the CRISP task *Check DM goal*. It is a conceptual task which describes in textual form that the goal of the integration of the data mining solution has to be

supervised fraud detection in order to match with the process pattern. When applying the process pattern for the integration, the user manually processes this task. The task *Check if each attribute is based on a known data type* represents the requirement for a specific data format. The data mining solution is based on combining different similarity measures for attributes of different data types. For each of those types, a measure has to exist. The data is sent to the model building part of the process via the task *Send supplier and payroll data* and the labels via *Send labelled Data*. The assignment of the similarity measures is performed at the task *Assign Similarity Measure*. It is a configurable specialization of the task *Train Model* that can be further specialized by the user, e.g. by a configuration file. All executable tasks, e.g. the task *Train Similarity*, are connected to an underlying service.

5.9.3. Integration and Reuse

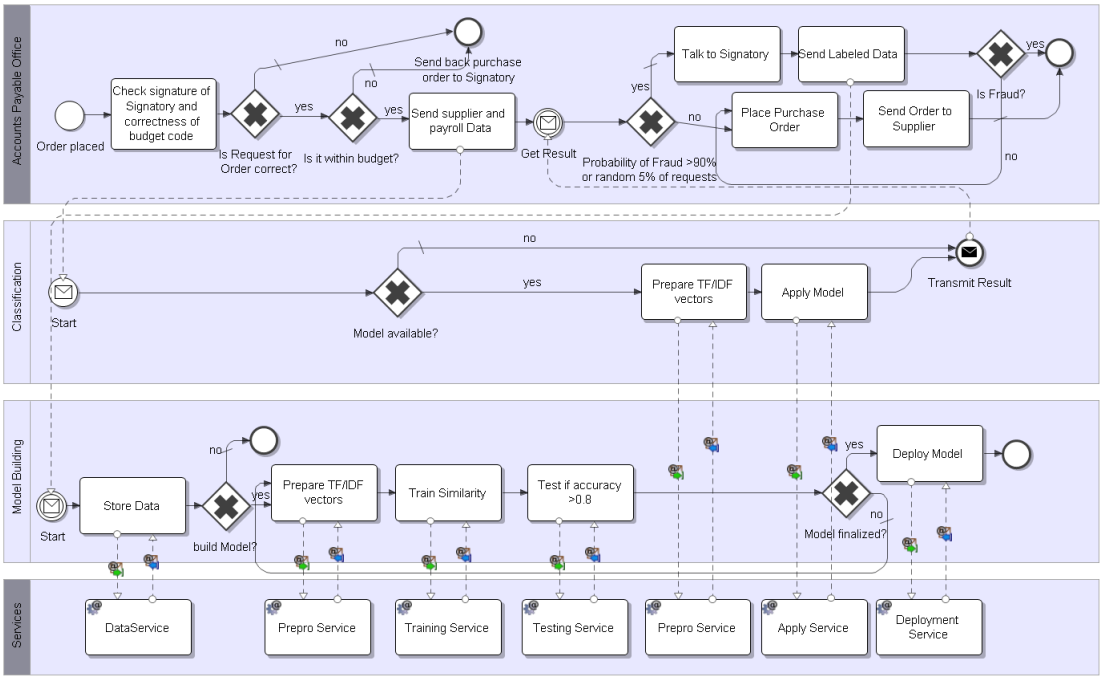

Figure 5.28.: Integration of the procurement fraud process pattern into the business process *place purchase order* visualized with Intalio BPMS 6.0.1 [76].

In [78] (Section 6) a set of business processes from the health care domain is presented which contains candidates for the integration of the fraud detection solution. We will focus on the business process *Purchase Order Inspection* from the RBH scenario, which consists of a random checking of several rules for a request for an order followed by the decision of placing or not placing this order. According to the business objectives of the business process, the data mining process pattern for procurement fraud detection presented in Figure 5.27 is a candidate for the integration. Following the meta-process defined in Section 5.5.2, the tasks of the data mining pattern have to be specified to the executable

level and the process pattern has to be deployed into the business process. Figure 5.28 shows the result of the integration of the presented data mining process pattern into the business process. Compared to the process pattern, the task *Check DM goal* disappeared because the user took the decision that the data mining goal indeed matched and hence removed it. The tasks for sending data and receiving the results are connected with the tasks of the business process. The tasks *Check if each attribute is based on a known data type* and *Assign Similarity Measure* disappeared as well, as the user manually specified the checking and the assignment of the similarities according to his knowledge of the data (e.g. by a parameter file). The information on the assignment is now part of the Train Similarity task. During the integration, the pool *Requirements* and its tasks disappeared at all due to the specification and deployment tasks of the meta-process.

By this somewhat simplified example we have shown how to apply a data mining process pattern to a business process in order to get an integrated process and how manual steps can be specified to executable tasks.

In addition, the re-usability can be demonstrated by our example. In the *Find new supplier* process in [78] (Section 6), in which new suppliers are to be found for certain purchase order requests, a similar problem has to be solved. As the data mining solution is generic, it can be reused and integrated in both process examples easily (see Figure 5.29).

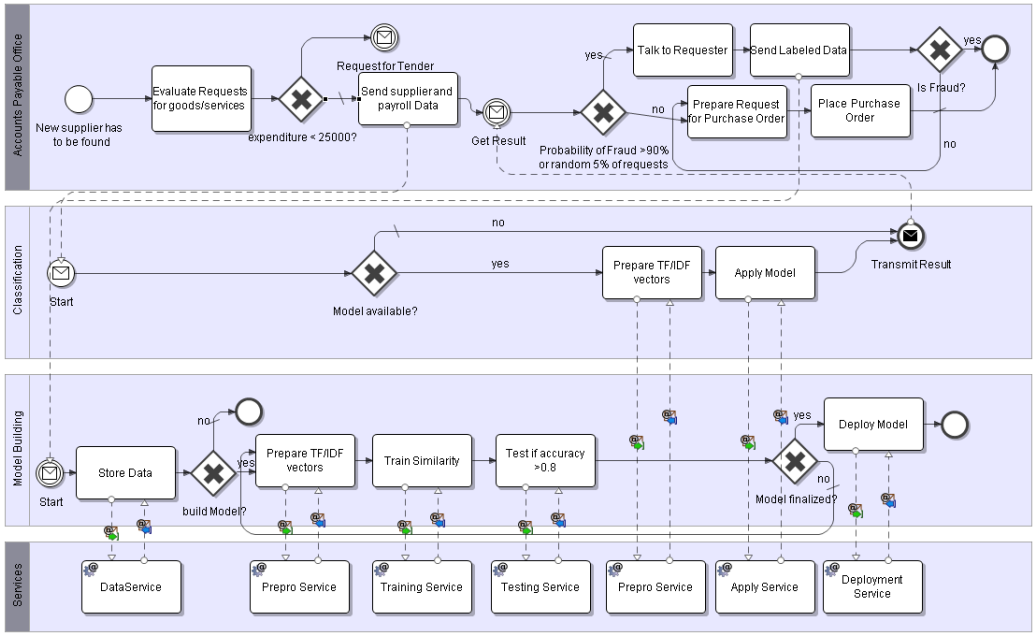

Figure 5.29.: Integration of the procurement fraud process pattern into the business process *find new supplier* visualized and implemented with Intalio BPMS 6.0.1 [76].

5.10. Evaluation & Wrap-up

In this section we will describe how our approach to reusing processes can be evaluated according to best practices in business process redesign. In addition, we will summarize the context of this chapter.

5.10.1. Evaluation in the Context of Business Process Redesign

By our approach for the reuse of processes we want to improve the efficiency and effectiveness in bioinformatics scenarios. Ideally, a redesign or modification of a process decreases the time required to handle incidents, it decreases the required cost of executing the process, it improves the quality of the service that is delivered and it improves the ability of the process to react flexible to variation [17]. However, a property of such an evaluation is that trade-off effects become visible, which means that in general, improving upon one dimension may have a weakening effect on another.

In [104], a set of best practice heuristic rules on process (re)design is evaluated according to the metric cost, time, flexibility and quality. In the following, we give details on rules from [104] which are related to our approach:

- *Order types*: 'determine whether tasks are related to the same type of order and, if necessary, distinguish new business processes' - If parts of business processes are not specific for the business process they are part of, this may result in a less effective management of this sub-process and a lower efficiency. Applying this best practice may yield faster processing times and less cost.

  In our context, the abstraction from and specialization to tasks of the executable level according to the task hierarchy addresses this issue. If executable tasks are not longer matching to the process, they are abstracted to configurable, structural or conceptual tasks and then specialized into new executable tasks addressing the needs of the process.

- *Task elimination*: 'eliminate unnecessary tasks from a business process' - A task is considered as unnecessary when it adds no value from a customer's point of view, e.g., control tasks that are incorporated in the process to fix problems created (or not elevated) in earlier steps. The aim of this best practice is to increase the speed of processing and to reduce the cost, while an important drawback may be that the quality of the service decreases.

  In our context, we intentionally go the other way around - we add tasks for checking requirements, which increases the time and cost in the executable process. However, the check tasks facilitate the reuse and can help to early detect problems, thus reducing the time and cost for the reuse.

- *Triage*: 'consider the division of a general task into two or more alternative tasks' or 'consider the integration of two or more alternative tasks into one general task' and *Task composition*: 'combine small tasks into composite tasks and divide large tasks into workable smaller tasks' - These best practice improve the quality of the business process due to a better utilization of resources with obvious cost and time advantages.

However, too much specialization can make processes become less flexible and less efficient.

According to the task hierarchy these best practices can be addressed by the abstraction to and specialization from tasks at the structural level. However, the concrete specification of tasks is up to the user, which makes it hard to make a general statement in terms of cost, time and flexibility.

- *Knock-out*: 'order knock-outs in a decreasing order of effort and in an increasing order of termination probability' and *Control addition*: 'check the completeness and correctness of incoming materials and check the output before it is send to customers' - Additional checks increase time and costs, but increase the quality delivered. However, checking of conditions that must be satisfied to deliver a positive end result in the correct order reduces the cost and time without loss in quality.

  The meta-process is defined such that only patterns with a matching business objective are considered for an application. By modelling tasks for checking the data mining goal as well as the requirements and prerequisites at the beginning of a process pattern it can be ensured that these checks are performed at an early stage in the process in order to stick to the best practices, thus reducing cost and time.

5.10.2. Summary

In this chapter we have presented an approach for describing data mining based analysis processes in the context of bioinformatics in a way that facilitates reuse. Our approach is based on CRISP and includes the definition of data mining process patterns, a hierarchy of tasks to guide the specialization of abstract process patterns to concrete processes, and a meta-process for applying process patterns to new problems. These data mining process patterns allow for representing the reusable parts of a data mining process at different levels of abstraction from the CRISP model as most abstract representation to executable workflows as most concrete representation, thus providing a simple formal description for the reuse and integration of data mining.

In addition, we have introduced an approach on data semantic aware data mining process patterns. The approach is based on encoding data requirements and pre-conditions for analysis processes that are related to the data as queries to semantically annotated data sources inside a data mining process pattern. Using a medical ontology, semantic information can be integrated into the data mining process patterns for formally checking the data requirements of analysis scenarios. In our approach we combine the concept of semantic mediation of data sources with the concept of data mining process patterns. With our approach we support the reuse of data mining based analysis processes in the area of medical and bioinformatics by a formal representation of data semantics and thus speed up the development of new solutions.

We evaluated our approach in 3 case studies. In the first case study we presented how to create and apply data mining process patterns in the context of a clinical trial scenario. It was shown that it is possible to create a process pattern by abstracting executable tasks of a script and to apply this pattern by specializing it into a workflow including a manual task. The second case study dealt with the transformation of a data mining process

pattern of a multi-center-multi-platform scenario into a process pattern for describing the abstract process of meta analysis in bioinformatics. By this it was shown that it is possible to describe abstract processes consisting of conceptual tasks. In the third case study we described how data mining process patterns can be integrated into business processes in a fraud detection scenario in the health care domain. We demonstrated how a data mining process pattern can be created based on information from a data mining paper and how to integrate this pattern into business processes.

Thus, it was shown that our approach meets the requirements of the users in terms of supporting reuse of data mining based analysis processes and that it can be applied in practice. Furthermore, we presented how our approach can be attributed to best practices for process optimization in the context of business processes.

6. Conclusion

This chapter presents the conclusion of this thesis. First, the main contributions of this work will be summarized in Section 6.1. Second, Section 6.2 presents a discussion on the limitations of the contributions.

6.1. Summary

This thesis provided several contributions for supporting users from the medical and bioinformatics domain in the integration of data mining into scientific data analysis processes. In the following, the main contributions that represent the answers to the research questions formulated in Section 1.2 will be summarized:

Q1: *What is a suitable integration mechanism that allows users with a lot of domain knowledge but without knowledge on grid systems to integrate data mining components that have been developed in single computer environments into distributed grid-based analysis environments used for bioinformatics scenarios?*

In Chapter 3 we presented an approach to support users in integrating already available data mining components into distributed grid environments. The approach is based on describing data mining components that have been developed for single computer environments with the help of a predefined XML schema (the Application Description Schema) in such a way that, in addition to the already available executable file of the component, only the metadata description is needed for the integration into a distributed grid environment. By this, the procedure for the integration is facilitated for the users. The application descriptions are used for interacting with core services of a grid system to register and search for available data mining components on the grid, to match analysis jobs with suitable computational resources, and to dynamically create user interfaces. The presented approach allows for an integration of data mining components by users without deeper knowledge on the underlying grid technology and without intervention on the component side. We have shown that it is possible to cover all information necessary for the execution of data mining components in OGSA-based grid environments with a single XML schema and that it is possible to create a technical system for the execution of data mining components based on data exchange via the XML schema. Our approach allows for a web-site based procedure of grid-enabling data mining components.

We validated our approach in several case studies in the context of the DataMiningGrid project. In the first case study it was shown that it is possible to implement the architecture of our approach in the context of the DataMiningGrid project. The second case study demonstrated that it is possible to create a user interface for grid-enabling data mining components based on the Application Description Schema which is easy to use by users who do not have knowledge on grid technology. In the third case study we have shown

that our approach is applicable for the integration of standard data mining components into grid environments by grid-enabling algorithms of the Weka data mining toolkit. The forth case study demonstrated that our approach also supports the data mining scenarios Data Partitioning, Classifier Comparison and Parameter Variation, which are based on grid-enabled components.

Q2: *Is it possible to create a technical system that allows scientific users to interactively develop data mining scripts consisting of one or more data mining components for bioinformatics scenarios in distributed grid-based analysis environments?*

In addition to reusing existing components for data mining from single computer environments, scenarios in bioinformatics demand for developing new data mining components and developing them further within distributed analysis environments. In Chapter 4, this thesis presented an approach for interactive development of data mining scripts in the context of bioinformatics scenarios in grid environments. The approach is implemented in the GridR toolkit, consisting of the GridR service and the GridR client. The GridR service is a single grid service with complex inputs and outputs that allows for providing scripts as parameter instead of registering each single script as separate component in the grid. By this, it allows for developing and executing data mining scripts based on the scripting language R in grid environments without the need for developing, describing and deploying atomic components, thus making the development process more efficient. In addition, the GridR client allows for interactively developing data mining scripts in grid environments, which allows users to use a well known tool as user interface for the grid environment.

The presented approach was evaluated in two case studies that demonstrated the applicability of the approach in different application scenarios in the context of the ACGT project. In detail, we demonstrated a data analysis scenario from bioinformatics implemented with GridR and a scenario from an industrial application that is parallelized using GridR.

Q3: *Can we define a description for data mining processes that allows for the reuse of existing data mining processes based on data mining components and scripts?*

As today's analysis processes of scenarios in bioinformatics include complex process chains and due to increasing collaboration, the reuse of processes and components becomes more important. In Chapter 5 we presented an approach for supporting the reuse of data mining based data analysis processes based on describing the steps of these processes at different levels of abstraction. The basic idea is to abstract tasks from a concrete executable workflow to create a reusable process pattern. This process pattern can then be reused by specializing it to an executable workflow for another problem. Our approach is based on CRISP and includes the definition of data mining process patterns, a hierarchy of tasks to guide the specialization of abstract process patterns to concrete processes, and a meta-process for applying process patterns to new problems. The data mining process patterns allow for the description of process tasks and requirements based on the task hierarchy. Such process patterns represent a flexible representation for different levels of generality of tasks in the analysis process and allow for describing processes between the CRISP process as most abstract process and executable workflows as most concrete

processes. The meta-process guides the user when applying a process pattern and includes the steps for the specialization of generic tasks to executable tasks.

Our approach was evaluated in 3 case studies in the context of the projects ACGT, p-medicine and iWebCare. In the first case study we presented how to create and apply data mining process patterns in the context of a clinical trial scenario. It was shown that it is possible to create a process pattern by abstracting executable tasks of a script and to apply this pattern by specializing it into a workflow including a manual task. In the second case study we transformed a data mining process pattern of a multi-center-multi-platform scenario into a process pattern for describing the abstract process of meta analysis in bioinformatics. By this it was shown that it is possible to describe abstract processes consisting of conceptual tasks. In the third case study we described how data mining process patterns can be integrated into business processes in a fraud detection scenario from the health care domain. We demonstrated how a data mining process pattern can be created based on information from a data mining paper and how to integrate this pattern into business processes. Furthermore, we showed how our approach can be evaluated according to best practices in business process redesign.

6.2. Discussion

In this thesis we tackled the research question on how to support users in the bioinformatics and medical domain in data mining based data analysis in the context of heterogeneous settings including heterogeneous user groups, heterogeneous data sources and heterogeneous computing environments. Summarized, we developed tools and methods that facilitate the use and reuse of data mining components, scripts and processes in scenarios from this domain to answer the question stated above. By our contributions, the development of new scenarios becomes more efficient, as these can be founded on existing components, scripts and processes more easily.

It has been shown that the presented tools and methods help bioinformaticians in their work. However, although we delivered important building blocks to address the problem of supporting users as described above, there remain open issues for future work.

The challenge of heterogeneous group of users has been addressed by providing a method for reusing data mining components that is flexible enough to support a variety of tools used by the different users. By focussing on OGSA-based grid environments, which allow to build secure distributed systems, the challenges of users in different locations, multi-computer environments and distributed data sources have been addressed. The solutions presented in Chapters 3 and 4 are focussed on OGSA-based grid computing environments. The Application Description Schema including the associated services as well as the architecture of the GridR service are based on the batch job processing functionality that such environments currently provide.

However, the tools used and the distributed environments will be further developed and new architectures and infrastructures will emerge. Thus, if these environments evolve in future, our approaches might not longer directly match. Today, bioinformaticians are working with a huge set of different tools ranging from data mining toolkits such as Weka or RapidMiner and scripting environments such as R up to workflow environments such

as Taverna or Galaxy and their workflow sharing mechanisms, and they probably will work with lots of different analysis tools and process environments in future. Thus, the integration of such tools, including both the reuse of the functionality of the tools and the reuse of their user interfaces, with up-to-date distributed environments and the reuse of the solutions developed with these tools will still remain a problem.

The direct integration of scalable distributed environments into data mining toolkits, as described for R and grid computing environments in the context of the GridR client in Chapter 4, can also be useful for other environments. E.g., in [144] we investigated the integration of Weka and Hadoop, which is a system for cluster and cloud computing environments.

The challenge of complex process chains has been addressed by providing a method for supporting the reuse of processes based on different levels of abstraction. The data mining process pattern approach presented in Chapter 5 provides a step into the direction of describing processes for reuse from the perspective of the bioinformatician and the steps that he needs to perform to reuse a process. However, there is a need for a tool that guides through the process of reuse that can be attached to different process environments. In addition, there is no support for deciding whether a process pattern is a good process pattern for a given data mining problem. The question remains on how the quality of a process pattern can be determined and how to select a process pattern from a process pattern database. Furthermore, it has to be analysed whether the hierarchy of tasks is adequate or if it would make sense to combine the current user focussed view on the tasks that the user needs to perform with additional structures for domain-oriented tasks that could be provided by the use of a task ontology. Furthermore, a detailed user study on our approach for reusing processes would be beneficial to provide more evidence for its usefulness.

In summary, we provided a solution based on existing technology based on methods that allow for the reuse of existing data mining components in grid environments, for the interactive development of data mining scripts consisting of one or more components in a grid environment, and for the reuse of processes including data mining components and scripts.

A. Appendix

A.1. Application Description Schema Source Files

In the following, we present the xsd files representing the schema definition of the DataMiningGrid Application Description Schema as described in Section 3.3.2.

A.1.1. dmg_application_description.xsd

Listing A.1: XML schema dmg_application_description.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.datamininggrid.org/applicationDescription"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:app="http://www.datamininggrid.org/applicationDescription"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
        jaxb:version="1.0">

        <xsd:import namespace="http://www.datamininggrid.org/common"
            schemaLocation="dmg_common.xsd" />
        <xsd:import namespace="http://www.datamininggrid.org/datamining"
            schemaLocation="dmg_data_mining.xsd" />

        <xsd:annotation>
                <xsd:documentation xml:lang="EN">
                        Schema for describing data mining applications for
                            the grid environment developed as part of the
                            project "Data Mining Tools and Services for Grid
                            Computing Environments" (DataMiningGrid), funded
                            by the European Commission (grant no.
                            IST-2004-004475). For more information about
                            this project please visit the homepage at
                            www.datamininggrid.org .
                </xsd:documentation>
        </xsd:annotation>
```

```
<xsd:complexType name="ApplicationDescriptionType">
        <xsd:sequence>
                <xsd:element name="generalInformation"
                    type="com:GeneralInformationType" />
                <xsd:element name="applicationType"
                    type="dm:DataMiningType" />
                <xsd:element name="execution"
                    type="com:ExecutionType" />
                <xsd:element name="applicationInformation">
                        <xsd:complexType>
                                <xsd:sequence>
                                        <xsd:element name="options"
                                            type="com:OptionType"
                                            minOccurs="0"
                                            maxOccurs="unbounded" />
                                        <xsd:element name="dataInputs"
                                            type="com:DataInputType"
                                            minOccurs="0"
                                            maxOccurs="unbounded" />
                                        <xsd:element
                                            name="dataOutputs"
                                            type="com:DataOutputType"
                                            minOccurs="0"
                                            maxOccurs="unbounded" />
                                        <!-- Env variables to set on
                                            execution machine -->
                                        <xsd:element
                                            name="hostEnvironmentVariables"
                                            type="com:EnvironmentType"
                                            minOccurs="0"
                                            maxOccurs="unbounded" />
                                        <!-- Requirements to meet by
                                            excution machine -->
                                        <xsd:element
                                            name="requirements"
                                            type="com:RequirementType"
                                            minOccurs="0"
                                            maxOccurs="1" />
                                        <!-- Parameters defined for
                                            creating multiple jobs
                                            (e.g., for parameter
                                            sweeps) -->
                                        <xsd:element
                                            name="parameterLoops"
                                            type="com:ParameterLoopType"
```

```
                                                    minOccurs="0"
                                                    maxOccurs="unbounded" />
                                                <xsd:element
                                                    name="parameterLists"
                                                    type="com:ParameterListType"
                                                    minOccurs="0"
                                                    maxOccurs="unbounded" />
                                            </xsd:sequence>
                                        </xsd:complexType>
                                    </xsd:element>
                                </xsd:sequence>
                        </xsd:complexType>

                        <xsd:element name="application"
                            type="app:ApplicationDescriptionType"/>

</xsd:schema>
```

A.1.2. dmg_common.xsd

Listing A.2: XML schema dmg_common.xsd

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.datamininggrid.org/common"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        version="1.0">

        <xsd:complexType name="GeneralInformationType">
                <xsd:sequence>
                        <xsd:element name="longDescription"
                            type="xsd:string" /> <!-- Comprehensive
                            discription of the algorithm -->
                        <xsd:element name="comment" type="xsd:string"
                            minOccurs="0" /> <!-- Additional comments. -->
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:string" use="required"
                    />  <!-- Globally unique ID. Will be set
                    automatically-->
                <xsd:attribute name="vendor" type="xsd:string"
                    use="required" />
                <xsd:attribute name="swVersion" type="xsd:string"
                    use="required" /> <!-- Version of the software, e.g.,
                    3.2-->
```

```
            <xsd:attribute name="codeVersion" type="xsd:string"
                use="required" /> <!-- Version of the source code,
                e.g., 2.4.15 -->
            <xsd:attribute name="build" type="xsd:positiveInteger"
                use="required" /><!-- Version of the build, e.g., 5234
                -->
            <xsd:attribute name="shortDescription" type="xsd:string"
                use="required" /><!-- Three sentences at most. -->
            <xsd:attribute name="uploadDate" type="xsd:date"
                use="required" /> <!-- Will be set automatically -->
        </xsd:complexType>

<!-- START: Loops & lists -->
        <!-- Structure for creating multiple values for a single option in
            a loop-like manner.
                The parameter name (e.g. $X) may be referenced by option
                values (e.g. -number $X), input, and output file/dir
                names.
            The loop is evaluated automatically by the Execution
                sub-system and each iteration results in a separate
                execution of the application.
            Applies to numeric options only.
            FILLED BY END-USER OR SYSTEM! -->
        <xsd:complexType name="ParameterLoopType">
            <xsd:attribute name="parameterName" type="xsd:string"
                use="required" />
            <xsd:attribute name="fromValue" type="xsd:string"
                use="required" />
            <xsd:attribute name="toValue" type="xsd:string"
                use="required" />
            <xsd:attribute name="step" type="xsd:string"
                use="required" />
        </xsd:complexType>

        <!-- Structure for defining multiple values for a single option as
            a list.
                The parameter name (e.g. $X) may be referenced by option
                values (e.g. -name $X), input, and output file/dir
                names.
            The list is evaluated automatically by the Execution
                sub-system and each value results in a separate
                execution of the application.
            Applies to any option.
            FILLED BY END-USER OR SYSTEM! -->
        <xsd:complexType name="ParameterListType">
```

```
                <xsd:attribute name="parameterName" type="xsd:string"
                    use="required" />
                <xsd:attribute name="values" type="com:ValueListType"
                    use="required" />
        </xsd:complexType>
<!-- END: Loops & lists -->

<!-- Start: Options -->
        <xsd:complexType name="OptionType">
                <!-- Name of option to be displayed to the user on the
                    client -->
                <xsd:attribute name="label" type="xsd:string"
                    use="required" />
                <!-- Data type of this option. -->
                <xsd:attribute name="dataType" type="com:OptionDataType"
                    use="required" />
                <!-- Used for storing 1 to n default values. -->
                <xsd:attribute name="defaultValues"
                    type="com:ValueListType" use="required" />
                <!-- Used for storing the value specified by the user and
                    for defining parameter sweeps. FILLED BY END-USER OR
                    SYSTEM! -->
                <xsd:attribute name="value" type="xsd:string"
                    use="optional" />
                <!-- All possible values. Applies to data type LIST only
                    -->
                <xsd:attribute name="possibleValues"
                    type="com:ValueListType" use="optional" />
                <!-- Multiple values possible? Applies to data type LIST
                    only -->
                <xsd:attribute name="multipleValues" type="xsd:boolean"
                    use="optional" />
                <!-- Required for delimiting multiple values. Must be
                    printed to cmd line by the system. Applies only if
                    multipleValues=true. -->
                <xsd:attribute name="delimiter" type="xsd:string"
                    use="optional" />
                <!-- Flag to be printed to command line at startup by the
                    system -->
                <xsd:attribute name="flag" type="xsd:string"
                    use="required" />
                <!-- Has this option to be defined? -->
                <xsd:attribute name="optional" type="xsd:boolean"
                    use="required" />
```

```
                <!-- Hide option from user? If set to true, defaultValues
                    MUST be set!-->
                <xsd:attribute name="hidden" type="xsd:boolean"
                    use="required" />
                <!-- Two short sentences at most. -->
                <xsd:attribute name="toolTip" type="xsd:string"
                    use="required" />
        </xsd:complexType>

        <xsd:simpleType name="ValueListType">
                <xsd:list itemType="xsd:string" />
        </xsd:simpleType>

        <xsd:simpleType name="OptionDataType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="int" />
                        <xsd:enumeration value="posint" /> <!-- Positive
                            integer -->
                        <xsd:enumeration value="float" />
                        <xsd:enumeration value="double" />
                        <xsd:enumeration value="boolean" />
                        <xsd:enumeration value="string" />
                        <xsd:enumeration value="list" /> <!-- For a limited
                            number of possible discrete values to select. -->
                </xsd:restriction>
        </xsd:simpleType>
<!-- END: Options -->

<!-- START: Input/output data -->
        <!-- Files containing input data for the application. -->
        <xsd:complexType name="DataInputType">
                <xsd:sequence>
                        <!-- Used for storing value specified by the user.
                            FILLED BY END-USER OR SYSTEM! -->
                        <xsd:element name="input"
                            type="com:RemoteFileDirType" minOccurs="0"
                            maxOccurs="1" />
                </xsd:sequence>
                <xsd:attribute name="ioType" type="com:IOType"
                    use="required" />
                <xsd:attribute name="label" type="xsd:string"
                    use="required" />
                <xsd:attribute name="flag" type="xsd:string"
                    use="required" />
```

```
        <xsd:attribute name="toolTip" type="xsd:string"
            use="required" />
        <xsd:attribute name="optional" type="xsd:boolean"
            use="required" /> <!-- Optional input? -->
        <xsd:attribute name="hidden" type="xsd:boolean"
            use="required" /> <!-- Hide from user? -->
        <!-- Whether to stage in (physically copy) this input to
            the execution machine. -->
        <xsd:attribute name="stageIn" type="xsd:boolean"
            use="optional" default="true"/>
        <!-- Whether to pass the input to the appliation via
            command line or just stage it in. -->
        <xsd:attribute name="appendToCmdLine" type="xsd:boolean"
            use="optional" default="true"/>
</xsd:complexType>

<!-- Files containing any results. -->
<xsd:complexType name="DataOutputType">
        <xsd:attribute name="ioType" type="com:IOType"
            use="required" />
        <xsd:attribute name="label" type="xsd:string"
            use="required" />
        <xsd:attribute name="flag" type="xsd:string"
            use="required" />
        <xsd:attribute name="toolTip" type="xsd:string"
            use="required" />
        <xsd:attribute name="optional" type="xsd:boolean"
            use="required" /> <!-- Optional output? -->
        <xsd:attribute name="hidden" type="xsd:boolean"
            use="required" /> <!-- Hide from user? -->
        <!-- Whether to stage out this output to a save storage
            server in the grid. -->
        <xsd:attribute name="stageOut" type="xsd:boolean"
            use="optional" default="true"/>
        <!-- Used for storing the value specified by the user. May
            contain references to parameters. FILLED BY END-USER OR
            SYSTEM! -->
        <xsd:attribute name="fileDirParameterName"
            type="xsd:string" use="optional" />
        <!-- Whether to pass the output to the appliation via
            command line or just stage it out. -->
        <xsd:attribute name="appendToCmdLine" type="xsd:boolean"
            use="optional" default="true"/>
</xsd:complexType>
```

```
        <xsd:complexType name="RemoteFileDirType">
                <xsd:attribute name="protocol"
                    type="com:RemoteProtocolType" use="required" />
                <xsd:attribute name="port" type="xsd:int" use="required" />
                <xsd:attribute name="host" type="xsd:string"
                    use="required" />
                <xsd:attribute name="localPath" type="xsd:string"
                    use="required" />
                <!-- Name of dir/file required for stage-in on
                    server-side. May contain references to parameters. -->
                <xsd:attribute name="fileDirParameterName"
                    type="xsd:string" use="required" />
                <xsd:attribute name="description" use="optional" />
        </xsd:complexType>

        <!-- Protocols for file transfers allowed in the DMG system.
                Maybe enhanced with protocols ftp, http, https in later
                    stages of the project. -->
        <xsd:simpleType name="RemoteProtocolType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="gsiftp" />
                        <xsd:enumeration value="http" />
                </xsd:restriction>
        </xsd:simpleType>

        <xsd:simpleType name="IOType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Datafile" />
                        <xsd:enumeration value="Directory" />
                        <xsd:enumeration value="Parameterfile" />
                </xsd:restriction>
        </xsd:simpleType>
<!-- END: Input/output data -->
<!-- START: Environment -->
        <!-- The environment variables to be set at the execution machine
            before execution. -->
        <xsd:complexType name="EnvironmentType">
                <xsd:attribute name="variable" type="xsd:string"
                    use="required" />
                <xsd:attribute name="value" type="xsd:string"
                    use="required" />
                <xsd:attribute name="toolTip" type="xsd:string"
                    use="required" />
        </xsd:complexType>
<!-- END: Environment -->
```

```
<!-- START: Execution -->
        <xsd:complexType name="ExecutionType">
                <xsd:sequence>
                        <xsd:choice>
                                <xsd:element name="javaExecution"
                                    type="com:JavaLanguageType" /> <!-- Java
                                    -->
                                <xsd:element name="cExecution"
                                    type="com:CLanguageType" />  <!-- C/C++
                                    -->
                                <xsd:element name="unixExecution"
                                    type="com:BashShellLanguageType" /> <!--
                                    Bash-Shell -->
                                <xsd:element name="pythonExecution"
                                    type="com:PythonLanguageType" /> <!--
                                    Python -->
                        </xsd:choice>
                        <xsd:element name="requiredLibrary"
                            type="com:RemoteFileDirType" minOccurs="0"
                            maxOccurs="unbounded" />
                </xsd:sequence>
                <xsd:attribute name="stageIn" type="xsd:boolean"
                    use="optional" default="true"/>
                <xsd:attribute name="remoteSubDirectory" type="xsd:string"
                    use="optional"/>
        </xsd:complexType>

        <xsd:complexType name="SandboxLanguageType" abstract="true">
                <xsd:sequence>
                        <xsd:element name="interpreterArguments"
                            type="xsd:string" minOccurs="0"
                            maxOccurs="unbounded" />
                        <xsd:element name="applicationRunFile"
                            type="com:RemoteFileDirType" minOccurs="1"
                            maxOccurs="1" />
                </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="JavaLanguageType">
                <xsd:complexContent>
                        <xsd:extension base="com:SandboxLanguageType">
                                <xsd:attribute name="interpreterCommand"
                                    type="xsd:string" use="required"
                                    fixed="java" />
```

```
                        <xsd:attribute name="mainClass"
                            type="xsd:string" use="required" />
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <xsd:complexType name="BashShellLanguageType">
                <xsd:complexContent>
                    <xsd:extension base="com:SandboxLanguageType">
                        <xsd:attribute name="interpreterCommand"
                            type="xsd:string" use="required"
                            fixed="/bin/sh" />
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <xsd:complexType name="PythonLanguageType">
                <xsd:complexContent>
                    <xsd:extension base="com:SandboxLanguageType">
                        <xsd:attribute name="interpreterCommand"
                            type="xsd:string" use="required"
                            fixed="python" />
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <xsd:complexType name="DirectlyExecutableLanguageType">
                <xsd:sequence>
                    <xsd:element name="executable"
                        type="com:RemoteFileDirType" minOccurs="1"
                        maxOccurs="1" />
                </xsd:sequence>
            </xsd:complexType>

            <xsd:complexType name="CLanguageType">
                <xsd:complexContent>
                    <xsd:extension
                        base="com:DirectlyExecutableLanguageType" />
                </xsd:complexContent>
            </xsd:complexType>
    <!-- END: Execution -->

    <!-- START: Requirements for the resource broker -->
            <xsd:complexType name="RequirementType">
                <xsd:sequence>
```

```
                <xsd:element name="minMemory"
                    type="com:MinMemoryRequirementType"
                    minOccurs="0" maxOccurs="1" />
                <xsd:element name="minDiskSpace"
                    type="com:MinDiskSpaceRequirementType"
                    minOccurs="0" maxOccurs="1" />
                <xsd:element name="gramIPs"
                    type="com:GramIPRequirementType" minOccurs="0"
                    maxOccurs="1" />
                <!-- No entry means that the application may runs
                    as "fork" and as "condor" job. -->
                <xsd:element name="gramJobManager"
                    type="com:GramJobManagerRequirementType"
                    minOccurs="0" maxOccurs="1" />
                <xsd:element name="operatingSystem"
                    type="com:OperatingSystemRequirementType"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="architecture"
                    type="com:ArchitectureRequirementType"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="maxNumberOfMachines"
                    type="xsd:positiveInteger" minOccurs="0"
                    maxOccurs="1" />
        </xsd:sequence>
</xsd:complexType>

<!-- Minimum RAM -->
<xsd:complexType name="MinMemoryRequirementType">
        <xsd:attribute name="label" fixed="Min memory"
            type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string"
            use="required" />
        <xsd:attribute name="unit" type="com:UnitType"
            use="required" />
        <xsd:attribute name="toolTip" type="xsd:string"
            use="optional" />
</xsd:complexType>

<!-- Minimum free disk space -->
<xsd:complexType name="MinDiskSpaceRequirementType">
        <xsd:attribute name="label" fixed="Min free disk space"
            type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string"
            use="required" />
```

```
        <xsd:attribute name="unit" type="com:UnitType"
            use="required" />
        <xsd:attribute name="toolTip" type="xsd:string"
            use="optional" />
</xsd:complexType>

<!-- List of IPs of machines with GT4 installations the
    application is allowed to run on. -->
<xsd:complexType name="GramIPRequirementType">
        <xsd:attribute name="label" fixed="Execution machines"
            type="xsd:string" use="required" />
        <xsd:attribute name="value" type="com:ValueListType"
            use="required" />
        <xsd:attribute name="toolTip" type="xsd:string"
            use="optional" />
</xsd:complexType>

<!-- Determines how job are executed by GT4 -->
<xsd:complexType name="GramJobManagerRequirementType">
        <xsd:attribute name="label" fixed="Job manager"
            type="xsd:string" use="required" />
        <xsd:attribute name="value" type="com:JobManagerListType"
            use="required" />
        <xsd:attribute name="toolTip" type="xsd:string"
            use="optional" />
</xsd:complexType>

<xsd:simpleType name="UnitType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="MB" />
                <xsd:enumeration value="GB" />
                <xsd:enumeration value="TB" />
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="JobManagerListType">
        <xsd:list itemType="com:JobManagerType" />
</xsd:simpleType>

<xsd:simpleType name="JobManagerType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Fork" />  <!-- Run on Gram
                    machine -->
                <xsd:enumeration value="Condor" /> <!-- Submit jobs
                    further to Condor pool -->
```

```
            </xsd:restriction>
        </xsd:simpleType>

        <xsd:complexType name="OperatingSystemRequirementType">
            <xsd:attribute name="label" fixed="Operating system"
                type="xsd:string" use="required" />
            <xsd:attribute name="value" type="com:OSType"
                use="required" />
            <xsd:attribute name="toolTip" type="xsd:string"
                use="optional" />
        </xsd:complexType>

        <xsd:simpleType name="OSType">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="ALL" />
                <xsd:enumeration value="WINNT51" />
                <xsd:enumeration value="LINUX" />
            </xsd:restriction>
        </xsd:simpleType>

        <xsd:complexType name="ArchitectureRequirementType">
            <xsd:sequence>
                <xsd:element name="value"
                    type="com:ArchitectureType" minOccurs="1"
                    maxOccurs="2" />
            </xsd:sequence>
            <xsd:attribute name="label" fixed="Architecture"
                type="xsd:string" use="required" />
            <xsd:attribute name="toolTip" type="xsd:string"
                use="optional" />
        </xsd:complexType>

        <xsd:simpleType name="ArchitectureType">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="ALL" />
                <xsd:enumeration value="INTEL" />
                <xsd:enumeration value="PPC" />
                <xsd:enumeration value="ITANIUM" />
            </xsd:restriction>
        </xsd:simpleType>
<!-- END: Requirements for the resource broker -->
<!-- Provenance information -->
        <xsd:complexType name="SingleJobProvenanceType">
            <xsd:sequence>
```

```
                    <xsd:element name="variables"
                        type="com:EnvironmentType" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="jobId" type="xsd:string"
                    use="required" />
                <xsd:attribute name="status" type="xsd:string"
                    use="required" />
                <xsd:attribute name="gram" type="xsd:string"
                    use="optional" />
                <xsd:attribute name="submissionToGRAMTime"
                    type="xsd:dateTime" use="required" />
                <xsd:attribute name="completionTime" type="xsd:dateTime"
                    use="required" />
                <xsd:attribute name="failureDescription" type="xsd:string"
                    use="required" />
        </xsd:complexType>
        <xsd:complexType name="SingleGramProvenanceType">
                <xsd:attribute name="gram" type="xsd:string"
                    use="optional" />
                <xsd:attribute name="doneJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="failedJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="activeJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="pendingJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="submittedJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="localQueueJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="preStageJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="unknownJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="waitingJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="totalJobs" type="xsd:long"
                    use="required" />
                <xsd:attribute name="totalCPUs" type="xsd:long"
                    use="required" />
        </xsd:complexType>
<!-- END: Provenance information -->
</xsd:schema>
```

A.1.3. dmg_data_mining

Listing A.3: XML schema dmg_data_mining.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.datamininggrid.org/datamining"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        version="1.0">

        <xsd:complexType name="DataMiningType">
                <!-- Application domain must be fixed to data mining. -->
                <xsd:attribute name="applicationDomain" type="xsd:string"
                    use="required" fixed="Data Mining"/>
                <!-- Name of the (large-scale) application this algorithm
                    belongs to (e.g. Weka) -->
                <xsd:attribute name="applicationGroup" type="xsd:string"
                    use="required" />
                <!-- Name of the atomic application/algorithm. Should be
                    unique within an application group. -->
                <xsd:attribute name="applicationName" type="xsd:string"
                    use="required" />
                <!-- The type of problem the algorithm solves
                    (e.g.Regression) -->
                <xsd:attribute name="functionalArea"
                    type="dm:FunctionalAreaType" use="required" />
                <!-- The technique used by the actual algorithm (e.g.
                    Regression Tree) -->
                <xsd:attribute name="technique" type="xsd:string"
                    use="required" />
                <!-- See below -->
                <xsd:attribute name="crispDMPhase"
                    type="dm:CrispDMPhaseType" use="required" />
        </xsd:complexType>
        <xsd:simpleType name="FunctionalAreaType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Classification" />
                        <xsd:enumeration value="Regression" />
                        <xsd:enumeration value="Clustering" />
                        <xsd:enumeration value="Attribute Importance" />
                        <xsd:enumeration value="Association" />
                        <xsd:enumeration value="Other" /><!-- Everything
                            that does not fit into above categories -->
                </xsd:restriction>
        </xsd:simpleType>
```

```
        <xsd:simpleType name="CrispDMPhaseType">
                <xsd:restriction base="xsd:string">
                        <!-- Expected to appear rather sporadically -->
                        <xsd:enumeration value="Business Understanding" />
                        <!-- E.g. data exploration techniques (e.g.
                            univariate data plots), data quality issues,
                            attribute interactions, etc. -->
                        <xsd:enumeration value="Data Understanding" />
                        <!-- All preprocessing algorithms, e.g. data
                            integration, selection, sampling,
                            transformation, cleaning, etc. -->
                        <xsd:enumeration value="Data Preparation" />
                        <!-- All DM algorithms, e.g. classifiers,
                            differential equation solvers, etc. -->
                        <xsd:enumeration value="Modelling" />
                        <!-- All validating techniques -->
                        <xsd:enumeration value="Evaluation" />
                        <!-- E.g. monitoring and evaluation effectiveness
                            after deployment-->
                        <xsd:enumeration value="Deployment" />
                </xsd:restriction>
        </xsd:simpleType>

</xsd:schema>
```

A.1.4. dmg_provenance

Listing A.4: XML schema dmg_provenance.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.datamininggrid.org/provenance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:app="http://www.datamininggrid.org/applicationDescription"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        xmlns:prov="http://www.datamininggrid.org/provenance"
        version="1.0">

        <xsd:import
            namespace="http://www.datamininggrid.org/applicationDescription"
            schemaLocation="dmg_application_description.xsd" />
        <xsd:import namespace="http://www.datamininggrid.org/common"
            schemaLocation="dmg_common.xsd" />
```

```
        <xsd:annotation>
                <xsd:documentation xml:lang="EN">
                        Schema for describing data mining applications for
                            the grid environment developed as part of the
                            project
                        "Data Mining Tools and Services for Grid Computing
                            Environments" (DataMiningGrid), funded by the
                            European
                        Commission (grant no. IST-2004-004475). For more
                            information about this project please visit the
                            homepage at
                        www.datamininggrid.org .
                </xsd:documentation>
        </xsd:annotation>

        <xsd:element name="provenance" type="prov:ProvenanceType" />

        <xsd:complexType name="ProvenanceType">
                <xsd:sequence>
                        <xsd:element name="applicationDescription"
                            type="app:ApplicationDescriptionType"
                            minOccurs="1" maxOccurs="1"/>
                        <xsd:element name="submissionTime"
                            type="xsd:dateTime" minOccurs="1" maxOccurs="1"
                            />
                        <xsd:element name="completionTime"
                            type="xsd:dateTime" minOccurs="1" maxOccurs="1"
                            />
                        <xsd:element name="schedulerStatus"
                            type="xsd:string" minOccurs="1" maxOccurs="1" />
                        <xsd:element name="resultsLocation"
                            type="xsd:string" minOccurs="1" maxOccurs="1" />
                        <xsd:element name="jobsStatus"
                            type="com:SingleJobProvenanceType" minOccurs="0"
                            maxOccurs="unbounded"/>
                        <xsd:element name="gramsStatus"
                            type="com:SingleGramProvenanceType"
                            minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

A.2. Application Description Schema Instances

In the following, we present examples of instances of the Application Description Schema for Weka components that have been grid-enabled in the context of the DataMiningGrid project as described in Section 3.4.3.

A.2.1. Weka IBK

Listing A.5: Description of the IBK component of Weka

```
<app:application
    xmlns:app="http://www.datamininggrid.org/applicationDescription"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <generalInformation build="355" codeVersion="3.5.5"
                     id="weka_ibk"
                     shortDescription="Class for ..."
                     swVersion="3.5.5"
                     uploadDate="2007-05-09"
                     vendor="University of Waikato New Zealand">
    <longDescription>Attribute selection: Set the method used to select
        attributes for use in the linear regression. Available methods
        are: no attribute selection, attribute selection using M5's method
        (step through the attributes removing the one with the smallest
        standardised coefficient until no improvement is observed in the
        estimate of the error given by the Akaike information criterion),
        and a greedy selection using the Akaike information
        metric.</longDescription>
    <comment></comment>
  </generalInformation>
  <applicationType applicationName="IBK"
                   applicationGroup="Weka"
                   applicationDomain="Data Mining"
                   crispDMPhase="Modelling"
                   functionalArea="Classification"
                   technique="KNN"/>
  <execution>
    <javaExecution interpreterCommand="java"
        mainClass="weka.classifiers.lazy.IBk">
      <interpreterArguments>-Xmx1000m</interpreterArguments>
      <applicationRunFile description="The main jar containing the main
          class." host="grid2.kd-grid.ais.fraunhofer.de"
          localPath="/gridapps/weka/" fileDirParameterName="weka3-5-5.jar"
```

```
        protocol="gsiftp" port="2811" />
  </javaExecution>
</execution>
<applicationInformation>
  <options dataType="posint"
         defaultValues="1"
         flag="-kNN"
         hidden="false"
         label="KNN"
         optional="false"
         toolTip="The number of neighbours to use."/>
  <options dataType="boolean"
         defaultValues="false"
         flag="-crossValidate"
         hidden="false"
         label="CrossValidate"
         optional="false"
         toolTip="Whether hold-one-out cross-validation will be used to
             select the best k value."/>
       <options dataType="boolean"
         defaultValues="false"
         flag="-debug"
         hidden="false"
         label="Debug"
         optional="false"
         toolTip="If set to true, classifier may output additional info
             to the console."/>
   <options dataType="boolean"
         defaultValues="false"
         flag="-meanSquared"
         hidden="false"
         label="MeanSquared"
         optional="false"
         toolTip="Whether the mean squared error is used rather than
             mean absolute error when doing cross-validation for
             regression problems."/>
  <options dataType="posint"
         defaultValues="0"
         flag="-windowSize"
         hidden="false"
         label="WindowSize"
         optional="false"
         toolTip="Gets the maximum number of instances allowed in the
             training pool. The addition of new instances above this
             value will result in old instances being removed. A value
```

```
                of 0 signifies no limit to the number of training
                instances."/>
     <options dataType="string"
            defaultValues="price"
            flag="-classAttribute"
            hidden="false"
            label="Class Attribute"
            optional="false"
            toolTip="The attribute to predict."/>
        <options dataType="posint"
            defaultValues="10"
            flag="-cv"
            hidden="false"
            label="CV folds"
            optional="false"
            toolTip="Number of cv folds ( has to be > 1, 0 and 1 means no
                cv )."/>
<!-- Hidden -->
    <options dataType="string"
            defaultValues="LinearNN -A weka.core.EuclideanDistance"
            value="LinearNN -A weka.core.EuclideanDistance"
            flag="-nearestNeighbourSearchAlgorithm"
            hidden="true"
            label="NearestNeighbourSearchAlgorithm"
            optional="false"
            toolTip="The nearest neighbour search algorithm to use
                (Default: LinearNN)."/>


<!-- All possible data inputs. All printings to SDTOUT & STDERR by the
    application are covered by the system automatically and thus have not
    to be specified here. -->
   <dataInputs flag="-train"
             ioType="Datafile"
             label="Training Data"
             toolTip="Input files containing the training data in Weka's
                 ARFF format."
                     optional="false"
                     hidden="false"
                     stageIn="true"
                     appendToCmdLine="true"/>
<!-- All possible data outputs -->
   <dataOutputs flag="-result"
              ioType="Datafile"
              label="Result File"
```

```
            toolTip="File containing the results."
                      optional="false"
                      hidden="false"
                      stageOut="true"
                      appendToCmdLine="true"/>

<!-- Requirements for execution machines -->
       <requirements>
              <minMemory label="Min memory" value="512" unit="MB"
                 toolTip="Minimum memory of execution machine must be
                 larger than minimum memory required for JVM." />
              <operatingSystem label="Operating system" value="ALL"/>
              <architecture label="Architecture">
                     <value>ALL</value>
              </architecture>
       </requirements>
  </applicationInformation>
</app:application>
```

A.2.2. Weka LWL

Listing A.6: Description of the LWL component of Weka

```
<app:application
    xmlns:app="http://www.datamininggrid.org/applicationDescription"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <generalInformation build="355" codeVersion="3.5.5"
                     id="weka_lwl"
                     shortDescription="Class for ..."
                     swVersion="3.5.5"
                     uploadDate="2007-05-09"
                     vendor="University of Waikato New Zealand">
    <longDescription>.</longDescription>
    <comment></comment>
  </generalInformation>

  <applicationType applicationName="LWL"
                   applicationGroup="Weka"
                   applicationDomain="Data Mining"
                   crispDMPhase="Modelling"
                   functionalArea="Classification"
                   technique="Locally weighted learning"/>
```

```
<execution>
  <javaExecution interpreterCommand="java"
     mainClass="weka.classifiers.lazy.LWL">
    <interpreterArguments>-Xmx1000m</interpreterArguments>
    <applicationRunFile description="The main jar containing the main
       class." host="grid2.kd-grid.ais.fraunhofer.de"
       localPath="/gridapps/weka/" fileDirParameterName="weka3-5-5.jar"
       protocol="gsiftp" port="2811" />
  </javaExecution>
</execution>
<applicationInformation>
  <options dataType="posint"
         defaultValues="1"
         flag="-kNN"
         hidden="false"
         label="KNN"
         optional="false"
         toolTip="How many neighbours are used to determine the width
            of the weighting function ( 0 or less means all
            neighbours)."/>
  <options dataType="boolean"
         defaultValues="false"
         flag="-debug"
         hidden="false"
         label="Debug"
         optional="false"
         toolTip="If set to true, classifier may output additional info
            to the console."/>
   <options dataType="posint"
         defaultValues="0"
         flag="-weightingKernel"
         hidden="false"
         label="weightingKernel"
         optional="false"
         toolTip="Determines weighting function. [0 = Linear, 1 =
            Epnechnikov,2 = Tricube, 3 = Inverse, 4 = Gaussian and 5 =
            Constant. (default 0 = Linear)]"/>
  <options dataType="string"
         defaultValues="price"
         flag="-classAttribute"
         hidden="false"
         label="Class Attribute"
         optional="false"
         toolTip="The attribute to predict."/>
```

```
    <options dataType="posint"
            defaultValues="10"
            flag="-cv"
            hidden="false"
            label="CV folds"
            optional="false"
            toolTip="Number of cv folds ( has to be > 1, 0 and 1 means no
                cv )."/>

<!-- All possible data inputs. All printings to SDTOUT & STDERR by the
    application are covered by the system automatically and thus have not
    to be specified here. -->
   <dataInputs flag="-train"
              ioType="Datafile"
              label="Training Data"
              toolTip="Input files containing the training data in Weka's
                  ARFF format."
                      optional="false"
                      hidden="false"
                      stageIn="true"
                      appendToCmdLine="true"/>
<!-- All possible data outputs -->
   <dataOutputs flag="-result"
               ioType="Datafile"
               label="Result File"
               toolTip="File containing the results."
                        optional="false"
                        hidden="false"
                        stageOut="true"
                        appendToCmdLine="true"/>

<!-- Requirements for execution machines -->
       <requirements>
              <minMemory label="Min memory" value="512" unit="MB"
                  toolTip="Minimum memory of execution machine must be
                  larger than minimum memory required for JVM." />
              <operatingSystem label="Operating system" value="ALL"/>
              <architecture label="Architecture">
                     <value>ALL</value>
              </architecture>
       </requirements>
  </applicationInformation>
</app:application>
```

A.2.3. Weka M5P

Listing A.7: Description of the M5P component of Weka

```
<app:application
    xmlns:app="http://www.datamininggrid.org/applicationDescription"
        xmlns:dm="http://www.datamininggrid.org/datamining"
        xmlns:com="http://www.datamininggrid.org/common"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <generalInformation build="355" codeVersion="3.5.5"
                      id="weka_m5p"
                      shortDescription="Class for ..."
                      swVersion="3.5.5"
                      uploadDate="2007-05-09"
                      vendor="University of Waikato New Zealand">
    <longDescription>Attribute selection: Set the method used to select
       attributes for use in the linear regression. Available methods
       are: no attribute selection, attribute selection using M5's method
       (step through the attributes removing the one with the smallest
       standardised coefficient until no improvement is observed in the
       estimate of the error given by the Akaike information criterion),
       and a greedy selection using the Akaike information
       metric.</longDescription>
    <comment></comment>
  </generalInformation>
  <applicationType applicationName="M5P"
                   applicationGroup="Weka"
                   applicationDomain="Data Mining"
                   crispDMPhase="Modelling"
                   functionalArea="Classification"
                   technique="M5Base Tree"/>
  <execution>
    <javaExecution interpreterCommand="java"
       mainClass="weka.classifiers.trees.M5P">
      <interpreterArguments>-Xmx1000m</interpreterArguments>
      <applicationRunFile description="The main jar containing the main
         class." host="grid2.kd-grid.ais.fraunhofer.de"
         localPath="/gridapps/weka/" fileDirParameterName="weka3-5-5.jar"
         protocol="gsiftp" port="2811" />
    </javaExecution>
  </execution>
  <applicationInformation>
    <options dataType="boolean"
           defaultValues="false"
           flag="-buildRegressionTree"
```

```
        hidden="false"
        label="BuildRegressionTree"
        optional="false"
        toolTip="Whether to generate a regression tree/rule instead of
            a model tree/rule."/>
     <options dataType="boolean"
        defaultValues="false"
        flag="-debug"
        hidden="false"
        label="Debug"
        optional="false"
        toolTip="If set to true, classifier may output additional info
            to the console."/>
  <options dataType="double"
        defaultValues="4.0"
        flag="-minNumInstances"
        hidden="false"
        label="MinNumInstances"
        optional="false"
        toolTip="The minimum number of instances to allow at a leaf
            node."/>
  <options dataType="boolean"
        defaultValues="false"
        flag="-saveInstances"
        hidden="false"
        label="SaveInstances"
        optional="false"
        toolTip="Whether to save instance data at each node in the
            tree for visualization purposes."/>
  <options dataType="boolean"
        defaultValues="false"
        flag="-unpruned"
        hidden="false"
        label="unpruned"
        optional="false"
        toolTip="Whether unpruned tree/rules are to be generated."/>
  <options dataType="boolean"
        defaultValues="false"
        flag="-useUnsmoothed"
        hidden="false"
        label="UseUnsmoothed"
        optional="false"
        toolTip="Whether to use unsmoothed predictions."/>
  <options dataType="string"
        defaultValues="price"
```

```
            flag="-classAttribute"
            hidden="false"
            label="Class Attribute"
            optional="false"
            toolTip="The attribute to predict."/>
        <options dataType="posint"
            defaultValues="10"
            flag="-cv"
            hidden="false"
            label="CV folds"
            optional="false"
            toolTip="Number of cv folds."/>
<!-- All possible data inputs. All printings to SDTOUT & STDERR by the
    application are covered by the system automatically and thus have not
    to be specified here. -->
   <dataInputs flag="-train"
             ioType="Datafile"
             label="Training Data"
             toolTip="Input files containing the training data in Weka's
                 ARFF format."
                    optional="false"
                    hidden="false"
                    stageIn="true"
                    appendToCmdLine="true"/>
<!-- All possible data outputs -->
   <dataOutputs flag="-result"
              ioType="Datafile"
              label="Result File"
              toolTip="File containing the results."
                       optional="false"
                       hidden="false"
                       stageOut="true"
                       appendToCmdLine="true"/>
<!-- Requirements for execution machines -->
        <requirements>
               <minMemory label="Min memory" value="512" unit="MB"
                  toolTip="Minimum memory of execution machine must be
                  larger than minimum memory required for JVM." />
               <operatingSystem label="Operating system" value="ALL"/>
               <architecture label="Architecture">
                      <value>ALL</value>
               </architecture>
        </requirements>
  </applicationInformation>
</app:application>
```

A.3. R-based Bioinformatics Scenario

In this section, we present the R script code of the scenario described in Section 2.3.5, which is part of the clinical trial case study presented in Section 5.7.

A.3.1. R Code

The code of the original R script of the clinical trial scenario is presented in the following code listing:

Listing A.8: R code of the clinical trial scenario

```
setwd("/p-medicine/pmed_WP2_Data_Example/CEL")

####### LIBRARIES ############

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)

####### READING DATA ############

#Reading AffyBatch from CEL files
mysamples <- ReadAffy()

# Adjusting sampleNames
smpls <- sub(".CEL","", sub(paste(getwd(),"/",
    sep=""),"",sampleNames(mysamples)))
sampleNames(mysamples) <- smpls
sampleNames(mysamples)
print(smpls)

setwd("/Simona/p-medicine/pmed_WP2_Data_Example")


####### NORMALIZATION ############
esetmysamples <- rma(mysamples)

# Saving QC images

degmysamples <- AffyRNAdeg(mysamples)

anname<-"GSE22138"
```

```
png(file=paste("Intensities", anname, ".png", sep=""))
hist(mysamples,col="blue")
dev.off()

png(file=paste("Degradation", anname, ".png", sep=""))
plotAffyRNAdeg(degmysamples)
dev.off()


png(file=paste("BoxplotUnNormal", anname, ".png", sep=""), width=700,
    height=700)
par(mar=c(8,4,4,2))
boxplot(mysamples, names=sampleNames(esetmysamples),las=3,
    cex.axis=0.7,outline=F)

dev.off()
png(file=paste("BoxplotNormal", anname, ".png", sep=""), width=700,
    height=700)
par(mar=c(8,4,4,2))
boxplot(exprs(esetmysamples),
    names=sampleNames(esetmysamples),las=3,cex.axis=0.7,outline=F)

dev.off()

######## ENVIRONMENT STRUCTURE CREATION ############

GSE22138<-new.env()
esetm<-as.matrix(esetmysamples)

GSE22138$gex<-t(esetm)

GSE22138$clin<-read.table(file="GSE22138_clin.csv",header=T,row.names=1)



# differentially expressed genes
design <- model.matrix(~0+factor(d.mfs), data=GSE22138$clin)
colnames(design) <- c("NoMet","Met")
x<-t(GSE22138$gex[rownames(design),])
n<-ncol(x)
fit <- lmFit(x, design)

 contrast.matrix <- makeContrasts(Met-NoMet, levels=design)
 fit2 <- contrasts.fit(fit, contrast.matrix)
```

```
 fit2 <- eBayes(fit2)
  # A list of top genes differential expressed in group2 versus group1
      can be obtained from
topTable(fit2, coef=1, adjust="BH")

  results <- decideTests(fit2,p.value=0.05,lfc=1)
#A Venn diagram showing numbers of genes significant in each comparison
    can be obtained from
png(file="VennDiagram.png")
vennDiagram(results, main="Venn Diagram - p.value < 0.05, FC > 2")
dev.off()

pdf(file="prova_volcano.pdf")
volcanoplot(fit2, coef=1,highlight=6,main="Detach-NoDetach")
dev.off()

x<-results@.Data[abs(results@.Data[,1])==1,]


names(x)
GSE22138$gexG <- GSE22138$gex[,names(x)]


######################### Cluster diagram (with dChip colors) ############

library(geneplotter)
row.dist <- as.dist(1 - cor(GSE22138$gexG))
col.dist <- as.dist(1 - cor(t(GSE22138$gexG)))

c2<-GSE22138$clin$d.mfs
spcol <- ifelse(as.numeric(c2) == as.numeric(c2)[1], "grey40", "grey80")
    # Set color (here greys), for horizontal bar. Each color one class

hv <- heatmap(t(GSE22138$gexG), Colv=as.dendrogram(hclust(col.dist,
    method="average")),
Rowv=as.dendrogram(hclust(row.dist, method="average")), keep.dendro=TRUE)
    # Pearson's Distances based heatmap

heatmap(t(GSE22138$gexG)[hv$rowInd,], Rowv = NA, labRow = " ",
    Colv=reorder(hv$Colv,1:n, agglo.FUN= mean), col = dChip.colors(256),
    margin = c(7,7),
ColSideColors=spcol) # with an horizontal color bar to annotate the two
    groups and without gene names and gene dendrograms. To remove bar,
    remove ColSideColors
```

```
postscript(file=paste("cluster", anname, ".pdf", sep=""))
heatmap(t(GSE22138$gexG)[hv$rowInd,], Rowv = NA, labRow =
    colnames(GSE22138$gexG)[hv$rowInd], Colv=reorder(hv$Colv,1:n,
    agglo.FUN= mean), col = dChip.colors(256), margin = c(7,7),
    ColSideColors=spcol) # with an horizontal color bar to annotate the
    two groups and without gene names and gene dendrograms
dev.off()


####### CALCULATE RISK INDEX ############

a<-intersect(topTable(fit2, coef=1,
    adjust="BH")$ID[1],colnames(GSE22138$gex))

GSE22138$sigdata <- cbind( GSE22138$gex[,a] )

GSE22138$clin$ri <- apply(GSE22138$sigdata,1,function(x)
    mean(x,na.rm=TRUE))

GSE22138$clin$ri.rank <-
    rank(GSE22138$clin$ri)/sum(!is.na(GSE22138$clin$ri))


####### SURVIVAL ANALYSIS ############

png(file=paste("KM_Risk", anname, ".png", sep=""), width=700, height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~factor(ri.rank>0.5),data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
    Risk",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("Low Risk", "High Risk"), lty = 2:3)
dev.off()

summary(coxph(Surv(t.mfs/12, d.mfs) ~ factor(ri.rank>0.5),
    data=GSE22138$clin))

summary(coxph(Surv(t.mfs/12, d.mfs) ~ retinal.detachment.d,
    data=GSE22138$clin))
summary(coxph(Surv(t.mfs/12, d.mfs) ~ retinal.detachment.d +
    strata(gender.d), data=GSE22138$clin))

png(file=paste("KM_gender", anname, ".png", sep=""), width=700,
    height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~gender.d,data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
```

```
   Gender",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("Male", "Female"), lty = 2:3)
dev.off()

png(file=paste("KM_eyeside", anname, ".png", sep=""), width=700,
   height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~eye.side.d,data=GSE22138$clin),
   conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
   EyeSide",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("DX", "SX"), lty = 2:3)
dev.off()

png(file=paste("KM_retinal", anname, ".png", sep=""), width=700,
   height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~retinal.detachment.d,data=GSE22138$clin),
   conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
   RetinalDetachment",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("No Detach", "Yes Detach"), lty = 2:3)
dev.off()

save(GSE22138,mysamples,file="GSE22138.rda")

rm(list=ls())
```

A.3.2. R Code of Components

As part of the clinical trial case study (see Section 5.7), the R script was split into 9 different components *Read data*, *Normalize*, *QC degradations*, *QC intensities*, *QC log intensity*, *Volcano and Venn*, *Heatmaps*, *Risk calculation*, and *Kaplan-Meier*. The following code listings present the code of the individual components:

Read data

Listing A.9: R code of the *Read data* component

```
setwd("C:/pMed_testdata")

####### LIBRARIES ############

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
```

```
library(limma)
library(survival)

####### READING DATA ############

#Reading AffyBatch from CEL files
mysamples <- ReadAffy()

# Adjusting sampleNames
smpls <- sub(".CEL","", sub(paste(getwd(),"/",
    sep=""),"",sampleNames(mysamples)))
sampleNames(mysamples) <- smpls
sampleNames(mysamples)
print(smpls)

ls()

### safe workspace on R Server
save.image()
```

Normalize

Listing A.10: R code of the *Normalize* component

```
setwd("C:/pMed_testdata")
load(".Rdata")

ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

####### NORMALIZATION ############
esetmysamples <- rma(mysamples)

# Saving QC images

degmysamples <- AffyRNAdeg(mysamples)

anname<-"GSE22138"
```

QC degradation

Listing A.11: R code of the *QC degradation* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

png(plotAffyRNAdeg)
degmysamples <- AffyRNAdeg(mysamples)
summaryAffyRNAdeg(degmysamples)
write.table(degmysamples[c(2,5,6)], file="AffyRNAdeg.csv", sep=",")
plotAffyRNAdeg(degmysamples)
dev.off()

png(file=paste("Degradation", anname, ".png", sep=""))
plotAffyRNAdeg(degmysamples)
dev.off()
```

QC intensities:

Listing A.12: R code of the *QC intensities* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

png(file=paste("BoxplotUnNormal", anname, ".png", sep=""), width=700,
    height=700)
```

```
par(mar=c(8,4,4,2))
boxplot(mysamples, names=sampleNames(esetmysamples),las=3,
    cex.axis=0.7,outline=F)

dev.off()

png(file=paste("BoxplotNormal", anname, ".png", sep=""), width=700,
    height=700)
par(mar=c(8,4,4,2))
boxplot(exprs(esetmysamples),
    names=sampleNames(esetmysamples),las=3,cex.axis=0.7,outline=F)

dev.off()

### safe workspace on R Server
save.image()
```

QC log intensity

Listing A.13: R code of the *QC log intensity* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

#png(histpng)
#hist(mysamples)

#dev.off()


png(file=paste("Intensities", anname, ".png", sep=""))
hist(mysamples,col="blue")

dev.off()
```

Volcano and Venn

Listing A.14: R code of the *Volcano and Venn* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

GSE22138<-new.env()
esetm<-exprs(esetmysamples)

GSE22138$gex<-t(esetm)

GSE22138$clin<-read.table(file="GSE22138_clin.csv",header=T,row.names=1)
clin<-read.table(file="GSE22138_clin.csv",header=T,row.names=1)
GSE22138$clin<-clin[rownames(GSE22138$gex),]

# differentially expressed genes
design <- model.matrix(~0+factor(d.mfs), data=GSE22138$clin)
colnames(design) <- c("NoMet","Met")
x<-t(GSE22138$gex[rownames(design),])
n<-ncol(x)
fit <- lmFit(x, design)

contrast.matrix <- makeContrasts(Met-NoMet, levels=design)
fit2 <- contrasts.fit(fit, contrast.matrix)
fit2 <- eBayes(fit2)
# A list of top genes differential expressed in group2 versus group1 can
    be obtained from

toptable<-topTable(fit2, coef=1, adjust="BH")


results <- decideTests(fit2,p.value=0.56,lfc=1)
#A Venn diagram showing numbers of genes significant in each comparison
    can be obtained from
```

```
png(file="VennDiagram.png")
vennDiagram(results, main="Venn Diagram - p.value (adj) < 0.56, FC > 2 -
    Stat INCORRECT - JUST FOR TRAINING")
dev.off()

pdf(file="prova_volcano.pdf")
volcanoplot(fit2, coef=1,highlight=6,main="Detach-NoDetach")
dev.off()

x<-results@.Data[abs(results@.Data[,1])==1,]


result_names<-names(x)
GSE22138$gexG <- GSE22138$gex[,names(x)]


### safe workspace on R Server
save.image()
```

Heatmaps

Listing A.15: R code of the *Heatmap* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

library(geneplotter)
row.dist <- as.dist(1 - cor(GSE22138$gexG))
col.dist <- as.dist(1 - cor(t(GSE22138$gexG)))

c2<-GSE22138$clin$d.mfs
spcol <- ifelse(as.numeric(c2) == as.numeric(c2)[1], "grey40", "grey80")
    # Set color (here greys), for horizontal bar. Each color one class


png(file=paste("Heatmap2", anname, ".png", sep=""))
```

```
hv <-heatmap(t(GSE22138$gexG), Colv=as.dendrogram(hclust(col.dist,
    method="average")),
Rowv=as.dendrogram(hclust(row.dist, method="average")), keep.dendro=TRUE)
    # Pearson's Distances based heatmap
dev.off()

png(file=paste("Heatmap", anname, ".png", sep=""))
heatmap(t(GSE22138$gexG)[hv$rowInd,], Rowv = NA, labRow = " ",
    Colv=reorder(hv$Colv,1:n, agglo.FUN= mean), col = dChip.colors(256),
    margin = c(7,7),
ColSideColors=spcol)
dev.off()

#postscript
png(file=paste("cluster", anname, ".pdf", sep=""))
heatmap(t(GSE22138$gexG)[hv$rowInd,], Rowv = NA, labRow =
    colnames(GSE22138$gexG)[hv$rowInd], Colv=reorder(hv$Colv,1:n,
    agglo.FUN= mean), col = dChip.colors(256), margin = c(7,7),
    ColSideColors=spcol) # with an horizontal color bar to annotate the
    two groups and without gene names and gene dendrograms
dev.off()


### safe workspace on R Server
save.image()
```

Risk calculation

Listing A.16: R code of the *Risk calculation* component

```
setwd("C:/pMed_testdata")
load(".Rdata")

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

a<-intersect(topTable(fit2, coef=1,
    adjust="BH")$ID[1],colnames(GSE22138$gex))

GSE22138$sigdata <- cbind( GSE22138$gex[,a] )
```

```
GSE22138$clin$ri <- apply(GSE22138$sigdata,1,function(x)
    mean(x,na.rm=TRUE))

GSE22138$clin$ri.rank <-
    rank(GSE22138$clin$ri)/sum(!is.na(GSE22138$clin$ri))


### safe workspace on R Server
save.image()
```

Kaplan-Meier plots and tests

Listing A.17: R code of the *Kaplan Meier* component

```
setwd("C:/pMed_testdata")
load(".Rdata")
ls()

library(Biobase)
library(affy)
library(gcrma)
library(genefilter)
library(geneplotter)
library(annaffy)
library(limma)
library(survival)

png(file=paste("KM_Risk", anname, ".png", sep=""), width=700, height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~factor(ri.rank>0.5),data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
    Risk",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("Low Risk", "High Risk"), lty = 2:3)
dev.off()

summary1<-summary(coxph(Surv(t.mfs/12, d.mfs) ~ factor(ri.rank>0.5),
    data=GSE22138$clin))

summary2<-summary(coxph(Surv(t.mfs/12, d.mfs) ~ retinal.detachment.d,
    data=GSE22138$clin))
summary3<-summary(coxph(Surv(t.mfs/12, d.mfs) ~ retinal.detachment.d +
    strata(gender.d), data=GSE22138$clin))

png(file=paste("KM_gender", anname, ".png", sep=""), width=700,
    height=700)
```

```
plot(survfit(Surv(t.mfs/12,d.mfs)~gender.d,data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
    Gender",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("Male", "Female"), lty = 2:3)
dev.off()

png(file=paste("KM_eyeside", anname, ".png", sep=""), width=700,
    height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~eye.side.d,data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
    EyeSide",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("DX", "SX"), lty = 2:3)
dev.off()

png(file=paste("KM_retinal", anname, ".png", sep=""), width=700,
    height=700)
plot(survfit(Surv(t.mfs/12,d.mfs)~retinal.detachment.d,data=GSE22138$clin),
    conf.int=F, xlab="Time (Years)", main="K-M (MFS) -
    RetinalDetachment",ylab="follow-up probability",lty = 2:3)
legend(8, .9, c("No Detach", "Yes Detach"), lty = 2:3)
dev.off()


save(GSE22138,mysamples,file="GSE22138.rda")


#rm(list=ls())
```

Bibliography

[1] The GridLab project at `http://www.gridlab.org/`.

[2] The GridOneD project at `http://www.gridoned.org/`.

[3] Web Services Resource Framework (WSRF) standard v1.2, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`.

[4] Web Services Business Process Execution Language Version 2.0. Technical report, OASIS Web Services Business Process Execution Language (WSBPEL) TC, April 2007.

[5] SPARQL Protocol And RDF Query Language, W3C, `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[6] Activiti. Activiti BPM platform, `http://activiti.org/`, 2012.

[7] Software AG. ARIS platform, `http://www.softwareag.com/de/products/aris_platform/default.asp`, 2012.

[8] Ali Shaikh Ali and Ian J. Taylor. Web Services Composition for Distributed Data Mining. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, ICPPW '05, pages 11–18, Washington, DC, USA, 2005. IEEE Computer Society.

[9] Salman AlSairafi, Filippia-Sofia Emmanouil, Moustafa Ghanem, Nikolaos Giannadakis, Yike Guo, Dimitrios Kalaitzopoulos, Michelle Osmond, Anthony Rowe, Jameel Syed, and Patrick Wendel. The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *High-Performance Computing Applications*, 17(3):297–315, 2003.

[10] Ilkay Altintas, Adam Birnbaum, Kim K. Baldridge, Wibke Sudholt, Mark Miller, Celine Amoreira, Yohann Potier, and Bertram Ludaescher. A Framework for the Design and Reuse of Grid WorkFlows. In *International Workshop on Scientific Aspects of Grid Computing*, pages 120–133. Springer-Verlag, 2005.

[11] Per Kragh Andersen, Ornulf Borgan, Richard D. Gill, and Niels Keiding. *Statistical Models Based on Counting Processes*. Springer, 1993.

[12] Mario Antonioletti, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil P. Chue Hong, Brian Collins, Neil Hardman, Hu Alastair C., Alan Knox, Mike Jackson, Amy Krause, Simon Laws, James Magowan, Norman W. Paton, Dave Pearson, Tom Sugden, Paul Watson, and Martin Westhead. The design and implementation

of Grid database services in OGSA-DAI: Research Articles. *Concurr. Comput. : Pract. Exper.*, 17:357–376, February 2005.

[13] ArrayExpress. `http://www.ebi.ac.uk/arrayexpress/`, 2011.

[14] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition).* John Wiley Interscience, March 2004.

[15] D. Atwood. BPM Process Patterns: Repeatable Design for BPM Process Models. BPTrends, May, 2006.

[16] Jeff Banfield. Rweb: Web-based Statistical Analysis. *Journal of Statistical Software*, 4(1):1–15, 3 1999.

[17] Kahina Bessai, Bruno Claudepierre, Oumaima Saidani, and Selmin Nurcan. Context-aware Business Process Evaluation and Redesign. In *Int. Workshop on Business Process Management, Design and Support, at Int. Conference on Advanced Information Systems, Montpellier, France. Proceedings*, pages 1–10. Springer, 2008.

[18] Kanishka Bhaduri, Kamalika Das, Kun Liu, and Hillol Kargupta. Distributed Data Mining Bibliography, `http://www.csee.umbc.edu/~hillol/DDMBIB/9`, 2009.

[19] BioMoby Consortium, Mark D. Wilkinson, Martin Senger, Edward Kawas, Richard Bruskiewich, Jerome Gouzy, Celine Noirot, Philippe Bardou, Ambrose Ng, Dirk Haase, Enrique de Andres d. e. . A. Saiz, Dennis Wang, Frank Gibbons, Paul M. Gordon, Christoph W. Sensen, Jose Manuel Rodriguez M. Carrasco, José M. Fernández, Lixin Shen, Matthew Links, Michael Ng, Nina Opushneva, Pieter B. Neerincx, Jack A. Leunissen, Rebecca Ernst, Simon Twigger, Bjorn Usadel, Benjamin Good, Yan Wong, Lincoln Stein, William Crosby, Johan Karlsson, Romina Royo, Iván Párraga, Sergio Ramírez, Josep Lluis L. Gelpi, Oswaldo Trelles, David G. Pisano, Natalia Jimenez, Arnaud Kerhornou, Roman Rosset, Leire Zamacola, Joaquin Tarraga, Jaime Huerta-Cepas, Jose María M. Carazo, Joaquin Dopazo, Roderic Guigo, Arcadi Navarro, Modesto Orozco, Alfonso Valencia, M. Gonzalo Claros, Antonio J. Pérez, Jose Aldana, Mar M. Rojano, Raul Fernandez-Santa Cruz, Ismael Navas, Gary Schiltz, Andrew Farmer, Damian Gessler, Heiko Schoof, and Andreas Groscurth. Interoperability with Moby 1.0–it's better than sharing your toothbrush! *Briefings in bioinformatics*, 9(3):220–231, May 2008.

[20] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[21] P. Bremer. Erstellung einer Datenbasis von Workflowreihen aus realen Anwendungen (in german). Diploma Thesis, University of Bonn, 2010.

[22] Peter Brezany, Ivan Janciak, and A Min Tjoa. GridMiner: A Fundamental Infrastructure for Building Intelligent Grid Systems. *Web Intelligence, IEEE / WIC / ACM International Conference on*, 0:150–156, 2005.

[23] Mathias Brochhausen, Andrew D. Spear, Cristian Cocos, Gabriele Weiler, Luis Martín, Alberto Anguita, Holger Stenzhorn, Evangelia Daskalaki, Fatima Schera, Ulf Schwarz, Stelios Sfakianakis, Stephan Kiefer, Martin Dörr, Norbert Graf, and Manolis Tsiknakis. The ACGT Master Ontology and its applications - Towards an ontology-driven cancer research and management system. *J. of Biomedical Informatics*, 44:8–25, February 2011.

[24] Anca Bucur, Stefan Rüping, Thierry Sengstag, Stelios Sfakianakis, Manolis Tsiknakis, and Dennis Wegener. The ACGT project in retrospect: Lessons learned and future outlook. *Procedia Computer Science*, 4:1119 – 1128, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[25] The caBIG Project (USA) Cancer Biomedical Informatics Grid. `https://cabig.nci.nih.gov/"`.

[26] Mario Cannataro, Domenico Talia, and Paolo Trunfio. Distributed data mining on the grid. *Future Gener. Comput. Syst.*, 18:1101–1112, October 2002.

[27] Bruce Carlson. SNPs A Shortcut to Personalized Medicine. *Genetic Engineering Biotechnology News*, 28(12), 2008.

[28] Sebastian Celis and David R. Musicant. Weka-Parallel: Machine Learning in Parallel. Technical report, Carleton College, CS TR, 2002.

[29] Nadia Cerezo and Johan Montagnat. Scientific workflow reuse through conceptual workflows on the virtual imaging platform. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 1–10, New York, NY, USA, 2011. ACM.

[30] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. CRISP-DM 1.0 Step-by-step data mining guide. Technical report, The CRISP-DM consortium, August 2000.

[31] W. K. Cheung, Xiao-Feng Zhang, Ho-Fai Wong, Jiming Liu, Zong-Wei Luo, and F. C. H. Tong. Service-Oriented Distributed Data Mining. *Internet Computing, IEEE*, 10(4):44–54, 2006.

[32] David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. Programming scientific and distributed workflow with Triana services: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1021–1037, August 2006.

[33] Workflow Management Coalition. XML Process Definition Language (XPDL), `http://www.wfmc.org/xpdl.html`, 2008.

[34] ACGT consortium. Advancing Clinico Genomic Trials on Cancer - The ACGT project (EU), project web site: `http://www.eu-acgt.org`, 2010.

[35] DataMiningGrid consortium. The DataMiningGrid project (Data Mining Tools and Services for Grid Computing Environments). Project website: `http://www.DataMiningGrid.org`, 2006.

[36] Oscar Corcho, Pinar Alper, Ioannis Kotsiopoulos, Paolo Missier, Sean Bechhofer, and Carole Goble. An overview of S-OGSA: A Reference Semantic Grid Architecture. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2):102–115, jun 2006.

[37] CRAN. The Comprehensive R Archive Network, `http://cran.r-project.org/`.

[38] Vasa Curcin, Moustafa Ghanem, Yike Guo, Martin Kohler, Anthony Rowe, Jameel Syed, and Patrick Wendel. Discovery net: towards a grid of knowledge discovery. In *Proc. of the Eigth International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, August 2002.

[39] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *J. Grid Comput.*, 1(1):25–39, 2003.

[40] Christine Desmedt. ACGT Deliverable 12.6 - Review and extension of the ACGT clinical studies., 2008.

[41] Mahmoud El-Gayyar, Y. Leng, and Armin Cremers. Distributed Management of Scientific Workflows in SWIMS. *International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, 0:327–331, 2010.

[42] BASE BioArray Software Environment. `http://base.thep.lu.se/`.

[43] Arbeitsgemeinschaft Media-Analyse e.V. Infos für Presse, Radio, TV und Online, Press Release 17.12.2007, `http://www.agma-mmc.de`, 2007.

[44] P. Farmer, H. Bonnefoi, V. Becette, M. Tubiana-Hulin, P. Fumoleau, D. Larsimont, G. Macgrogan, J. Bergh, D. Cameron, D. Goldstein, S. Duss, A. L. Nicoulaz, C. Brisken, M. Fiche, M. Delorenzi, and R. Iggo. Identification of molecular apocrine breast tumours by microarray analysis. *Oncogene*, 24(29):4660–4671, 2005. Swiss Institute of Bioinformatics, Lausanne, Switzerland.

[45] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17:37–54, 1996.

[46] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, Version 1.0. Informational Document, Global Grid Forum (GGF), 2005.

[47] Ian Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[48] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13. Springer-Verlag, 2005.

[49] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. 2002.

[50] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Jounral of Supercomputer Applications*, 15(3), 2001.

[51] A. Frank and A. Asuncion. (UCI) Machine Learning Repository, 2010.

[52] Eibe Frank, Mark Hall, Len Trigg, Geoffrey Holmes, and Ian H. Witten. Data mining in bioinformatics using weka. *Bioinformatics*, 20:2479–2481, 2004.

[53] Michael Y. Galperin and Guy Cochrane. The 2011 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection. *Nucleic Acids Research*, 39(Database-Issue):1–6, 2011.

[54] J. J. Garcia Adeva and R. Calvo. Mining Text with Pimiento. *Internet Computing, IEEE*, 10(4):27–35, 2006.

[55] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[56] GEO. `http://www.ncbi.nlm.nih.gov/geo/`, 2011.

[57] Jan Erik Gewehr, Martin Szugat, and Ralf Zimmer. BioWeka–extending the Weka framework for bioinformatics. *Bioinformatics*, 23(5):651–653, 2007.

[58] Carole A. Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li, and David De Roure. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(suppl 2):W677–W682, July 2010.

[59] Antoon Goderis. Workflow Re-use and Discovery in Bioinformatics. PhD Thesis, School of Computer Science, The University of Manchester, 2008.

[60] Antoon Goderis, Ulrike Sattler, Phillip W. Lord, and Carole A. Goble. Seven Bottlenecks to Workflow Reuse and Repurposing. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web*

Conference, volume 3729 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005.

[61] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86+, August 2010.

[62] Daniel Grose, Rob Crouchley, Ties Van Ark, Rob Allan, John Kewley, and Mark Hayes. SabreR: Grid-enabling the analysis of multi-process random effect response data in R. In *Proceedings of the 2nd International Conference on e-Social Science*, 2006.

[63] Dorgival Guedes, Wagner Meira Jr., and Renato Ferreira. Anteater: A Service-Oriented Architecture for High-Performance Data Mining. *IEEE Internet Computing*, 10:36–43, 2006.

[64] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, December 2001.

[65] Mark Hall and Peter Reutemann. WEKA KnowledgeFlow Tutorial for Version 3-5-8, University of Waikato, 2008.

[66] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperBusiness, April 1994.

[67] G. Hardiman. Microarray platforms–comparisons and contrasts. *Pharmacogenomics*, 5(5):487–502, July 2004.

[68] M. A. Harris, J. Clark, A. Ireland, J. Lomax, M. Ashburner, R. Foulger, K. Eilbeck, S. Lewis, B. Marshall, C. Mungall, J. Richter, G. M. Rubin, J. A. Blake, C. Bult, M. Dolan, H. Drabkin, J. T. Eppig, D. P. Hill, L. Ni, M. Ringwald, R. Balakrishnan, J. M. Cherry, K. R. Christie, M. C. Costanzo, S. S. Dwight, S. Engel, D. G. Fisk, J. E. Hirschman, E. L. Hong, R. S. Nash, A. Sethuraman, C. L. Theesfeld, D. Botstein, K. Dolinski, B. Feierbach, T. Berardini, S. Mundodi, S. Y. Rhee, R. Apweiler, D. Barrell, E. Camon, E. Dimmer, V. Lee, R. Chisholm, P. Gaudet, W. Kibbe, R. Kishore, E. M. Schwarz, P. Sternberg, M. Gwinn, L. Hannick, J. Wortman, M. Berriman, V. Wood, N. de la Cruz, P. Tonellato, P. Jaiswal, T. Seigfried, R. White, and Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research*, 32 (Database Issue):258–261, 2004.

[69] Melanie Hilario, Alexandros Kalousis, Phong Nguyen, and Adam Woznica. A Data Mining Ontology for Algorithm Selection and Meta-Learning. *Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-09)*, (Dm):76–87, 2009.

[70] M F Hornick, E Marcad, and S Venkayala. *Java data mining: strategy, standard, and practice: a practical guide for architecture, design, and implementation.* Morgan Kaufmann, 2007.

[71] K. Hornik. R FAQ - Frequently Asked Questions on R. Version 2.13.2011-07-05, 2011.

[72] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(Web Server issue):W729–732, July 2006.

[73] IBM. IBM SPSS Statistics, `http://www-01.ibm.com/software/de/analytics/spss/products/statistics/`, 2012.

[74] Affymetrix Inc. `http://www.affymetrix.com`.

[75] Illumina Inc. `http://www.illumina.com/`.

[76] Intalio Inc. Intalio BPMS, `http://www.intalio.com/bpms`, 2010.

[77] SAS Institute Inc. SAS Enterprise Miner, `http://www.sas.com/technologies/analytics/datamining/miner/`, 2012.

[78] iWebCare Project. Deliverable D01 - Business process model of e-gov fraud detection processes in the health care domain, http://iwebcare.iisa-innov.com/documents/D1-Business Process Modeling v4.3.zip, 2006.

[79] jBoss community. jBPM, `http://www.jboss.org/jbpm`, 2012.

[80] J. Karlsson, V. Martin-Requena, J. Rios, and Oswaldo Trelles. Workflow composition and enactment using jORCA. In *Proceedings of 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, 2010.

[81] Nicholas T. Karonis, Brian R. Toonen, and Ian T. Foster. MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, 2003.

[82] R Khoussainov. Grid-enabled Weka: A toolkit for machine learning on the grid. *ERCIM News*, 59:4748, 2004.

[83] J-U Kietz, F Serban, A Bernstein, and S Fischer. Towards cooperative planning of data mining workflows. In *Proc of the ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-09)*, 2009.

[84] J U Kietz, F Serban, A Bernstein, and S Fischer. Data mining workflow templates for intelligent discovery assistance in RapidMiner. In *Proc of RCOMM'10*, SEP 2010.

[85] Na (Michael) Li and A. J. Rossini. rpvm: R interface to PVM, `http://cran.r-project.org/web/packages/rpvm/rpvm.pdf`.

[86] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1039–1065, August 2006.

[87] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is bioinformatics? A proposed definition and overview of the field. *Methods of information in medicine*, 40(4):346–358, 2001.

[88] Xiaosong Ma, Jiangtian Li, and Nagiza F. Samatova. Automatic Parallelization of Scripting Languages: Toward Transparent Desktop Parallel Computing. *Parallel and Distributed Processing Symposium, International*, 0:298, 2007.

[89] Oscar Marbán, Javier Segovia, Ernestina Menasalvas, and Covadonga Fernández-Baizán. Toward data mining engineering: A software engineering approach. *Inf. Syst.*, 34:87–107, March 2009.

[90] V. Martin-Requena, J. Rios, M. Garcia, S. Ramirez, and O. Trelles. jORCA: easily integrating bioinformatics Web Services. *Bioinformatics*, 26(4):553, 2010.

[91] R.J. Mayer and P.S. Dewitte. Delivering Results: Evolving BPR from art to engineering. In: Elzinga, D.J., Gulledge, T.R., Lee, C.Y. (eds.): Business process engineering: advancing the state of the art, 1998.

[92] Microsoft. SQL Server 2012 Data Mining, `http://www.sqlserverdatamining.com/ssdm/`, 2012.

[93] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In Lyle Ungar, Mark Craven, Dimitrios Gunopulos, and Tina Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, August 2006. ACM.

[94] Michael Mock, Dennis Wegener, Anuj Sharma, and Francesca Buffa. p-medicine project deliverable D11.1: Initial definition of data mining patterns, 2012.

[95] MPI. Message Passing Interface Forum: `http://www.mpi-forum.org`.

[96] Oracle. Oracle Data Mining, `http://www.oracle.com/technetwork/database/options/advanced-analytics/odm/index.html`, 2012.

[97] p-medicine consortium. From data sharing and integration via VPH models to personalized medicine - The p-medicine project (EU), project web site: `http://www.p-medicine.eu/`, 2011.

[98] María Pérez, Alberto Sánchez, Pilar Herrero, Vctor Robles, and José Peña. Adapting the Weka Data Mining Toolkit to a Grid Based Environment. In Piotr Szczepaniak, Janusz Kacprzyk, and Adam Niewiadomski, editors, *Advances in Web Intelligence*, volume 3528 of *Lecture Notes in Computer Science*, pages 819–820. Springer Berlin / Heidelberg, 2005.

[99] Anne Planche, Marina Bacac, Paolo Provero, Carlo Fusco, Mauro Delorenzi, Jean-Christophe Stehle, and Ivan Stamenkovic. Identification of Prognostic Molecular Features in the Reactive Stroma of Human Breast and Prostate Cancer. *PLoS ONE*, 6(5):e18640, 2011.

[100] PubMed. `http://www.ncbi.nlm.nih.gov/pubmed/`, 2011.

[101] Juliusz Pukacki, Micha Kosiedowski, Rafa Mikoajczak, Marcin Adamski, Piotr Grabowski, Micha Jankowski, Mirosaw Kupczyk, Cezary Mazurek, Norbert Meyer, Jarek Nabrzyski, Tomasz Piontek, Michael Russell, Maciej Stroiski, and Marcin Wolski. Programming Grid Applications with Gridge, Computational Methods in Science and Technology vol. 12, Poznan, 2006.

[102] PVM. Parallel Virtual Machine: `http://www.csm.ornl.gov/pvm/pvm_home.html`.

[103] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[104] Ha Reijers and S Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005.

[105] rJava. A simple R-to-Java interface, `http://www.rforge.net/rJava/`.

[106] David S. Roos. Bioinformatics–Trying to Swim in a Sea of Data. *Science*, 291(5507):1260–1261, February 2001.

[107] J. Ros, J. Karlsson, and O. Trelles. Magallanes: a web services discovery and automatic workflow composition tool. *BMC Bioinformatics*, 10(1):334, 2009.

[108] Cornelius Rosse and José L. V. Mejino, Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *J. of Biomedical Informatics*, 36:478–500, December 2003.

[109] Simona Rossi, Marie-Luise Christ-Neumann, Stefan Rüping, Francesca Buffa, Dennis Wegener, Gordon McVie, Peter Coveney, Norbert Graf, and Mauro Delorenzi. p-Medicine: From data sharing and integration via VPH models to personalized medicine. *ecancermedicalscience*, 5(218), 2011.

[110] Simona Rossi, Masayoshi Shimizu, Elisa Barbarotto, Milena S Nicoloso, Federica Dimitri, Deepa Sampath, Muller Fabbri, Susan Lerner, Lynn L Barron, Laura Z Rassenti, Li Jiang, Lianchun Xiao, Jianhua Hu, Paola Secchiero, Giorgio Zauli, Stefano Volinia, Massimo Negrini, William Wierda, Thomas J Kipps, William Plunkett, Kevin R Coombes, Lynne V Abruzzo, Michael J Keating, and George A Calin. microRNA fingerprinting of CLL patients with chromosome 17p deletion identify a miR-21 score that stratifies early survival. *Blood*, 116:945–52, 2010.

[111] A.J. Rossini, Luke Tierney, and Na Li. Simple Parallel Statistical Computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, June 2007.

[112] Boris Rottmann. Gegenüberstellung von Modellierungsansätzen für Workflows im Kontext von Bioinformatik-Szenarien des Projekts p-medicine (in german), Bachelor Thesis, Hochschule Bonn-Rhein-Sieg, 2011.

[113] Stefan Rüping, Natalja Punko, Björn Günter, and Henrik Grosskreutz. Procurement Fraud Discovery using Similarity Measure Learning. *Tran. CBR*, 1(1):37–46, 2008.

[114] Stefan Rüping, Dennis Wegener, and Philipp Bremer. Re–using Data Mining Workflows. *Proceedings of the ECML PKDD 2010 Workshop on Third–Generation Data Mining: Towards Service–Oriented Knowledge Discovery (SoKD́10)*, 2010.

[115] Stefan Rüping, Dennis Wegener, Stelios Sfakianakis, and Thierry Sengstag. Workflows for intelligent monitoring using proxy services. In *Healthgrid research, innovation and business case: Proceedings of HealthGrid 2009, , 29 June - 1 July, Berlin, Germany*, pages 277–282, Amsterdam, Netherlands, 2009. IOS Press.

[116] Rok Rupnik and Jurij Jaklic. The Deployment of Data Mining into Operational Business Processes. *Data Mining and Knowledge Discovery*, (February), 2009.

[117] Nick Russell, Arthur, Wil M. P. van der Aalst, and Natalya Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPMcenter.org, 2006.

[118] Web services architecture working group. `http://www.w3.org/2002/ws/arch/`, 2006.

[119] Stelios Sfakianakis, Norbert M. Graf, Alexander Hoppe, Stefan Rüping, Dennis Wegener, Lefteris Koumakis, and George Zacharioudakis. Building a System for Advancing Clinico-Genomic Trials on Cancer. In *Proceedings of the Workshops of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations, AIAI 2009, Thessaloniki, Greece, April 23-25, 2009*, pages 36–47. CEUR-WS.org, 2009.

[120] Sumana Sharma and Kweku-Muata Osei-Bryson. Framework for formal implementation of the business understanding phase of data mining projects. *Expert Syst. Appl.*, 36:4114–4124, March 2009.

[121] Colin Shearer. The CRISP-DM Model: The New Blueprint for Data Mining. *Journal of Data Warehousing*, 5(4), 2000.

[122] Matthew Shields. Control- Versus Data-Driven Workflows. In Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 167–173. Springer London, 2007.

[123] Vlado Stankovski, Martin Swain, Valentin Kravtsov, Thomas Niessen, Dennis Wegener, Jörg Kindermann, and Werner Dubitzky. Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. *Future Generation Comp. Syst.*, 24(4):259–279, 2008.

[124] Vlado Stankovski, Martin Swain, Valentin Kravtsov, Thomas Niessen, Dennis Wegener, M. Röhm, Jernej Trnkoczy, Michael May, Jürgen Franke, Assaf Schuster, and

Werner Dubitzky. Digging Deep into the Data Mine with DataMiningGrid. *IEEE Internet Computing*, 12(6):69–76, 2008.

[125] Lincoln D. Stein. Genome annotation: from sequence to biology. *Nature Reviews Genetics*, 2(7):493–503, July 2001.

[126] A. Tagaris, G. Konnis, X. Benetou, T. Dimakopoulos, K. Kassis, N. Athanasiadis, S. Rüping, H. Grosskreutz, and D. Koutsouris. Integrated Web Services Platform for the facilitation of fraud detection in health care e-government services. In *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on. Proceedings*, 2009.

[127] Domenico Talia, Paolo Trunfio, and Oreste Verta. Weka4WS: a WSRFenabled Weka Toolkit for Distributed Data Mining on Grids. In *Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005*, pages 309–320. Springer-Verlag, 2005.

[128] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana Workflow Environment: Architecture and Applications. In Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 320–339. Springer London, 2007.

[129] YAWL Team. YAWL: Yet Another Workflow Language, `http://www.yawlfoundation.org/`, 2012.

[130] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[131] Gruber TR. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition; 5(2): 199-220., 1993.

[132] Manolis Tsiknakis. ACGT project deliverable 2.1 - User requirements and specification of the ACGT internal clinical trial, 2006.

[133] Manolis Tsiknakis, Mathias Brochhausen, J. Nabrzyski, J. Pucacki, Stelios Sfakianakis, George Potamias, C. Desmedt, and Dimitris Kafetzopoulos. A Semantic Grid Infrastructure Enabling Integrated Access and Analysis of Multilevel Biomedical Data in Support of Postgenomic Clinical Trials on Cancer. *IEEE Transactions on Information Technology in Biomedicine*, 12(2):205–217, 2008.

[134] The CancerGrid Project (UK). `http://www.cancergrid.org/`.

[135] Simon Urbanek. Rserve - A Fast Way to Provide R Functionality to Applications. In *Proc. of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), ISSN 1609-395X, Eds.: Kurt Hornik, Friedrich Leisch and Achim Zeileis, 2003 (`http://rosuda.org/Rserve`*, 2003.

[136] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesińska, Rutger F. H. Hofman, Ceriel J. H. Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a flexible and efficient Java-based Grid programming environment: Research Articles. *Concurr. Comput.: Pract. Exper.*, 17:1079–1107, June 2005.

[137] Joaquin Vanschoren and Larisa Soldatova. Exposé: An ontology for data mining experiments. In *International Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-2010),*, pages 31–46, September 2010.

[138] Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A Grid service broker for scheduling e-Science applications on global data Grids: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:685–699, May 2006.

[139] Dennis Wegener. DataMiningGrid project deliverable Deliverable D32(2): Software: Java based distributed workflow editor module, 2006.

[140] Dennis Wegener, Alberto Anguita, and Stefan Rüping. Enabling the reuse of data mining processes in healthcare by integrating data semantics. In *Proceedings of the 3th International Workshop on Knowledge Representation for Health Care (KR4HC 2011)*, 2011.

[141] Dennis Wegener, Dirk Hecker, Christine Körner, Michael May, and Michael Mock. Parallelization of R-programs with GridR in a GPS-trajectory mining application. In *Proceedings of the First Ubiquitous Knowledge Discovery Workshop, UKD08, in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008), Antwerp, Belgium, 2008*, 2008.

[142] Dennis Wegener and Michael May. Extensibility of grid-enabled data mining platforms: A case study. In *Proceedings of the 5th International Workshop on Data Mining Standards, Services and Platforms, DM-SSP '07 , in conjunction with the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007), August 12, 2007, San Jose, California, USA*, pages 13–22, New York, USA, 2007. Association for Computing Machinery -ACM-.

[143] Dennis Wegener and Michael May. Specification of distributed data mining workflows with DataMiningGrid. In W. Dubitzky, editor, *Data mining techniques in grid computing environments*, pages 219–234. John Wiley and Sons, Hoboken, NJ, 2008.

[144] Dennis Wegener, Michael Mock, Deyaa Adranale, and Stefan Wrobel. Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters. In *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops, ICDM 2009 , December 06, Miami, Florida, USA*, pages 296–301, Washington, DC, USA, 2009. IEEE Computer Society.

[145] Dennis Wegener, Simona Rossi, Francesca Buffa, Mauro Delorenzi, and Stefan Rüping. Towards an Environment for Data Mining based Analysis Processes in Bioinformatics & Personalized Medicine. In *Proceedings of the BIBM 2011 Workshops*, pages 570–577, 2011.

[146] Dennis Wegener and Stefan Rüping. On Integrating Data Mining into Business Processes. In Robert Tolksdorf Witold Abramowicz, editor, *Proceedings of the 13th International Conference on Business Information Systems (BIS 2010)*, volume 47 of *Lecture Notes in Business Information Processing (LNBIP)*, pages 183–194, Berlin, Germany, 2010. Springer Berlin Heidelberg.

[147] Dennis Wegener and Stefan Rüping. Integration and reuse of data mining in business processes a pattern–based approach. *Int. J. Business Process Integration and Management*, 5(3):218–228, 2011.

[148] Dennis Wegener and Stefan Rüping. On Reusing Data Mining in Business Processes – A Pattern–Based Approach. In Jianwen Su zur Michael Muehlen, editor, *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 264–276. Springer, 2011.

[149] Dennis Wegener, Thierry Sengstag, Stelios Sfakianakis, and Stefan Rüping. Supporting Parallel R Code in Clinical Trials: A Grid-Based Approach. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, Sydney, NSW, Australia, December 10-12, 2008*, pages 823–828, Washington, DC, USA, 2008. IEEE Computer Society.

[150] Dennis Wegener, Thierry Sengstag, Stelios Sfakianakis, Stefan Rüping, and Anthony Assi. GridR: An R-Based Grid-Enabled Tool for Data Analysis in ACGT Clinico-Genomics Trials. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007, 10-13 December 2007, Bangalore, India*, pages 228–235, Washington, DC, USA, 2007. IEEE Computer Society.

[151] Dennis Wegener, Thierry Sengstag, Stelios Sfakianakis, Stefan Rüping, and Anthony Assi. GridR: An R-based tool for scientific data analysis in grid environments. *Future Generation Comp. Syst.*, 25(4):481–488, 2009.

[152] Gabriele Weiler, Mathias Brochhausen, Norbert Graf, Fatima Schera, Alexander Hoppe, and Stephan Kiefer. Ontology based data management systems for post-genomic clinical trials within a European Grid Infrastructure for Cancer Research. *Proc. of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society.*, 2007:6435–6438, 2007.

[153] S A White and D Miers. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008.

[154] Stephen A. White. Introduction to BPMN, IBM, 2004.

[155] Wikipadia. DNA microarray, `http://en.wikipedia.org/wiki/DNA_microarray`, 2012.

[156] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

[157] P. G. Wodehouse. Bioinformatics and Pattern Recognition Come Together. *Journal of Pattern Recognition Research*, 1:37–41, 2006.

[158] Stefan Wrobel, Dietrich Wettschereck, Edgar Sommer, and Werner Emde. Extensibility in data mining systems. In Evangelos Simoudis, Jia W. Han, and Usama Fayyad, editors, *Proc. 2nd International Conference On Knowledge Discovery and Data Mining*, pages 214–219, Menlo Park, CA, USA, August 1996. AAAI Press.

[159] X. Xiang and G.R Madey. Improving the reuse of scientific workflows and their byproducts. In: ICWS, pp. 792-799. IEEE Computer Society, Los Alamitos, 2007.

[160] Hao Yu. Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface), `http://cran.r-project.org/web/packages/Rmpi/Rmpi.pdf`.