



DISSERTATION

Titel der Dissertation

**Scientific Workflows, Data Provenance
Management and Data Anonymization in
Context of the Genome Austria Tissue Bank**

verfasst von

Dipl.-Ing. Konrad Stark

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr.techn.)

Wien, 2013

Studienkennzahl lt. Studienblatt: A 786 881

Dissertationsgebiet lt. Studienblatt: Informatik

Betreut von: O.Univ.-Prof. Dipl.-Ing. Dr. Johann Eder

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die Dissertation selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Wien, 2013

Konrad Stark

Acknowledgements

Prof. Johann Eder - I would like to express you my gratitude for supervising my thesis. Thank you for stimulating my research activities and for all the helpful discussions and suggestions. You helped me with my first steps in the scientific community and assisted me in writing publications and giving conference talks. Thank you for encouraging me to finalize the thesis and for your patience.

Prof. Erich Schikuta - Thank you for supervising my thesis and for all the enriching talks and valuable inputs. You introduced me to other PhD students and found similarities in and bridges between different research areas. I learnt a lot from you concerning critical questioning and focused working, and I appreciate your open, fair and friendly interaction with your colleagues.

Family - I want to dedicate this thesis to my family. Your help, support and beliefs in me made it possible to start an academic career. Thank you for helping me moving around in Austria and for standing by me in strenuous and difficult times. I promise I will now have more time for all of you.

This work was funded by the GATiB project which was a part of the Austrian research program GEN-AU.

Contents

1	Introduction	6
2	Biobanks	13
2.1	History of Biobanks	13
2.2	Biobank Initiatives	20
2.3	GATiB - The Genome Austria Tissue Bank	23
3	Genomics	26
3.1	Central Dogma of Molecular Biology	29
3.2	DNA Replication	30
3.3	Protein Synthesis	31
3.4	Polymerase Chain Reaction	36
3.5	Quantification of Genetic Activity	37
3.5.1	Real-time Polymerase Chain Reaction	39
3.5.2	DNA Microarray	40
4	Support of Cooperative Work in Context of Biomedical Research	43
4.1	Collaboration Scenarios and Main Requirements	44
4.2	Wasabi-CSCW System	49
5	Scientific Workflows and Data Provenance in Context of Medical Research Information Systems	53
5.1	General Aspects of Scientific Workflow Management Systems .	55
5.2	Gene Expression Analysis Workflow	59
5.2.1	General Requirements	61
5.2.2	Data Provenance Related Requirements	64
5.2.3	Data Change Related Requirements	71
5.3	Data Provenance	79

5.3.1	General Aspects and Application Areas of Data Provenance	79
5.3.2	Open Provenance Model	81
5.4	Implementation of the Gene Expression Analysis Workflow . .	92
5.5	Integration of Scientific Workflow Execution in GAMECS-CSCW	109
5.5.1	Workflow Definition Component - Process Editor . . .	113
5.5.2	Process Execution Component	121
5.6	Data Provenance Model	136
5.6.1	Formal Representation	141
5.6.2	Answering General Data Provenance Queries	146
5.6.3	Answering Use Case Driven Data Provenance Queries .	156
5.6.4	Provenance Model - Java Programming Interface . . .	160
5.6.5	Detection and Handling of Data Changes	178
5.7	Established Scientific Workflow Management Systems	194
5.7.1	BioWBI	195
5.7.2	Discovery Net	196
5.7.3	Kepler	198
5.7.4	Kepler/pPod	204
5.7.5	Pegasus	207
5.7.6	Wings for Pegasus	210
5.7.7	Taverna	213
5.7.8	VisTrails	218
5.7.9	Comparison of Scientific Workflow Management Systems	222

6 Anonymization of Sensitive Data 225

6.1	k-Anonymity	225
6.2	Related Work	228
6.3	Generalization Hierarchies	229
6.4	Search Space	233
6.5	Anonymization Algorithm	235

6.6	Performance Issues and Optimizations	236
6.7	Data Repository	237
6.8	Anonymization Wizard	239
7	Conclusion	249
8	List of Figures	251
9	List of Tables	254
10	References	255
A	Abstract	269
B	Zusammenfassung	270
C	Curriculum Vitae	272
D	Installation and Configuration of Open_Anonymizer	273
D.1	Stable Binary Release	273
D.2	Source-Code Release	275

1 Introduction

Medical research is facing considerable changes within recent years. On the one hand advancements in genetic research created novel research branches such as functional genomics or proteomics and provided new insights in the areas of pathogenesis, drug discovery and therapeutic approaches. These new disciplines changed the medical research processes significantly, requiring specialized technologies, laboratory equipment and supporting IT infrastructure. On the other hand, more and more research studies are conducted on the basis of large patient cohorts using both human samples and associated medical data. Therefore, medical research centres invested a lot of effort in collecting and categorizing patient data and specimens. In this context, biobanks were established, gathering and managing medical relevant data and samples. Biobanks are embedded in highly interdisciplinary environments bringing together people with medical, biological and technical background, different expertise, skills and objectives. This thesis was created as part of the Austrian biobank initiative GATiB (Genome Austria Tissue Bank), which succeeded in establishing one of the world's largest collections of diseased and normal human tissue samples. One of the key aspects of GATiB was the design and implementation of an adequate IT system that supports medical research activities in context of a biobank. That is, a multi-purpose research platform for medical scientists was required accessing different biobank data repositories, offering cooperative annotation functionality, integrating various types of analysis software and considering appropriate privacy-protecting techniques for accessing sensitive data.

Medical research is characterized by its highly interdisciplinary environment and rapid progress in the associated scientific areas of genomics, pharmacology, statistics and informatics. Novel technologies in molecular biology (e.g. microarrays) have been developed as well as specialized analysis techniques in context of genetic research. Further, research findings are published and

shared by establishing large public databases (e.g. NCBI) which are storing scientific literature, data of experiments and deduced knowledge (e.g. functions of genes and proteins). Although medical researches increasingly take use of these global resources and benefit from them, they still strongly depend on capabilities of local research infrastructure. Medical databases are required to assess relevant information for planned cohort studies. Biological samples have to be available for extracting DNA for genetic research which in turn necessitates adequate microbiological equipment like microscopes or scanners. Moreover, research activities have to be coordinated, research progress is to be documented and analysis results are shared within the research group. In this context, there is a strong demand for appropriate information systems, that are capable of coordinating research activities as well as providing access to various data sources and incorporating analysis and visualization tools. This thesis presents the design and implementation of an information system for medical research at the Austrian biobank GATiB. The requirements for the information system embrace various aspects, which are summarized in the following.

Cooperative Research Platform

A key aspect of the developed information system is the support and coordination of research activities within a research team and between research groups. Therefore, documents, data and knowledge has to be contextualized for research projects and made accessible for project members. Cooperative work on documents is to be supported as well as annotating relevant items such as collected data, figures and analysis results. Selected documents or items should be made available to related research projects or shared on demand with cooperating research groups. Further, web 2.0 technologies like wikis and tagging should foster the research process and enrich available data with further details. Versioning mechanisms are required to keep track of the

development process of documents and search functions should be applicable to tagged information. Since the research platform is embedded in the IT infrastructure of the biobank, flexible interfaces to existing data repositories are required.

Support of Scientific Workflows

Medical research projects in context of GATiB required an information system for defining, executing and monitoring scientific workflows. A strong focus was on the automation of gene expression analysis processes deployed in tumour marker and drug discovery studies. Medical research projects frequently strive to investigate the development and course of diseases. The results of these studies are required for early diagnosis, disease treatment and drug development. In recent years, the research focus shifted more and more on the impact of genes on diseases. Therefore, standardized procedures have been designed allowing to measure the genetic activities in tissue samples. Genes that are assumed to have an influence on a certain disease pattern may be detected by a so-called gene expression analysis. This technique is a complex process consisting of various data transformation, filtering and calculation steps. Since gene expression analysis is a time-consuming task which involves several semi-automatic steps and the use of external statistical tools, there is a strong demand for automating the process in a scientific workflow. Therefore, the analysis process should be split in computable tasks which are chained in a workflow. The main requirements of the designed scientific workflow management system are summarized below.

- *Comfort and Customization:* A user interface for stepwise executing the analysis workflow is required allowing to design a virtual experiment, import necessary data and parametrize the analysis.
- *Traceability:* For each analysis performed, both the generated results as well as all executed steps are logged. Hence, it is possible to deduce

for each result the sequence of transformation steps, all intermediate results and parameters. In this context, aspects of data provenance are integrated in the analysis workflow.

- *Reproducibility:* As analysis results represent the foundation for further medical research, the entire analysis workflow should be transparent and reproducible. That is, detected statistical dependencies and descriptive data visualizations should be reproduced on demand. By integrating additional data (e.g. clinical parameters) or extending the number of samples in a cohort study, the analysis workflow should be repeatable with slightly modified parameters. The execution of analysis workflows shall be optimized by storing intermediate results of single execution steps.

Data Provenance

Data provenance is used to document the origin of data and tracks all transformation steps that are necessary to reaccess or reproduce the data. Data may be transformed by a sequence of processing steps which could be small operations (SQL joins, aggregations), the result of tools (analysis services) or the product of a human annotation. Thus, these transformations form a construction plan of a data object, which could be reasonably used in various contexts. Traceable transformation processes are useful for documentation purposes, for instance, for the materials and methods section of publications. Generally, the data quality may be improved due to the transparency of data access and transformation. Data provenance is an important aspect in the field of scientific workflow management systems. Scientific workflows consume input data, and generate intermediate data and output data. By precisely recording, which data has been produced by which execution task of a workflow, the construction plan of each data object becomes transparent. Data provenance queries may be formulated based on the collected data allowing to answer questions like 'Which tasks were executed with which input data in order to produce data object X'? In context of GATiB, data

provenance recording was used for documenting and validating analysis results. For instance, a gene is assumed to be involved in the pathogenesis of a certain disease. Scientists might ask for detailed analysis processes, in which the gene was involved and all related input and intermediate data of the processes. An elaborated provenance model was created that is capable of tracking the execution details of scientific workflows and fulfilling the requirement of medical research projects. Moreover, the provenance model covers both process-oriented data of workflow execution as well as collaborative data resulting from contextualizations and annotations of the CSCW system.

Data Anonymization

When releasing patient-specific data (e.g. in medical research cooperations) privacy protection has to be guaranteed for ethical and legal reasons. Even when immediately identifying attributes like name, address or day of birth are eliminated, other attributes (quasi-identifying attributes) may be used to link the released data with external data to re-identify individuals. In recent research much effort has been put on privacy preserving and anonymization methods. In this context, k-anonymity was introduced allowing to protect sensitive data by generating a sufficient number of k data twins. These data twins prevent that sensitive data is linkable to individuals. In order to facilitate exchange of patient-related data in medical research projects, a data anonymization tool is required which accesses sensitive data and transforms it into k-anonymous data sets. Anonymization of sensitive data is accomplished by transforming specific values into more general values. For instance, the day of birth of an individual is a strong characteristic that may allow to identify the individual within a group. In order to prevent the individual from being identified, the day of birth may be transformed into the age in years, which may be equal for several individuals. However, as a consequence of data generalization, information is lost and the transformed data may be useless for further processing. Therefore, an anonymization technique

is required ensuring that the anonymized data is still usable for subsequent analysis. Data quality standards must be definable for each data attribute, forcing the anonymization technique to generate appropriate anonymization solutions. An open source anonymization software, Open Anonymizer, was implemented in Java. Open Anonymizer is a highly customizable anonymization tool providing the best anonymization for a certain context. That is, the anonymization process is strongly influenced by data quality requirements of users. Users may specify the importance of attributes as well as transformation limits for attributes. These parameters are considered in the anonymization process, which delivers a solution that is guaranteed to fulfil the user requirements and has a minimal information loss. Open Anonymizer provides a wizard-based, intuitive user interface which guides the user through the anonymization process. Instead of anonymizing the entire data set of a data repository, a simple query interface allows to extract relevant subsets of data to be anonymized.

Contributions

Parts of the thesis have already been published. The GATiB-CSCW system was presented at the CAiSE'08 [97] and the ICEIS 2008 [88]. The basic concept of the anonymization algorithm was depicted in a publication presented at the DaWaK 2006 [94]. An extended version of the algorithm that is capable of anonymizing distributed data sources and is based on a hill-climbing search strategy was published afterwards [95]. This thesis contains an adapted version of the anonymization algorithm based on a best-first search strategy. The first version of the semantic data provenance model was presented at the DEXA 2010 [96]. Though, the extended provenance model, depicted in this thesis, allows to handle collections of data objects, complies with the Open Provenance Model and is capable of dealing with data changes.

This thesis is organized as follows: in Section 2, the main aspects of biobanks are presented. Within this section, the reasons for establishing biobanks in various research centers are highlighted. Biobanks have been constructed for different research scopes and collected various types of biological material and associated data. Further, an overview of leading biobank initiatives in Europe is given, which are compared to each other by organization, size, type of biological resources and collected data. Finally, the Austrian biobank GATiB is presented in more detail. Section 3 gives an introduction in the disciplines of genetics and bioinformatics, two research areas that have made strides in recent years and expedite medical research substantially. The basic functionality of genes, their encoding of information and their role in controlling biological processes are outlined. Monitoring genetic activity is a promising approach for investigating predispositions for diseases and courses of diseases. Some diseases may be linked to defective genes, others may be caused by overexpressed genes. Genetic techniques strive to find out the behaviour of genes in organisms and their impact on biological and cellular processes. Bioinformatics is a novel research discipline which is complementary to genetics. It supports the assessment and storing of data created by genetic experiments and has an important role in analyzing and documenting the results of experiments. In Section 4, the medical CSCW system for supporting the collaboration in research projects is presented. The CSCW system is based on a modern service-oriented architecture and integrates various communication channels and Web 2.0 technologies. The main part of the thesis, management of scientific workflows and provenance management, is presented in Section 5. Starting with an introduction in the area of scientific workflow, the workflow- and data provenance-related requirements of the GATiB are depicted. A user-interactive web application has been realized for manually executing the 'Gene Expression Analysis' workflow. Further, a distributed IT system for defining scientific workflows and batch-executing them was implemented. The IT system was realized as an extension of the

CSCW system. The formal specification of the novel data provenance model is given and the capabilities of the model are demonstrated by answering data provenance queries. The anonymization of sensitive patient data and the functionality of the anonymization tool Open Anonymizer are illustrated in Section 6.

2 Biobanks

2.1 History of Biobanks

In recent years, large collections of biological materials have gained importance for medical research. Many medical studies and research projects are based on rather small collections of specimen of patients suffering from a specific disease. While these studies investigate the reasons for a certain disease, and its course and treatment, the collected data and material is not reused in other medical research contexts or made available for follow-up studies. [100]

Generally, the necessity of biological resource centres may be justified by the weaknesses and inadequacy of current repositories of biological resources which do not fulfil the requirements of medical research and drug discovery. Current repositories commonly lack guaranteed quality of samples and standardized processing methods, which are essential indicators for reliable and reproducible research results. Further, sample repositories that are geographically and organizationally dispersed have difficulties in sharing resources and research results among each other. As many medical studies depend on large cohorts of specimens of equal disease patterns, there is a strong need to establish cooperations among research institutions and to share material and associated data. For instance, research on rare cancer types requires collecting of adequate cases over a long period of time and/or the cooperation of several research groups. Recent advances in genomics have risen the consciousness for establishing large repositories of human specimen

collections. In genomics, the set of genes that influences the outcome of a certain disease is to be detected. In simple terms, genes are said to be overexpressed, neutral or underexpressed in a single cell, sample or organism, whereas the expression of a gene measures the degree of its activity. Since a set of genes may be responsible for a certain cellular component (e.g. protein), molecular function or biological process and even interact with other set of genes in certain conditions, the task of identifying the gene group and prove its influence is crucial. Khoury [46] illustrates in a simple calculation example the complexity of detecting the gene set responsible for a disease outcome. Let's assume that 10 genes are considered of having an influence on a common disease, and each gene may be highly active (overexpressed) or inactive (neutral or underexpressed), then there exist 1024 ($= 2^{10}$) variants of activity of this gene group. That is, in a cohort study the set of patients may be classified in up to 1024 subsets, and each subset must have a sufficient number of elements to achieve significant statistical results. If 20 genes are assumed to contribute to a disease, even more than one million subsets may be created. Khoury [46] emphasizes the importance of collecting large biological materials in order to carry out relevant epidemiological studies. The Organization for Economic Co-operation and Development (OECD) pointed out the important roles that are played by biological resource centres:

1. *Preservation and provision of biological resources for scientific, industrial, agricultural, environmental and medical R&D and applications*
2. *Performance of R&D on these biological resources*
3. *Conservation of biodiversity*
4. *Repositories of Biological Reference Material*
5. *Repositories of biological resources for protection of intellectual property*
6. *Resources for public information and policy formulation* [29]

Various types of biobanks have been developed addressing different kinds of research questions. Generally, we may distinguish between population-based and disease-oriented biobanks. In a population-based biobank, biological samples and data are taken from donors that are randomly selected from a general population. The Council of Europe defines a population-based biobank as a collection of biological materials that has the following characteristics:

1. *The collection has a population basis.*
2. *It is established, or has been converted, to supply biological materials or data derived therefrom for multiple future research projects.*
3. *It contains biological materials and associated personal data, which may include or be linked to genealogical, medical and lifestyle data and which may be regularly updated.*
4. *It receives and supplies materials in an organized manner [66].*

Disease-oriented biobanks are collections of samples that have been taken in context of a certain medical diagnose or treatment. A large number of collected samples allows to track the course of a disease in various stages. Biomarkers may be identified that are applicable for early diagnoses, for predicting the disease progression and for estimating the success of a specific treatment or therapy. Biobanks can be further subcategorized as depicted in Figure 1.

Population-based biobanks that are composed of samples collected in a concrete region or taken from a certain ethnic cohort are referred to as population isolated biobanks. Population cohorts that are geographically or ethnically isolated are proved to be an precious resource for medical research, since there is less genetic variation in such groups than in population with a stronger ethnically heterogeneity. A longitudinal population biobank is a long-term research project monitoring a general population over a large period of

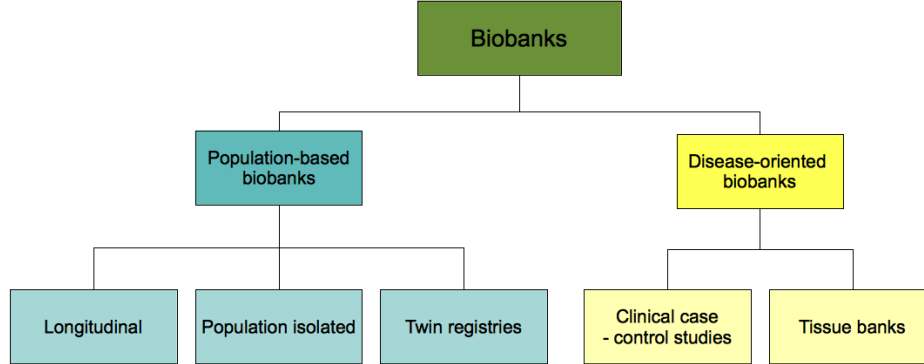


Figure 1: Biobank Categories

time [7]. Longitudinal biobanks initiatives allow to observe courses of diseases from a historical perspective. That is, individual susceptibility to diseases may change over time due to various environmental factors, life style habits or measures of preventive medicine. Moreover, advances in early diagnoses, drug discovery and new medical treatment techniques may alter courses of disease significantly. Long-term observations of population cohorts allows to draw conclusions from changed medical techniques about effects on diseases such as success of treatment or mortality. Twin registries are biobanks based on samples and data from twin cohorts. These registries are a valuable sources for medical research. Twin pairs are at the same age and are typically exposed to equal environmental factors like childhood conditions, making them ideal candidates for studying chronic disorders. Monozygotic twin pairs share the same genotype and are therefore ideal for investigating genetic and environmental factors affecting disease development [73]. Case-control studies are subtypes of disease-oriented biobanks that are composed of samples and data of both diseased and healthy individuals. Medical studies strive to identify so-called disease profiles by comparing patients with healthy persons. That is, genetic predispositions for a disease as well as ge-

netic activities influencing the stages of a disease are to be determined. A more general form of disease-oriented biobanks are tissue banks, which are built on various diseased human tissue samples together with associated diagnostic data. These samples are usually taken as part of a histopathological diagnosis and are stored in archives of pathologies and hospitals [7]. While case-studies are typically based on small cohorts of several hundred or thousand individuals, tissue banks contain much larger collections of samples that are constantly growing due to the daily routine work at a hospital. Moreover, case studies assess very detailed disease-specific data depending on the research question addressed in the study. The collected data is characterized by a good data quality accomplished by categorizing disease- and life-style specific parameters, making them applicable for statistical analysis. By contrast, the collected data in tissue banks is less detailed and less structured. Although diagnoses of diseased tissues have been made, the assessment of additional medical parameters like family anamnesis or life style data require corresponding personal and technical resources. That is, for large sample collection, the data assessment and sample annotation process is complex and time consuming. In cases of historical tissue archives, diagnostic findings may be available in unstructured text documents improper for further analysis. Therefore, diagnoses have to be transformed in standardized categories and medical parameters have to be extracted.

A key aspect in modern medical research is the early detection of diseases allowing effective interventions and therapies in order to reduce the mortality and morbidity [93]. Diseases are complex processes consisting of various stages that are reflected in biological processes and at the molecular level. In this context, biomarkers are being developed assessing the risk for a disease, its state or its future development. A biomarker may be defined as: “Biomarkers are measurable internal indicators of changes in organisms at the molecular or cellular level—offer great potential to understand environmentally mediated disease and to improve the process of risk assessment.

A valid biomarker could also be considered a key event linking a specific environmental exposure to a health outcome” [9]. One of the earliest known biomarker is the measurement of glucose in blood samples for diabetes diagnosis. In recent years, the research focus of biomarkers shifted to genetic influences on diseases. Since genes are involved in biological processes like cell grow or division, they have an essential role in the development process of diseases. Biomarkers at the genetic level are molecular signposts made up of active genes in a cell and their associated protein products and other chemicals [93]. A biomarker is defined to be a valid biomarker, if it has been proven to be an indicator for the presence of some disease state. For instance, the HER-2 gene could be identified as an important biomarker that is related to the proliferation of breast cancer cells. Beside the diagnostic use of this biomarker, the HER-2 gene was used as a therapeutic target. By the use of special antibodies, the activity of the gene was limited, thereby reducing the proliferation of breast cancer cells [81]. Thus, biomarkers are a valuable source for analyzing the etiology of a disease, its onset and course, and may provide significant therapeutic and diagnostic insights.

Biobanks may be further characterized by the research purposes that may be supported by them. Figure 2 illustrates various types of biomarker investigation in context of cancer research and relates them with different kinds of biobanks [83]. In the left side of the figure, different stages of the disease process of cancer are depicted, reaching from genetic susceptibility to the onset of the disease. During the disease process, different types of biomarkers may be used to assess the state of the disease.

The genetic susceptibility of a person for a disease is determined by the genotype. That is, the person has a higher risk of developing a disease because of his genetic predisposition. Biomarkers of susceptibility are used to assess the individual risk for the disease.

Biomarkers of exposure are indicators relating external exposures to internal dose [9]. For instance, air pollution and tobacco smoke are external

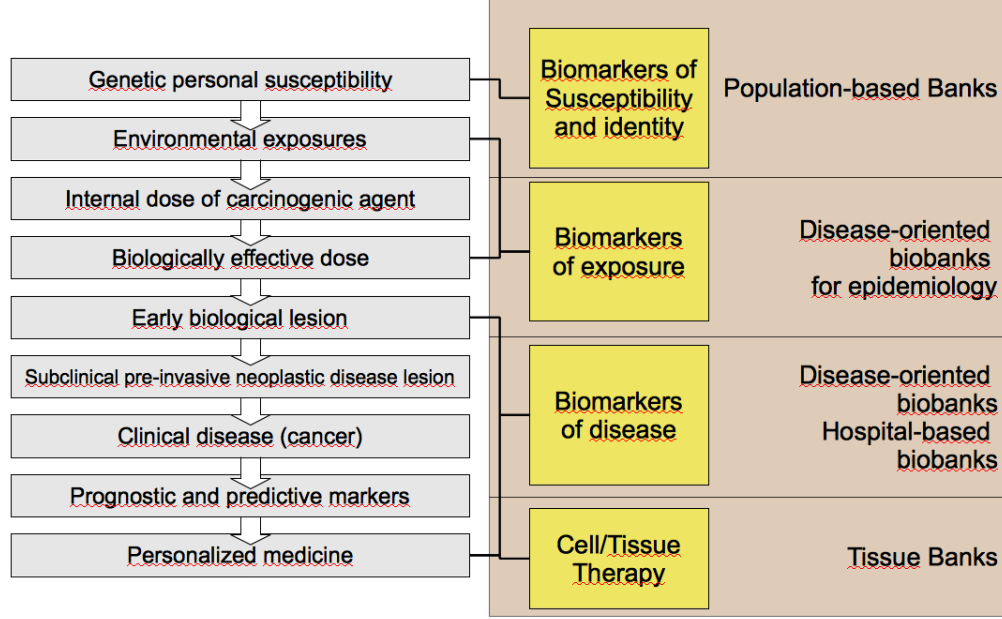


Figure 2: Research Purposes of Different Kinds of Biobanks [83]

factors increasing lung cancer risk. Though, the measurements in this stage of disease do not indicate the actual onset of the disease.

Biomarkers of disease indicate that both exposure and adverse effects of the disease have already occurred. Early biological lesions are detectable and clinical diseases are assessable. A prognostic biomarker allows to give information about the overall outcome of a disease. For instance, the number of tumor metastases may determine the malignancy of a tumor. Predictive biomarker are used to estimate the therapeutic effects of a therapy. For instance, the above-mentioned HER-2 gene is a target for therapy in breast cancer cases.

Cell therapies are treatments introducing new cells into damaged organs or tissue. The most notable advances in this research area have been accomplished with human stem cells. Stem cells have a high potential for therapeutic application because of their unique regeneration and transfor-

mation abilities. Stem cells are unspecialized cells that may be induced to transform themselves into specific cell types such as blood, muscle or any organ cell. These cells are also found in blastocyst embryos and build up the entire body of an organism. Moreover, in adult organisms stem cells have the natural function of replacing damaged or destroyed tissues or cells. For instance, bone marrow stromal cells have the ability to differentiate into bone and fat cells and may support the creation of new blood cells. Stem cell may reproduce themselves by cell division making them a valuable source for in vitro cell cultures. The most promising areas of cell therapy encompass the replacement of damaged cells to treat diseases including diabetes, heart disease or spinal cord injuries. Further, stem cell cultures may be used in drug development testing effects and efficiency of novel drugs [67].

2.2 Biobank Initiatives

The need for large repositories of human material was recognized in several biobank initiatives, especially in so-called *population-based* biobanks. The Icelandic Health Database project is one of the most publicly perceived population-based biobank projects, due to its large-scale collection, its comparatively long history and the public criticism, it was confronted with. The initiative started in 1998, when the Icelandic Ministry of Health presented the plans of constructing a Health Database covering the whole Icelandic population. These plans focused on combining medical records of patients with both genetic data and genealogical information [84]. The main idea was to identify the genetic influence on diseases under the consideration of patient anamnesis data and medical and genetic data of relatives. The Icelandic population is proved to be an excellent population for studies of the genetics of common diseases, since it is ethnically separated from other populations and has a homogeneous gene pool [39]. By contrast, genetic influences on diseases are harder to detect in a population with more variations in its genotype. False-positive associations for disease patterns may be detected,

while true associations may be overlooked [41]. Moreover, the well documented genealogy of the Icelandic population permits detailed tracking of relative relationships, which eases research on genetic heritability. The realization of the Health Database was performed by the company deCODE Genetics, which was granted an exclusive license for constructing and using the database for a period of 12 years. This model of public-private cooperation attracted the public’s attention and become a subject of discussion and criticisms. The combination of highly sensitive medical and genetic data privacy haven risen concerns regarding data privacy aspects and potential abuses. The database containing genealogy data comprises records of more than 600,000 Icelanders [58], whereas the genetic database was built on the basis of blood samples from more than 140,000 volunteer participants.

The idea of the UK Biobank initiative was to establish a long-term collection process of blood, blood fraction and urine samples. Over a period of 30 years, approximately 500,000 volunteers aged between 40 and 69 years will continuously donor samples. The volunteers participate in a longitudinal health observation study in which the results of basic medical measurements (e.g. body mass index or blood pressure), life style data (e.g. smoking and drinking habits, physical activity) and occurred diseases are repeatedly assessed. Furthermore, genetic data may be extracted by analyzing the human samples, whereas the large number of samples facilitates research on a great variety of diseases. Since the collection comprises medical and genetic data of healthy and ill people, it is a valuable resource for investigating health- and disease-related influences and determinants [70]. Medical research is often based on *prospective* and *retrospective* studies. While retrospective studies collect samples and data of patients after they have been diagnosed with a certain disease, prospective studies gather materials in advance. Retrospective studies may focus on specific disease types and may be realized with less effort than prospective studies. Though, medical studies require materials of non-diseased control groups in order to detect disease-related factors

in genetic and biological processes. Therefore, control samples have to be included in retrospective studies. The advantage of prospective studies is that specimens are taken before diseases have been developed. This allows to research on disease-related parameters of biological samples and to identify *biomarkers* which are genetic patterns associated with susceptibilities to certain diseases. Both the Icelandic Health Database and the UK Biobank are prospective collections of samples. The Estonian Genome project is located at the Estonian Genome Centre of the University of Tartu. The aim of the initiative is to collect both phenotype and genotype data from a representative cohort of the Estonian population [59]. Blood samples are taken from volunteer donors and genetic data is extracted. The volunteers are asked to fill out a detailed questionnaire assessing personal disease history data, family anamnesis data and life habits data. The ambitious objective of the project was to collect material and data from approximately 5 percent of the entire Estonian population (1,340,000 citizens). Though, towards the end of 2010 the number of donors contributing to the project reached 47,000.

While most of the population and disease-based biobanks were founded as regional or national projects, some initiatives are aimed at interconnecting research institutes internationally and sharing biological resources, medical data and expertise. One example is the GenomEUtwin project, which was created in order to achieve synergy effects in research on European twin and population cohorts [99]. GenomEUtwin established a distributed research network consisting of eight leading European research centres. On the one hand, the projects focused on twin cohort studies of the research centres. Since monozygotic twins share the same genotype, twin cohorts represent a unique research opportunity for studying genetic traits as well as the influence of environmental factors. GenomEUtwin comprises medical data from over 600,000 pairs of twins, of which 30,000 have donated DNA samples. On the other hand, it integrates resources from the MORGAM project, which is a multinational medical research project based on distributed cohorts as well.

The MORGAM project investigated risk factors of cardiovascular diseases. For this purpose, over 140,000 randomly selected participants have been recruited in 11 countries [73]. Disease history, family anamnesis and living habits have been assessed as well as follow-up data. Further, DNA samples from 69,000 participants have been taken. Since the subjects of the cohorts were selected randomly, a subset of individuals have been confronted with occurrences of cardiovascular diseases, while the remaining ones may be used as control groups. One of the largest population-based biobanks has been established at the Norwegian Institute of Public Health. The biobank platform Biohealth was funded by Norwegian functional genomics program (Fuge) and was built on the large mother and child cohort study MoBa which comprises 270,000 individuals, the CONOR cohort (180,000 individuals) and various smaller cohorts. The total number of subjects contributing to the biobank exceeds 408,000, of which 230,000 have donated DNA samples, making up one of the largest biobanks in the world. Besides the biological samples, genealogical data about the donors was integrated from the Norwegian Central Person Registry. Further, all cases of cancer in the Norwegian population since 1951 were recorded in a central cancer registry. Thus, detailed diagnosis and treatment data was documented as well as various causes of death.

2.3 GATiB - The Genome Austria Tissue Bank

The Institute of Pathology (IoP) of the Medical University Graz (MUG) has a long and meaningful history as a pathological service provider. The origins of the institute date back to Austrian-Hungarian monarchy in which the institute was almost the unique pathological diagnostic centre for the University Hospital in Graz and another 43 hospitals. Due to the lack of other pathological institutes in the region, the IoP of Graz was able to collect tissue samples from a non-selected Middle European population. That is, the collected material represents various human diseases at their natural frequency of occurrence. Since the human samples were taken in a clinical

context, diagnostic medical data is available as well as follow-up data such as surgical data, treatment and survival data. The high potential of the collection for medical research was recognized and much efforts have been made to transform the medical archive into a national biobank which is called GATiB (Genome Austria Tissue Bank). The funding for the initiative was provided by the Austrian Genome Program (GEN-AU).

The main idea of GATiB was to change the well-established population-based biobank into a disease-focused biobank allowing research on various cancers such as colon, breast, liver, as well as on metabolic liver diseases and on organs affected by the metabolic syndrome. Prospectively collected tissues are associated with blood samples and detailed data on the sample donor's disease, lifestyle and environmental exposure, following standard operating procedures [6].

Currently, approximately 2,900,000 biological samples from 888,000 patients have been taken and are preserved in the tissue archive of the IoP. Most of the samples are paraffin-embedded tissue samples, while a smaller collection consists of cryopreserved tissue samples. Both types of samples may be used for extracting RNA/DNA and proteins and may be utilized for genetic research.

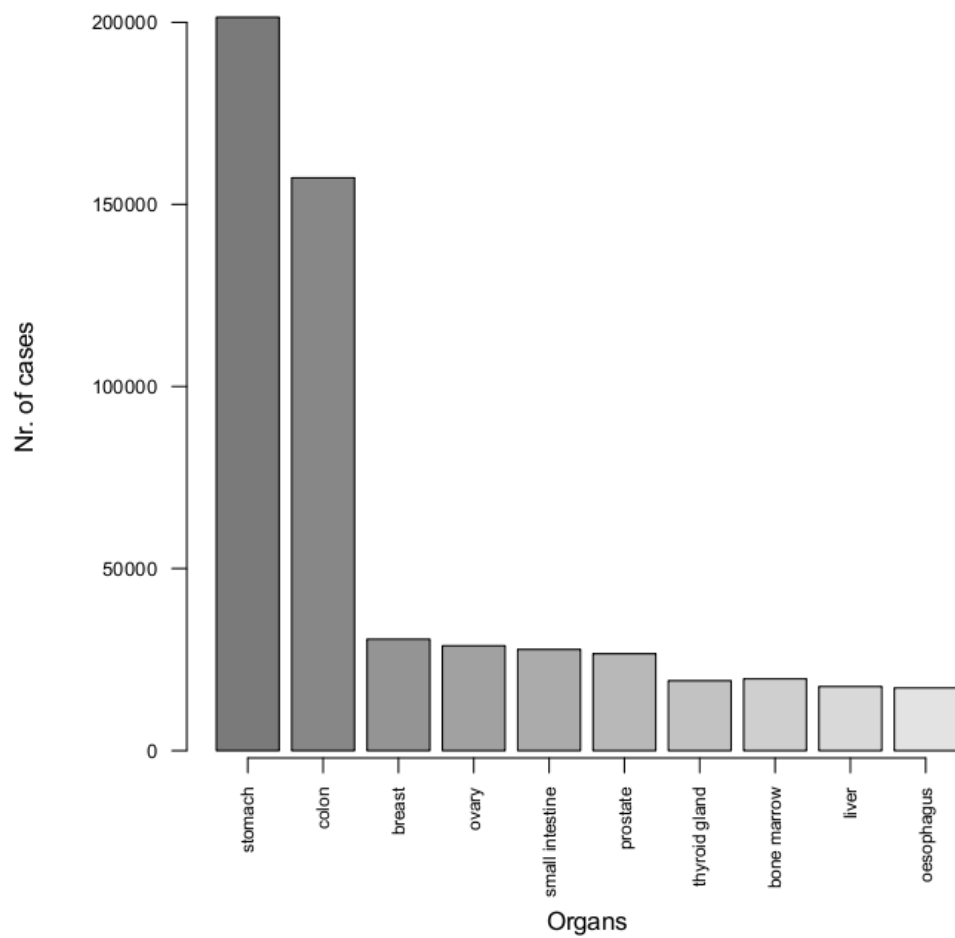


Figure 3: GATiB - Biological Samples Stratified by Donor Organs

Name	Type	sample type	samples	medical data	genetic data	geneal. data	life style data
Icelandic Health Database	population-based	blood	600,000	yes	yes	yes	no
UK Biobank	population-based	blood	500,000	yes	yes	no	yes
Estonian Genome project	population-based	blood	47,000	yes	yes	no	yes
GenomEUtwin	twin registry	blood	600,000	yes	yes	no	no
MORGAM project	disease-oriented	blood	140,000	yes	yes	no	yes
Biohealth	population-based	blood	408,000	yes	yes	yes	yes
GATiB	population-based	paraffin, cryo	2,900,000	yes	yes	no	yes

Table 1: Overview of Biobank Initiatives

In Figure 3 the distribution of biological samples over donor organs is illustrated [10]. The majority of collected material constitutes of samples taken from stomach and colon samples. Though, large collections of samples have been taken from breast, ovary, liver and prostate as well which reflect the expertise and research focus at the MUG. The annual growth of formalin-fixed paraffin-embedded (FFPE) tissue is around 120,000 while approximately 1,000 cryopreserved tissue can be taken from donors every year [6].

A comparison of established biobanks initiatives is presented in Table 1. The categories of biobanks are listed as well as the types and numbers of collected samples. Further, types of associated sample data are presented.

3 Genomics

Each organism is characterized by its genetic constitution that is responsible for building and maintaining a living example of the organism. The hereditary disposition of an organism is called genome and is encoded in the deoxyribonucleic acid (DNA). DNA is typically located in the cell nucleus, though some DNA may be found in mitochondria. DNA is composed of four types of chemical bases: adenine (A), thymine (T), cytosine (C) and guanine (G). Each of the bases has a complementary base, which is a base, that may be chemically bound to it. The complementary base of adenine is

thymine and guanine is the complementary base of cytosine. DNA is made up of a sequence of pairs of complementary bases, whereas each base has an attached sugar (deoxyribose) and a phosphate molecule. DNA has the three-dimensional shape of a double helix which is formed by two strands that are connected by complementary base pairs. The strands are made up of the sugar and phosphate molecules and form the backbone of the DNA. The slightly twisted form of DNA results from the chemical binding characteristics of the bases [28]. The three components base, sugar and phosphate are summarized under the term nucleotide. Another molecule, the ribonucleic acid (RNA) has a similar structure to the DNA. It is composed of a sequence of nucleotides, which form a single strand molecule. Though, RNA nucleotides are based on ribose sugar molecules and use the base uracil instead of thymine. RNA acts as a catalyst in various biological processes, for instance the protein synthesis.

When looking in detail at the bindings between nucleotides, different kinds of bindings may be noticed. On the one hand, bases of nucleotides may be bound to other bases by hydrogen bondings. These bondings are very weak in cases when a hydrogen atom is shared by atoms like oxygen or nitrogen. Thus, the hydrogen bondings may be broken when needed. This characteristic is utilized in the DNA replication process, in which the two DNA strands are separated. While two hydrogen bondings exist between adenine and thymine, there are three bondings between cytosine and guanine. On the other hand, nucleotides are linked to each other by connecting the phosphate of a nucleotide with the sugar molecule of an other nucleotide, which is called phosphodiester bond. The sugar molecule consists of five carbon atoms, of which two (position 3' and 5') are capable of binding to phosphate. In Figure 4, both kinds of bindings are illustrated. The base pair thymine-adenine is connected by two hydrogen bonds, while the two pairs cytosine and guanine are each connected by three hydrogen bonds. In the left strand, the first phosphate is bound to the position 5' of the sugar molecule,

while the last sugar molecule in this side is bound to a hydroxyl group (OH) on the position 3'. On the contrary, the right strand starts with a hydroxyl group at the first sugar molecule and ends with a phosphate on the last. The strands are said to have a different polarity and are antiparallel. This kind of polarity is necessary for enabling the hydrogen bondings [98].

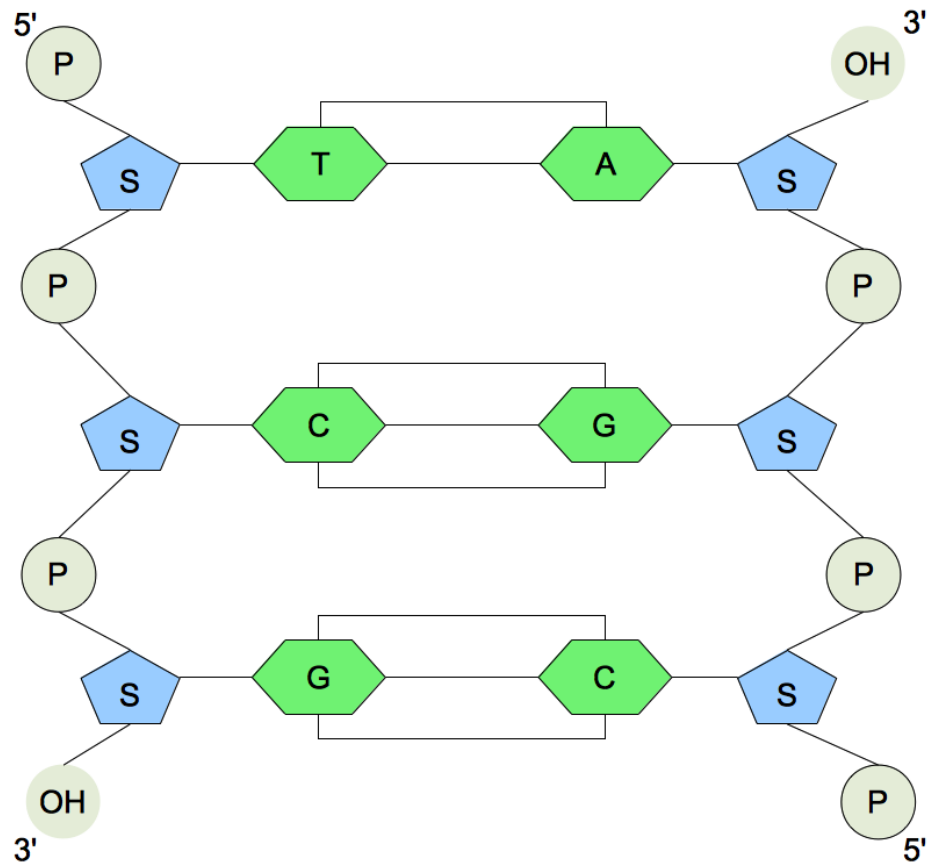


Figure 4: Nucleotides

3.1 Central Dogma of Molecular Biology

Human DNA is composed of approximately three billion bases. However, 99 percent of the entire human DNA is identical in all humans [68]. DNA sections may be classified into coding and non-coding sequences. A sequence of DNA that codes for a protein or a RNA is defined as a gene. While early research findings estimated that there exist 35,000 human protein coding genes, the estimated number was reduced to 20,000-25,000 in recent years [77]. More than 98 percent of human genome consists of non-coding sequences [26]. DNA is coiled around histones which are protein structures forming chromosomes. Human DNA is structured in 23 chromosome pairs where 22 chromosome pairs are diploid or autosome, which means that they carry the identical genetic code. The remaining pair is called sex chromosomes determining the sex of a human.

DNA may be considered as a carrier of genetic information. It encodes regulatory mechanisms controlling biological processes. In simple words, the genetic information determines a set of execution steps that have to be executed in a specific order for achieving a certain result. However, in order to operate biological processes, the interaction of several cellular components is required. Some have the task of reading the genetic information, some are responsible for transporting the information and others finally interpret the information and perform the necessary actions. The basic genetics mechanisms have been recognized and described by Francis Crick in the central dogma of molecular biology [20]. The dogma illustrates information flows and associated actions between DNA, RNA and proteins. The encoded genetic information is passed from DNA to a replicated DNA during cell division. Moreover, in the transcription process, a part of genetic information, a certain sequence of DNA, is read and a new RNA molecule is creating which is a copy of the read DNA sequence. The information encoded in RNA molecules is then used to synthesize proteins. That is, the RNA sequence is translated into a sequence of amino acids which are building components of proteins.

Additional information flows exist, which occur less regularly. For instance, viruses are capable of replicating RNA. Further, the RNA may act as a template for DNA synthesis in so called reverse transcriptions. The HIV virus may synthesize a DNA strand based on its own base sequence. The process of translating DNA immediately into proteins could only be induced under laboratory conditions [98].

3.2 DNA Replication

Generally, DNA fulfils two major functions: DNA replication and protein synthesis. DNA replication occurs in the process of cell growth, which is initiated by a cell division splitting a single cell into two daughter cells. There are several types of cell divisions depending on the type of the cell that is divided. Eukaryotic cells have a nucleus that is bound by a membrane. Within the nucleus, a set of paired chromosomes carry the genetic information. Animals, plants, fungi and many microorganisms possess eukaryotic cells. In prokaryotic cells, DNA is carried in a single circular chromosome. Prokaryotic cells may be found in bacteria. In both cell types, cell division is started with a replication of DNA. In the first step of replication, the DNA double helix is untwisted and the two strands are separated on certain regions of DNA, called the DNA origins. Each of the strands acts as a template for the newly generated DNA replicas. Complementary strands are synthesized for both single strands by stepwise adding free nucleotides according to their binding properties.

The replication process requires a variety of proteins. First, the protein helicase splits the DNA in two strands while a set of ssb proteins are attached to the single strands preventing them from reannealing. Further, short sequences of DNA are synthesized for creating RNA primers, which are starting segments for adding complementary nucleotides. The generation of RNA primers is executed by the primase protein. Finally, the polymerase enzyme ligates free nucleotides to the template strand starting at the RNA primer.

Since polymerase enzymes may only bind nucleotides to DNA in the 3'-to-5' direction, the replication process is different on each DNA strand. The DNA that is unwound in the 3'-to-5' direction is depicted a leading strand. After a primer has been set on the strand the polymerase enzyme may continuously add complementary nucleotides. The other strand, unwound in the 5'-to-3' direction, is called the lagging strand. On this strand, nucleotides are added discontinuously. A primer is synthesized to the strand, and the polymerase enzyme binds nucleotides on a short DNA sequence in the opposite direction. After the sequence has been complemented, a new primer must be set and the polymerase enzyme resumes its work. Thus, the leading strand requires just a single primer, while on the lagging strand multiple primers have to be synthesized resulting in disruptions of the replication process.

3.3 Protein Synthesis

Proteins are the essential compounds of any organism. On the one hand, proteins form the structure of cells and organisms. On the other hand, they support various chemical reactions in the role of enzymes, and transduce signals between cells.

Proteins are synthesized by cells whereas the type of cell determines the type of protein that is produced. The information about what kind of protein is to be synthesized in which quantity is encoded in the genetic information of the cell [98]. A protein is composed of a chain of amino acids, called a polypeptide structure, in which amino acids are tied together by peptide bonds. 21 different amino acids exist in the human organism.

Protein synthesis is a multi-level process which involves several components of a cell. In simple words, the construction plans of all proteins are encoded in DNA. If a specific protein is to be synthesized, a certain segment of DNA is read and the protein building information is extracted. Reading out the protein blueprint information is referred to as transcription process occurring in the cell nucleus. The protein blueprints are transported to the

cytoplasm, in which the ribosomes reside. Ribosomes are the protein building factories of a cell. They construct proteins by stepwise adding amino acids to polypeptides. The decoding of protein blueprints in ribosomes is referred to as translation process. In the following, the protein synthesis steps for eukaryotes are presented in more detail.

During transcription, a sequence of DNA is read and a RNA molecule carrying the coding information is built. The RNA molecule is called messenger RNA (mRNA) and is an exact replica of a sequence of one DNA strand. Hence, one of the two DNA strands is used as the source of the transcription, and is called coding strand. The other strand, the non-coding strand, is used as a template for creating a copy of the coding strand. While the coding strand runs from the 5'-to-3' direction, the template strand runs in the opposite direction. Besides mRNA, other RNA molecules may be synthesized by transcription, each having a particular function. The ribosomal acid, rRNA, resides in ribosomes and supports the translation process from mRNA into proteins. Further, transfer RNA (tRNA) is another RNA involved in protein synthesis. It is responsible for binding single amino acids for transporting them to the ribosomes. Each RNA molecule is synthesized by a certain protein enzyme. RNA polymerase I transcribes rRNA, RNA polymerase II transcribes mRNA and RNA polymerase III transcribes tRNA [48].

The sequence relevant for transcription is considered a gene and may be subdivided into following functional regions: exons, introns, promoters and enhancers. While promoters and enhancers regulate the transcription process, exons and introns are segments that are transcribed into the RNA molecule. Exons are code segments that are used in the protein construction process, while introns are assumed to not carry any coding information.

In the first step of the process, the relevant coding segment in the sequence of nucleic acid must be detected. Short DNA sequences, called promoters, mark the beginning of a coding segment. In eukaryotes, the most familiar promoter is the TATA box, which has a leading base sequence TATAAA.

Promoter regions serve as connectors for transcription factors which are proteins controlling the transcription process. The coding sequence, following the promoter region, is replicated by the enzyme RNA polymerase II. Before RNA polymerase initiates the replication, it must be bound to the promoter. Though, polymerase recognizes promoter regions only if separate transcription factors are found. Thus, transcription factors have a significant role in protein synthesis. Further, the multiprotein complex mediator enables signal transmission between transcription factors and RNA polymerase [45].

Similar to DNA replication process, the DNA double helix structure must be untwisted and separated in its two strands. The enzyme helicase splits the two strands starting at the promoter region. RNA polymerase starts at the end of the promoter region and moves along the template strand. RNA polymerase synthesizes a new RNA molecule by reading the nucleotide sequence of the template strand and binding free complementary nucleotides. Thus, an exact copy of the coding strand is created, except for base thymine which is replaced by uracil. Transcription may be repeated on the same DNA segment creating multiple RNA molecules. In this case, transcription is executed in several overlapping rounds. After reading 50 to 60 base pairs, the distance between RNA polymerase and the promoter region allows another RNA polymerase to be bonded to the promoter region initiating a new transcription round. The simultaneous production of RNA molecules is essential, since RNA has a relative short lifetime and is decayed after a certain period of time. Transcription of DNA may be stopped in two different ways, either by transcription termination sequences in DNA (self-termination), or by both a termination sequence and a termination protein. The termination sequence is rich of guanine and cytosine bases followed by a sequence of adenine bases. When the termination sequence is read and translated, the complementary binding characteristics of guanine and cytosine force the constructed RNA to be bound to itself forming a double-strand structure, also called a hairpin structure. RNA polymerase terminates the transcrip-

tion process and the RNA molecule is dissociated from the DNA template. Alternatively, in prokaryotes, termination of transcription may be supported by the Rho protein which moves along the RNA molecule and separates it from DNA [40]. In eukaryotes, after termination of transcription, additional methyl guanoses (5' cap) are added at the 5' end. The most important function of the 5' cap is to promote the translation process in ribosomes. At the 3' end, the enzyme endonuclease removes the non-protein-coding tail of the sequence. Further, a long sequence of adenine bases is appended to 3' end of RNA (polyadenylation). The end product of transcription is a RNA molecule consisting of exons and introns of the gene. The non-coding introns are removed from the RNA strand and the remaining exons are tied together [8]. The resulting molecule is called mRNA and is transported from the nucleus to the cytoplasm, where the ribosomes reside.

In the ribosomes, the decoding from mRNA bases into protein building plans takes place. Since proteins are composed of sequences of amino acids, mRNA encodes the exact order of amino acids that builds up a protein. Triplets of bases, also called codons, are used to encode single amino acids. For instance, the codon UUU encodes the amino acid phenylalanine. Having four encoding bases A,U,C, and G, a total number of 64 encodings may be created. However, only 20 amino acids must be encoded. Thus, redundant encodings are possible, mapping various codons to the same amino acid. In order to control the building process of proteins, start and stop signals are encoded by codons as well. There exist one start codon AUG and three stop codons UAA, UAG and UGA. 61 out of 64 codons encode for a certain amino acid. While the start codon encodes for the amino acid methionine, the stop codons do not encode any amino acid. There exist a specific tRNA molecule for each of the 61 amino acid encoding codons, where each tRNA possess the complementary base triplet of its associated codon. Thus, the start codon AUG has an associated tRNA molecule featuring the triplet UAC, which is called anticodon. The entire set of mappings from codons to amino acids,

start and stop signals is considered the genetic code.

The translation process is composed of various phases: initiation, elongation and termination. Before the decoding of codons is started, the initiation phase occurs. A small ribosomal subunit (20S) is bound to the 5' cap of the mRNA molecule, which is also called the leader sequence of mRNA. A set of protein factors embrace the 20S subunit and support its binding process. In the next step, the 20S subunit is moved along the mRNA molecule in the 5'-to-3' direction searching for the start codon AUG. After the start codon has been localized, a free tRNA molecule (initiator tRNA) carrying the first amino acid methionine is attracted. Both the ribosomal subunit and the initiator tRNA form the initiation complex. Then the elongation phase is initiated by releasing the protein factors and binding another ribosomal subunit (60S) to the 20S subunit. Together, they build the 80S ribosome, which is in charge of sequentially binding amino acids together. During elongation phase, all codons between the start and the end codons are stepwise read, corresponding tRNA molecules transport amino acids into the ribosome, and the amino acids are tied together in a growing polypeptide chain. The ribosome moves continuously along the mRNA strand until the end of the molecule [52]. The 80S ribosome has three adjacent binding sites for tRNA molecules. The tRNA molecule, which anticodon matches the currently read codon is first bound to an aminoacyl binding. Then, the tRNA molecule is bound to the peptidyl binding site, which is connected with the polypeptide chain. The amino acid of the tRNA molecule is bound to the polypeptide chain. In the next step, the amino acid is dissociated from the tRNA molecule which moves to the exit binding site, where it is discharged from the ribosome [21]. The released tRNA molecule may bind a free amino acid and participate in the synthesis process again. The elongation phase is terminated when the ribosome reaches a stop codon. Since no matching tRNA molecule exist for the stop codons, the translation process is stopped. The ribosome releases the last tRNA molecule and the ribosomal subunits

are dissociated from each other. The polypeptide-chain, which is now a complete protein, is separated from the ribosome. A single mRNA molecule may be translated by several ribosomes at the same time. After one ribosome started translation and moves along the 3'-direction of the strand an other ribosome may be attached to the 5' cap and begin with its initiation. Thus, this mechanism facilitates high production rates of proteins. For instance, a plasma cell may produce 2,000 antibody molecules in a second [52].

DNA molecules are a precious source for various areas of application. For instance, they are used in the detection of genetic fingerprints in forensic science or for identifying disease-related genes. Though, if the quantity of available DNA is too low, it is insufficient for further analyses. In order to overcome the lack of adequate quantities of DNA molecules, the laboratory technique polymerase chain reaction has been invented.

3.4 Polymerase Chain Reaction

This technique allows to amplify a certain sequence of DNA up to an arbitrary quantity. That is, in an iterative process, a fragment of DNA serves as a pattern, that is replicated in large amounts of new DNA molecules. Polymerase chain reaction is composed of a sequence of execution cycles, whereas each cycle consists of three steps: denaturation, primer annealing and DNA replication. The reaction is executed in a thermal cycler which is used to change the environment temperature. In the denaturation step, the double-stranded pattern DNA molecule is split in its strands by heating the mixture in the reaction tubes up to 95 degrees. The hydrogen bonds of the DNA are decomposed and the single strands are exposed for the next step. The temperature is lowered and one primer molecule for each strand is annealed. The primer molecules mark the begin and end areas of the DNA fragment to be replicated, whereas the start sequence is flagged on one DNA strand and the end sequence on the other. In the DNA replication step, heat-resistant enzymes catalyze the DNA polymerase process, in which complementary nu-

cleotides are added to both pattern strands. Both primers are extended in the 5'-to-3' direction and grow continuously. Since polymerase is an enduring process, it would last until the end of the DNA pattern has been reached, adding approximately 1,000 nucleotides per minute. However, as only a subsequence of DNA is to be replicated, the replication must be stopped by external regulation. The replication step is terminated by increasing again the temperature which forces the strands to separate from each other. The result of the first cycle are 4 single stranded DNA molecules, two long strands of the original DNA molecule and two short replicated strands. One replicated strand has the requested start sequence and a too long tail sequence. The other replicated strand has the correct end sequence and a too long leading sequence. After the first production cycle has been completed, the second cycle may be started. As the strands are separated, the denaturation step has already been accomplished. In the subsequent annealing step one primer is bound to each of the four strands. DNA polymerase enzymes synthesize four new DNA replicas and the replication process is again interrupted by raising the temperature. After denaturing the DNA molecules, eight single strand DNA are present: two strands of the original DNA, four DNA segments with a correct beginning or starting sequence and finally two DNA segments having a correct start and a correct end sequence. These two molecules are the desired end products of replication and are called target copies. In all following replication cycles, the number of target copies grows exponentially, while the number of other copies grows linearly. Within 20 cycles, more than a million target copies are produced. Typically, PCR is stopped after 30 cycles, where over one billion target copies are available [98].

3.5 Quantification of Genetic Activity

As already described, genes are substrands of DNA that encode for a certain protein or RNA. While the entire set of genes is equal in all cells, at any given time only a subset of genes is active in a cell. An active gene is a gene,

that has been transcribed into mRNA and these mRNA are still present in the cell. Active genes are said to be *expressed* in a cell. Therefore, the notion of gene expression has been introduced, indicating that there exists a quantity of mRNA transcribed from a certain gene at a given instant. The degree of genetic activity or expression may be determined by measuring the quantity of present mRNA molecules. The more identical mRNA molecules are available, the more related proteins or RNA are produced. Within a single cell, various genetic activities occur in parallel. That is, a set of genes is expressed contemporaneously. Some of the genes are only expressed at a low quantity. For example, in cases where protein synthesis has almost been completed, most of the mRNA molecules have been dissociated. Other genes are said to be overexpressed, as a large quantity of associated mRNA is present. The entire set of genes that is expressed at a given instant is referred to as transcriptome.

Monitoring genetic activities has been proved as a challenging method for addressing research questions in various research disciplines. Basic biological activities such as cell cycle or cellular differentiation are regulated by genes. The role and function of genes in this context are manifold. Genes may initiate biological processes and terminate them. They may act as enhancers or inhibitors of biological actions. Further, they may activate other genes, which in turn control or influence biological processes. By understanding the inherent roles and functions of genes, the effects of diseases on the genetic level could become more transparent. Thus, investigations on genetic activities are relevant for diagnostic research as well as for treatment of diseases and drug development [74]. The process of determining genetic activity is also called gene expression profiling. Various techniques exist for determining genetic activity by detecting and measuring the levels of gene expressions. In the following, two well established procedures in medical research are presented: real time polymerase chain reaction and microarray analysis.

3.5.1 Real-time Polymerase Chain Reaction

Real-time polymerase chain reaction (RT-PCR) is a technique for amplifying DNA molecules and additionally quantifying the amount of present DNA. RT-PCR is based on polymerase chain reaction (Section 3.4), which allows to generate replicas of DNA strands in a sequence of production cycles. RT-PCR is used to obtain a quantitative measurement of mRNA. Since low amounts of mRNA within cells impede the assessment of mRNA quantities, the molecules are amplified up to a detectable amount. Special probe molecules (reporters) are labelled with fluorophores, which are fluorescent molecules capable of emitting light of a certain color. The fluorescent signals may be detected and quantified in measures of intensity. The degree of measured color intensity allows to draw a conclusion on the quantity of mRNA available.

Before RT-PCR is initiated, mRNA are transformed into more stable cDNA (complementary DNA) molecules. A primer is bound to the mRNA molecule and the enzyme reverse transcriptase synthesizes a DNA molecule based on the nucleotide sequence of the mRNA. The resulting molecule is a double-stranded DNA, which is more robust than mRNA. The process of synthesizing a new DNA molecule on the basis of a RNA is referred to as reverse transcription. Similar to PCR, RT-PCR consists of the following steps: denaturation, annealing and replication. During denaturation, the cDNA molecules are split in two strands. Then, primer molecules are annealed on the strands marking the requested start points for polymerase. Additionally, probe molecules carrying the fluorescent signal molecules are annealed on the strands. A probe molecule is an oligonucleotide, which has a fluorescent reporter dye at the 5' end and a quencher of fluorescence on the 3' end. Oligonucleotides are short sequences of nucleotides, usually up to 50 base pairs, which have been synthesized for laboratory applications. Due to the close proximity of reporter and quencher, no fluorescent signal can be measured, as long as both are bound to the oligonucleotide. When the reporter is cleaved from the probe, a fluorescent signal is measurable. For every gene

that is to be measured, particular primer and probe molecules have to be available [11].

The enzyme polymerase sequentially binds free nucleotides to the strand, starting from the primer, moving along the 3'-to-5' direction of the strand, until it reaches the probe molecule. Polymerase causes a part of the reporter molecule to be dissolved from the strand. As a consequence, the bonded fluorescent molecule is released and a fluorescent light signal is emitted. At the end of the replication step of each production cycle, light signals are captured and quantified. Hence, the quantification takes place in real time during all RT-PCR cycles. The intensity of fluorescence increases in the same proportion as the amount of amplified DNA grows. The number of starting copies determines the intensity of fluorescence [11]. Thus, when investigating the gene expression levels of two genes in a probe, RT-PCR is executed for both, and the measured intensities of fluorescence indicate the gene expression levels. In order to normalize the measured fluorescence intensities, they are set in relation to so-called reference (or housekeeping) genes. These genes are in charge of elemental cellular functions and are expressed equally in all cells of an organism.

3.5.2 DNA Microarray

RT-PCR is an adequate technique for precisely determining gene expression levels of selected genes. It is well-suited in cases when genetic activity of certain genes are a-priori assumed or known and the expression of these genes should be demonstrated or verified. However, in order to determine which genes are expressed under certain conditions (e.g. in tumor tissue), different methods are required, which permit to assess expression levels of all genes and filter out genes that are significantly expressed. For this purpose, high-throughput analysis techniques have been developed. They allow to measure expressions of thousands of genes concurrently and retrieve a set of genes that may be relevant for further research.

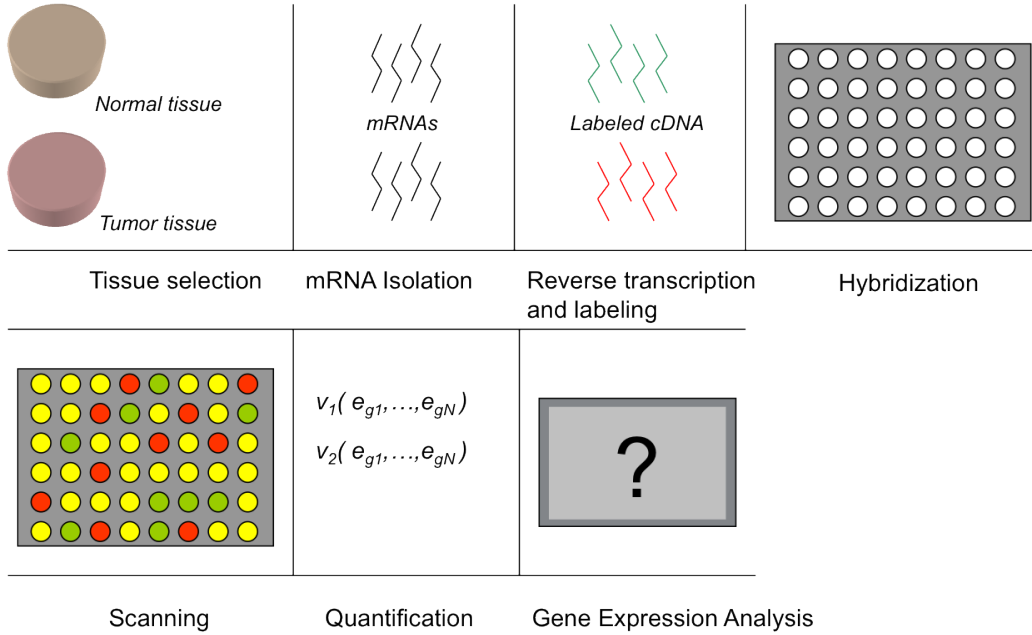


Figure 5: Gene Expression Analysis with Microarrays

DNA microarrays have been proved as a potent technology for simultaneously assessing the gene expression levels of thousands of genes. In a microarray experiment, two samples are compared with each other. Typically, the research focus lies on one sample, for instance a diseased or tumor tissue. The other tissue has the function of a control sample and is referred to as normal sample. The scope of a microarray experiment is the identification of genes, that are differentially expressed in normal and diseased tissue, whereas only those genes with a significant different gene expression are relevant. By comparing two samples, a gene may be similar expressed in both samples, over- or under-expressed in one of the two samples. Since medical research strives to comprehend all regulatory mechanisms of genes, both the over- and under-expressed genes are of importance. Thus, the deviation pattern of all genes, that are significantly differentially expressed in the disease sample, are potential candidates for disease-related genes and could be further analyzed. The processing steps in a microarray experiment are

illustrated in Figure 5.

In the first step, the tissue sample to be analyzed and a normal tissue sample are selected. Usually, an amount of several hundreds mg of cells are required. In the next step, mRNA is isolated from both samples. As mRNA is susceptible to degradation by RNase enzymes, the mRNA molecules must be protected. Therefore, mRNA is separated from RNase enzymes by lysing the cells and chemically induce RNase decomposition. Similar to RT-PCR, the mRNA molecules are reverse transcribed to complementary DNA (cDNA), which are used instead of mRNA in the experiment. In order to distinguish cDNA molecules of both tissue samples, the cDNA molecules are labelled, either with different fluorescence molecules or radioactivity. The labelled cDNA molecules are called targets, which are hybridized to a microarray. A microarray consists of a collection of spots, which are arranged in a matrix on a glass or plastic surface. Each spot contains a set of identical DNA fragments which are called probe DNA. In some cases, cDNA molecules are used as probe DNA, in other cases oligonucleotides are applied. Every piece of target DNA corresponds to a certain gene and is capable of binding a cDNA molecule featuring complementary base pairs.

In the hybridization step, the cDNA targets of both normal and diseased samples are coated to the microarray and bind (hybridize) to complementary probes in the spots. In order to support the hybridization, the microarray is rotated permitting cDNA targets to reach every spot on the microarray. After the hybridization phase has been completed - in Affymetrix Chips, it lasts approximately 14-16 hours - all cDNA molecules that could not be hybridized are washed off. Then, the microarray is dehydrated and put in the scanner. In the next step, the microarray is scanned, and the intensity of fluorescence is measured for each spot. Typically, cDNA molecules of normal tissue are labelled with the dye green and cDNA molecules of diseased tissue with the dye red. If a single spot emits a green fluorescence signal, then the quantity of corresponding cDNA molecules in normal tissue was much higher

than in the diseased tissue. Spots emitting green fluorescence signals indicate that corresponding cDNA molecules prevail in the disease tissue. Finally, yellow spots state, that cDNA molecules of a certain gene have been equally distributed in both normal and diseased tissue. Black spots are indicators, that insufficient quantities of cDNA molecules were present in both tissues resulting in undetectable fluorescence signals [74].

4 Support of Cooperative Work in Context of Biomedical Research

Health care and medical research are closely connected and mutually benefit from each other. As a part of medical treatment, courses of diseases are recorded at medical hospitals. These medical records comprise information about the pathogenesis, medications and cure of diseases. Courses of diseases are a valuable input for medical research projects which are involved in diagnosis of diseases, treatment and drug development. The results of medical research, in turn, may improve medical care at the hospitals. An abundance of data is generated in the routine activities of hospitals. For instance, the Klinikum Graz, a hospital of medium size, provides 512,000 patients with medical treatment every year.

Medical information systems are used for storing and retrieving patient data. They have a strong focus on the daily routine work and were not designed for supporting medical research. On the other hand, various biomedical applications have been realized in recent years, which are highly specialized programs for specific research activities, e.g. the analysis of microarray data. However, all of these systems lack functionalities for cooperative work that are needed in medical research. Within research project, medical data is categorized and annotated and shared among project participants. Thus, there is a strong demand for information systems, that allow to contextualize medical data, enrich it with annotations and share it in context of interdis-

plinary and inter-organisational collaborations. Following the requirements of research projects at the Genome Austria Tissue Bank, a Computer Supported Cooperative Work (CSCW) system was designed. The focus of the CSCW system was on the flexible integration of services and on object-oriented data integration. Data repositories, that are required for research questions, are distributed over various institutes and information systems. In this context, data access restrictions had to be considered by protecting sensitive patient data and granting data access by an elaborated access control system. Further, the CSCW system provides Web 2.0 communication technologies such as wikis, chats, forums and the self-organization of projects. Due to the specific requirements regarding data and service integration, standard CSCW systems could not be deployed in this context. Some approaches have been realized in the area of grid computing. For instance, Armendolia et al. [4] propose an infrastructure for effectively co-working on distributed image data. A similar work exposes distributed data sources on the grid and provides data access by web services [5]. Chu et al. present an environment for accessing distributed medical models. There are works focusing on collaboratively accessing distributed data and services [53, 27, 86]. However, the support of collaboration in biomedical research projects is covered only in a few works [56, 12]. In the following, the collaboration platform GATiB-CSCW is presented, which is a research infrastructure for biomedical research that is built on a service-oriented architecture. More details on the development of GATiB-CSCW can be found in [97] and [88].

4.1 Collaboration Scenarios and Main Requirements

Research projects in context of GATiB are characterized by a high variation of collaborative situations. Biomedical data may be captured, annotated and shared by both routine staff of the hospital and medical scientists. In the following, three exemplary collaboration scenarios are described.

Scenario 1: After a patient has been diagnosed with a rare type of mamma carcinoma, a detailed classification of the cancer is required. A tissue sample is taken and passed to the instantaneous section, where virtual slides of the sample are created. A team of pathologists is responsible for the correct classification and need access to the clinical records of the patient, anamnesis data of her relatives and the virtual slides. Further, the diagnostic team searches for similar mamma carcinoma cases and compares them with the current one.

Scenario 2: A liver cancer research project aims at analyzing courses of disease of liver carcinoma of the last 20 years. Therefore, detailed medical records containing diagnostic data, medication and treatment as well as family anamnesis data are required. Medical experts from the pathology, oncology and surgery are participating in the project. Additionally, the disease-free survival, the long-term survival and overall survival of patients should be assessed by analyzing the follow-up data. A stratification of all liver cancer cases on the basis of tumor subtypes should give information about the distribution and frequency of tumor types. At the end of the project, the stratification of tumor cases is passed to the administration of the biobank, and the analysis results of liver carcinoma cases are used in a publication.

Scenario 3: A biotech company is cooperating with a cancer research group of the pathology in context of a preliminary study used for drug discovery. The study requires a set of diseased and normal human tissue samples for the construction of a tissue microarray. For all tissue samples used in the project, an informed consent of the donating patients is mandatory. The biobank inventory is searched for adequate samples fulfilling disease specific criteria and are available in sufficient quantities. The identified tissue samples are procured together with associated diagnostic and follow-up data. Since an external organization is involved in the research project, sensitive patient-related data has to be removed. In order to draw conclusions from

the lifestyle of patients on the predisposition of the disease, questionnaires are filled out. The collected data is provided to both the pathological scientists and the industrial partners. The results of the study are finally shared with other research groups.

- **R(1) User and Role Management:** The CSCW system is deployed in a highly interdisciplinary environment, in which users of different organisations with various competences cooperate. A flexible user and role management is required to grant access rights to relevant resources and consider data protection mechanisms. Further, cooperation over organisation borders should be encouraged, as research groups often consist of experts from various domains and group members join or leave the team during the project.
- **R(2) Transparency of physical storage:** Data resources are frequently extracted, transformed and integrated from external sources. The end user should access data resources without caring about the physical location of the original sources. Complex data transformations and data structured are hidden from the user. However, the system must ensure traceability of data integration. Data changes in external sources must be reproducible in the CSCW system. Appropriate mechanisms for data search and data retrieval have to be supplied.
- **R(3) Flexible data presentation:** When researchers with different domain knowledge (medical, biological, mathematical, technical experts) cooperate, adequate presentation of data resources is a key factor. Flexible data views are required ensuring that relevant information is presented to users, without confusing them with non-domain knowledge. Users should be able to define individual data views on demand and share them with dedicated users or groups.
- **R(4) Flexible integration and composition of services:** Various software tools have been developed in the context of biomedical re-

search. Typically, these tools have a dedicated focus and are specialized in designated research questions. For instance, before gene expression profiles can be analyzed, the measured ratios of microarray experiments must be normalized. There are various tools providing normalization of microarray data, such as ArrayNorm [75] or the marrayNorm package of Bioconductor [91]. The CSCW system should be able to integrate research-specific tools as services, allowing to pass parameter values by standardized input and output interfaces. Users should be able to execute scientific tools on local or remote computational resources and should not be bothered by technical issues. By encapsulating the functionality of tools in web service interfaces, tools may be installed on the server infrastructure that is distributed over cooperating departments. For instance, scanning of tissue slides may be conducted in the laboratory the slides are created. Another key aspect is the ability of combining the functionality of tools in service compositions. By specifying an execution sequence of services and passing parameter values, complex research activities may be executed autonomously.

- **R(5) Support of cooperative functions:** Collaborative activities between users and groups may be efficiently supported by Web 2.0 technologies, that facilitate communication across organisational borders. Flexible creation of cooperating groups, simplified sharing of data and the possibility of cooperatively creating new information are the main requirements in this context. The CSCW system should provide wikis, discussions forums and blog functionality. Users should be able to extend and edit document contents using a user-friendly markup language and discuss topics in linked forums. Content changes are traceable in the revision history and generated information can be marked with tags. Further, synchronous communication techniques are required for real-time communication activities. Therefore, appropriate interfaces for the integration of instant messaging frameworks are needed.

- **R(6) Data-coupled communication mechanisms:** Aside from typical communication techniques, scientific work requires focused views on data tables, figures and images and the ability to integrate these objects in communication acts. Researches should be able to select relevant data and contextualize it in topics and discussion forums. Moreover, embedded data objects may be enriched with annotations, tags and linked with other objects. For instance, images of diseased tissues are cooperatively diagnosed by a group of expert pathologist. Image sections are marked and annotated, and diagnostic discussions are held in the same place.
- **R(7) Knowledge creation and knowledge processing:** Scientific research strives to acquire novel findings by analyzing relevant data sources, contrast the obtained results with related research and verify or falsify suggested hypotheses. Hence, analysis results may help to gain new insights and create or enhance knowledge. The challenge of the CSCW system is to facilitate the creation of new knowledge by seamlessly integrating data objects and offering data linkage, transformation and analysis functionality. A data object may be considered as an information, that was extracted from a medical information system or document or was entered manually. A knowledge object is the result of a non-empty sequence of functions (services) applied to at least one data (or knowledge) object. In order to provide transparency and support the learning process, all transformation steps are to be captured. A detailed recording of data transformations allows to document the development process of new findings and to repeat scientific experiments.

4.2 Wasabi-CSCW System

The GATiB-CSCW system is an adaption and extension of the collaborative work framework Wasabi which was developed on the university of Paderborn [87]. Wasabi is based on the service-oriented architecture of JBoss which is an application server providing high-performance and scalable execution of enterprise Java applications. The service-oriented architecture of JBoss allows to flexibly integrate, adapt and maintain local and external services which are important features for a modern CSCW system. Additionally, the service-orientation of Wasabi facilitates data integration from distributed data sources. The GATiB IT infrastructure is scattered across various institutes and laboratories. Data repositories are integrated by accessing file-based resources of shared network devices or by accessing interfaces to domain-specific databases (e.g. clinical databases). Data integration services allow an on-demand access of distributed data sources, without replicating entire data repositories to local resources. However, user access rights and restricted access to sensitive data must be considered as well.

Wasabi provides knowledge spaces for supporting the collaboration of users and groups. A knowledge space may be considered as a virtual environment that is defined for a certain collaboration context (e.g. a clinical study or a research project). Knowledge spaces allow to integrate, share and annotate relevant data sources and embed various communication channels such as discussion forums, chats, wikis and blogs. After a knowledge space has been set up for a group of cooperating actors, structuring of information is accomplished by users on their own. This self-organizing characteristic implies a high degree of flexibility and facilitates communication across organizational borders. Information exchange between knowledge spaces is encouraged by allowing data sharing with related knowledge spaces. For instance, separate knowledge spaces may be constructed for similar research projects operating on the same data sources. Intermediate results and findings may be exchanged between the knowledge spaces. An illustration of

the Wasabi knowledge spaces is given in Figure 6. A collaboration group is defined for every knowledge space, in which data objects and documents are shared and supplemented with context-specific information. Knowledge spaces may be structured hierarchically. That is, a knowledge space may be split in various sub-knowledge spaces. Further, relationships between knowledge spaces treating similar topics may be defined. Related knowledge spaces may benefit from each other by monitoring progress in similar research topics and exchanging results.

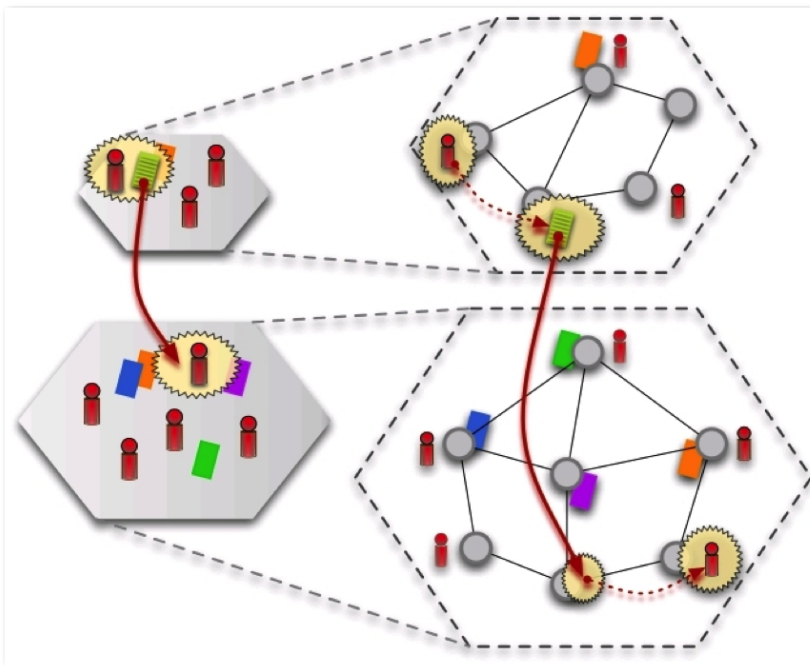


Figure 6: Wasabi - Virtual Knowledge Space

The Wasabi server is composed of four main parts: Wasabi core, EJB services, communication services and remote API. Wasabi core provides the main framework of the CSCW system. A generic data model is used for representing all collaboration-specific data. The basic classes are Container, Room, Document, User and Group, whereas all basic classes are subclasses of the generic WasabiObject. Knowledge spaces are modelled by class Room,

which is a container for embedded data (documents, data objects) and users. Users and groups are represented by the corresponding basic classes. The EJB services provide the main functionality for handling objects of the basic classes. For instance, the EJB service User Manager is responsible for reading and manipulating User objects. User authentication and authorization is accomplished by accessing an LDAP server via JNDI. The EJB service Document Manager is capable of accessing remote content repositories using the JCR (Java Content Repository) API. Database resources may be integrated by using the object-relational capabilities of Java Hibernate or JPA (Java Persistence API). The transformation and mapping of database records to Java objects is accomplished by the EJB service Object Manager. Wasabi communication services implement various protocols for message exchange with external communication services. Currently, communication with news-group servers is implemented by using the NNTP protocol. Synchronization of chat communications is based on the chat protocols IRC and Jabber, email synchronization is based on POP3. The remote API allows to expose CSCW services to consuming clients. A common interface provides access to adapted CSCW services, which may be called by standardized client requests.

The embedding of the CSCW system into the IT infrastructure of GATiB has already been published by Eder et al. [25]. In the following an overview of the IT system architecture of GATiB is given. In the context of the MUG biobank several types of information systems are accessed, as illustrated in Figure 8. The different data sources are integrated in a database federation, whereas interface wrappers have been created for the relevant data. On the one hand, there are large clinical information systems which are used for routine diagnostic and therapeutic activities of medical doctors. Patient records from various medical institutes are stored in the OpenMedocs system, pathological data in the PACS system and laboratory data in the laboratory information system LIS. On the other hand research databases from several institutes (e.g. the Archimed system) containing data about medical stud-

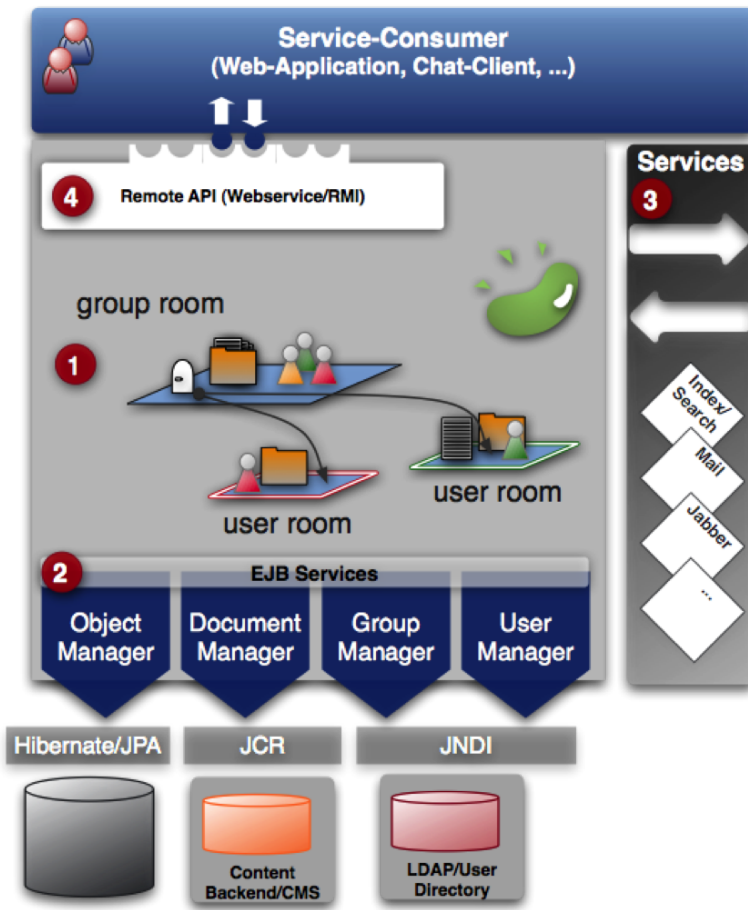


Figure 7: Wasabi - System Architecture

ies are incorporated as well as the biological sample management system SampleDB and diverse robot systems. Further, survival data of patients is provided by the external institution Statistics Austria. Clinical and routine information systems (at the bottom of Figure 8) are strictly separated from operational information systems of the biobank. That is, sensitive patient-related data is only accessible for medical staff and anonymized otherwise. The MUG Biobank operates an own documentation system in order to protocol and coordinate all cooperation projects. The CSCW system at the top of the figure provides a scientific workbench for internal and external project

partners, allowing to share data, documents, analysis results and services [25].

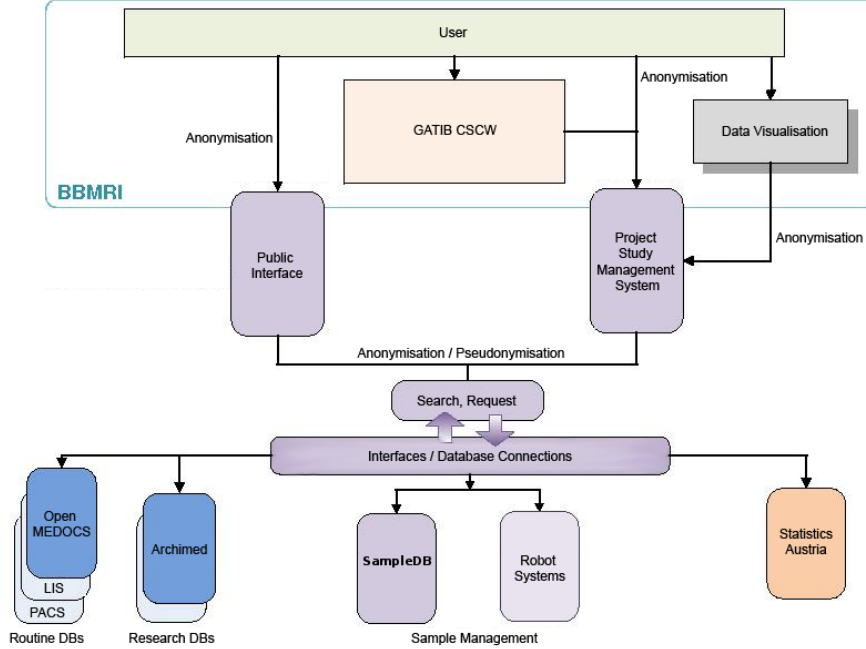


Figure 8: Integration of GATiB-CSCW in the MUG IT Infrastructure

5 Scientific Workflows and Data Provenance in Context of Medical Research Information Systems

Scientific workflow management systems (SWfMS) may assist researchers in defining, executing and monitoring experiments. They strive to integrate various data repositories and computational resources and offer researchers convenient interfaces for specifying and enacting sequences of processing steps in order to prove or reject scientific hypotheses. SWfMS are used to provide a process-oriented view on so-called *virtual experiments* that are conducted on

components of an IT infrastructure. By abstracting from technical details such as database access, data conversion, data transport and execution of heterogeneous software tools, scientists are able to focus on domain-specific research questions. This chapter covers the main aspects of the usage of scientific workflows in medical research projects. Section 5.1 outlines the basic concepts of scientific workflows and differentiates them from business workflows. In Section 5.2, a specific use case from the GATiB project - the analysis process of gene expression profiles - is presented, which is to be realized as a scientific workflow. From this use case, a set of requirements is derived that have to be fulfilled by a suitable IT research platform. Since some of the requirements are concerned with traceability and reproducibility of research results, data provenance issues do have to be considered. Provenance may be defined as *the background knowledge that enables a piece of data to be interpreted and used correctly within context* [80]. It allows to protocol the origin of data, how it was accessed and how it was transformed into other data. Regarding the requirements of the GATiB project, data provenance is used for validating results of gene expression analysis. A set of data provenance queries is defined in order to extract relevant information about medical records, associated genetic data and processing steps of the analysis. Section 5.3 presents different kinds of data provenance and the Open Provenance Model, which is a standardized model enabling interoperability between various SWfMS. A detailed survey on established SWfMS is given in Section 5.7. By comparing the functionality and features of these systems, we recognized, that some essential requirements of the GATiB project are only partially realized or not covered by the systems at all. Therefore, we extended our medical CSCW system, presented in Section 4, in order to incorporate execution of scientific workflows and tracking of data provenance. The technical details of realizing the gene expression analysis workflow are given in Section 5.4. Section 5.5 outlines, how the support of scientific workflows has been integrated into the medical CSCW system. A data provenance model

is presented, which is capable of answering standard provenance queries plus special queries relevant for the gene analysis process. Further, the provenance model is compatible with the Open Provenance Model allowing to exchange and compare provenance data with other SWfMS.

5.1 General Aspects of Scientific Workflow Management Systems

The Workflow Management Coalition defines a workflow as follows: *”Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal”* [42].

Scientific workflow management systems support researchers in planning, executing and documenting experiments. Usually, experiments are composed of a sequence of execution steps (tasks), whereas some require human actions and others are performed by computational resources. IT infrastructure has become an essential part in assisting scientific activities, particularly in biomedical research. Huge data repositories capture knowledge of research results, complex algorithms support analyses and experiments are documented electronically. However, IT resources may be distributed over various local systems or public infrastructure. For instance, information about functions of genes is maintained in publicly available research databases. On the other hand, experimental data (e.g. clinical parameters of patient cohorts) is locally available as well as powerful server infrastructure for data analysis. Frequently, scientists access research databases by web interfaces, download relevant data and merge it with local experimental data. Aside from the risk of generating error-prone data due to human mistakes, the consolidation of data is a cumbersome task, especially, as experiments are typically repeated with slightly modified or supplemented data. Therefore, it is desirable to automate experimental steps wherever it is possible. Data extraction, transformation and consolidation are to be encapsulated in IT services as well as

data movement and analysis processes. Scientists should be able to focus on their domain knowledge and research questions instead of being distracted by technical issues like data management.

Scientific workflow management systems aim at automating execution steps of experiments achieving reliable experimental results, minimizing execution time and providing comfortable mechanisms for controlling the execution of experiments.

Generally, a scientific workflow traverses various phases, which may be summarized in a workflow life cycle as proposed by Ludaescher et al. [55]. In Figure 9, the five steps of the scientific workflow life cycle are illustrated. Starting with a research question, a set of hypotheses or experiment goals are defined by scientists. In order to investigate the hypotheses, an appropriate experimental workflow is designed. A task repository contains a set of execution steps, of which a subset may be selected and arranged in an execution sequence, which specifies the control flow of the workflow. Additionally, the data flow between tasks has to be defined. Each task receives a set of input data items and generates a set of output data items. Input data items may be both single values or data files. Some of the input data items are required as parameters controlling the execution of the task (e.g. pi-value for statistical analysis). A data dependency for every output data item of a task, that is an input data item of an other task, has to be defined.

In the next step the workflow is prepared for its execution. The scientist selects all required initial input data items from a data repository and sets suitable parameter values for all tasks. During workflow execution, data items are assigned to tasks which in turn generate new data items for subsequent tasks. The workflow execution is monitored allowing to assess the current status of execution. Finally, the results are presented in the post-execution phase. All generated data items are listed and may be analyzed by scientists. As already mentioned, scientific workflows are typically iterative processes, that are repeated several times with similar parameters. For

instance, in gene expression analysis different gene expression levels are detected in disjunct person groups. One analysis run may detect significant genes comparing persons with a long-term survival to those with a short-term survival. In another run, long-term survivors may be grouped in age groups or discriminated by sex. Thus, the same gene expression profiles are analyzed in the same workflow but grouped differently. Alternatively, gene expression profiles may be grouped identically but analyzed by a different statistical algorithm. By redefining experimental goals, parameters of the scientific workflow are changed and a new execution cycle may be triggered.

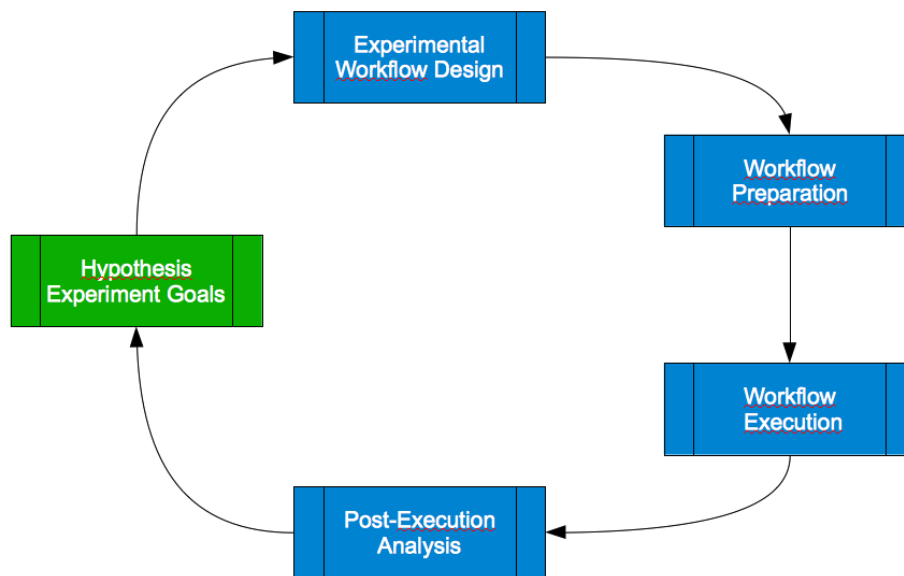


Figure 9: Life Cycle of a Scientific Workflow [55]

Scientific workflows are close related to business workflows. However, they originate from different communities and are applied in different scopes. Business workflows aim at coordinating the work of humans by structuring individual tasks into process structures that may be executed in organiza-

tions. Human interaction is an essential criteria as well as role and right management. By contrast, scientific workflows are job-oriented focusing on planning and execution of automated computations in context of research experiments. The main aspects of scientific workflows are the efficient staging of data on computational resources, managing the execution of computer-intensive calculations and documenting the results. As opposed to business workflows, scientific workflows are executed by fewer persons who specify parameters for automated workflow execution instead of being assigned tasks by the workflow management system like in business workflow execution [60]. Process modelling is a key aspect in business workflows. By assessing the sequence of execution tasks, involved persons and roles the entire process may be represented, whereas only the automatable parts of a process are transformed into a workflow. In scientific workflows, process modelling is strongly influenced by the experimental design. These workflows are constructed in order to facilitate automatic execution of experimental steps. While models of business workflows are frequently target of redesign and optimization, models of scientific workflow are rather stable. Therefore, process modelling has not been paid much attention in the scientific workflow community. Further, both workflow types differ from each other regarding the goals they attempt to achieve. Business workflows fulfil a-priori known business goals, such as handling of orders or supplying of products. By contrast, the results of scientific workflows are used to accept or reject hypotheses. Depending on the outcome of a scientific workflow, the same workflow may be rerun with modified parameters, a related workflow is executed or the experiment is stopped as the hypothesis could be verified. Another important difference concerns the number of concurrent workflow instances during execution. In business workflows, multiple workflow instances are executed in parallel and are independent from each other. For instance, a large number of customer orders are handled at the same time. Usually, scientific workflow systems execute a much lower amount of workflows in parallel. A scientist may ini-

Aspect	Business workflows	Scientific Workflows
Focus	Coordination of work	Automation of work
Users	Many users of different departments	Few users of research team
Goals	A-priori known business goals	Testing hypotheses
Parallelism	Several hundred or thousand parallel instances	Few parallel instances
Execution type	Execution determined by control actions	Data-driven pipelined execution

Table 2: Differences between Business and Scientific Workflows

tiate contemporaneously a set of similar or related workflows which are part of the same experiment. Though, these workflows may last for hours or days and are computer-intensive. Since the execution of scientific workflows is automated with little or no human interaction at all, this kind of execution is also considered as a pipelined execution. After all necessary initial input data is available, the entire workflow may be executed autonomously. The execution of single tasks of a workflow is controlled by the data flow between them. Whenever a task is finished, the generated data is passed to its subsequent task which in turn starts its execution. That is, the data flow implicitly defines the control flow of the workflow. In business workflows control flow is modelled and executed separately from data flow. The workflow management system is in charge of executing tasks, evaluating transition conditions and passing data items from one task to another [55]. The set of differences between business and scientific workflows is summarized in Table 2.

5.2 Gene Expression Analysis Workflow

One of the key aspects in context of the GATiB project was to support medical research activities based on the biological sample collection of the biobank and on the pathological and clinical data repositories available at the

Medical University of Graz. Since genetic research has gained importance in medical science, many research projects strive to investigate the impacts of genes on human diseases. By understanding the regulation mechanisms of genes and their roles in biological processes, genetic activities that are related to diseases are to be detected. The assessment of disease-related genes may be accomplished by a so-called gene expression analysis in which gene expression profiles of patients are analyzed in statistical tests. Gene expression analysis aims at identifying statistically significant deviations in groups of gene expression profiles. Thus, a medical scientist defines groups of gene expression profiles on the basis of associated medical data. For instance, gene expression profiles of patients with small tumors are compared to those with large tumors. The result of the analysis is a set of genes that have been detected as significantly up- or down-regulated in the investigated groups.

Gene expression analysis is a complex process that involves various data access and transformation steps as well as executing statistical analysis and plotting tools. Further, it is a highly repetitive process, as the same dataset is analyzed in multiple analysis runs with different selection and grouping parameters. In order to automate the analysis process, an IT research infrastructure was required that is capable of executing analysis processes as scientific workflows, managing the analysis results and keeping track of how the analysis results have been generated. In this chapter, the details of the gene expression analysis are presented. A short overview about the required input data, the generated analysis results and the necessary transformation steps is given. The gene expression analysis process is modelled as a scientific workflow with corresponding tasks, control flow and data flow specifications. Further, a set of requirements is defined that have to be fulfilled by an IT analysis platform realizing the gene expression analysis.

From a medical point of view, the results of gene expression analysis may help to gain an insight into the course of a specific disease. If the activity of a certain gene (or a group of genes) can be associated with a certain dis-

ease, these genes may be used as indicators for the disease. In oncology, the presence of specific tumors types may be assessed by tumor marker genes. Moreover, tumor markers are not only used to characterize a tumor but also for determining the current state of tumor development and for obtaining information about the response of tumor treatment. Another application area of gene expression analysis is the identification of tumor suppressor genes, which are genes that prevent the mutation of normal cells into cancerous cells. The genetic variability of humans has an essential influence on the courses of disease and on responsiveness to treatments and medications. Personalized medicine takes into account the individual patterns of genetic activity and seeks to offer customized treatments of diseases. Therefore, the investigation of gene expression profiles of patients linked with their therapeutic measures and clinical outcome can provide valuable information about the adequacy of various medical treatments.

5.2.1 General Requirements

Gene expression analysis requires various types of input data. First of all, gene expression profiles of patients have to be obtained. Microarray experiments are conducted in order to measure the genetic activity in certain cells of human tissue samples. The selection of samples is based on topographic criteria (e.g. organs, skins) and on disease-related criteria (e.g. tumor type). With the help of microarray technology, genetic activities are assessed in a hybridization and scanning process which results in a gene expression profile for each investigated sample. Every activity pattern is stored in a separate result file. For instance, GenePix microarray scanners¹ generate files in GPR file formats. The single gene expression profiles are combined into a gene expression matrix, which is a more compact representation that is used as input format for analysis tests. Further, tissue samples are associated with medical records of patients that are required for classifying gene expression profiles

¹GenePix Microarray Scanner <http://www.moleculardevices.com/x143.xml>

into test groups. Medical data may comprise basic patient specific data (age, sex, body mass index), diagnostic parameters, anamnesis data, lifestyle data (e.g. alcohol and smoking habits), therapeutic and survival data. Medical researchers define test groups based on patient-related data in order to investigate their hypotheses. For example, invasive breast cancer tumors might exhibit a slightly different gene activity pattern than non-invasive tumors. In context of oncology-based research, grouping parameters such as the comparison of malignant and benign tumors, the differentiation of tumors by size or metastasis behaviour are defined. After specifying the grouping parameters, a statistical analysis of the associated gene expression profiles may be executed. The result of the analysis is a set of gene groups, that is significantly up- or down-regulated in one of the compared groups. Usually, statistical analysis tools return a list of gene groups ranked by their significance. A simplified representation of the gene expression analysis is given in Figure 10.

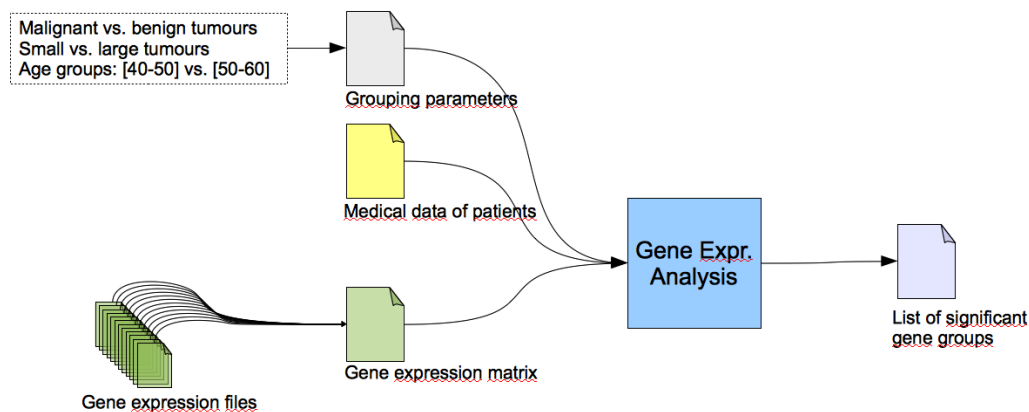


Figure 10: Gene Expression Analysis

While the laboratory infrastructure of GATiB allows to execute highly standardized microarray experiments, there was a lack of IT infrastructure for

combining the generated gene expression profiles with medical databases and executing bioinformatics analysis tools. In the following, the key requirements for the analysis platform are listed.

- **Workflow-based execution:** The entire analysis process should be executable as a scientific workflow. By specifying user-defined input data and analysis parameters, all necessary data selection, data transformation and calls of bioinformatics analysis tools are to be performed transparently without user interaction.
- **Persistence and Repeatability:** The results of workflow runs need to be stored together with all execution-specific parameters. The execution of a workflow should be repeatable in order to validate results. In some cases partial execution of workflows is required. For instance, if results of an analysis should be plotted with different visualization tools, the analysis does not have to be reexecuted.
- **Flexibility and Distributability:** Although, the analysis process is based on a certain set of software tools, flexible addition or exchange of single software components should be supported. If an alternative analysis library should be deployed, it should be integrated with little effort. A distributed execution of workflow steps is encouraged. Statistical analysis of gene expression profiles are computationally intensive and should be executed on dedicated servers. Changing the location of workflow execution steps at runtime allows to flexibly allocate additional computational resources if the current workload increases. Moreover, in cases of system failures computational nodes can be easily exchanged.
- **Different execution modes:** The analysis platform should support two execution variants. On the one hand, a user-interactive execution mode should enable a step-wise execution of the workflow. Therefore,

a wizard-based user interface is required which allows to trigger workflow execution manually and viewing intermediate results. On the other hand, the workflow should be also be executable in a batch execution mode. A researcher assigns all required input data and input parameters at once and triggers the workflow execution without any further user interaction. The wizard-based execution is used for iterative workflow runs on smaller data sets, while the batch-mode is preferred for long-lasting executions on huge data sets.

5.2.2 Data Provenance Related Requirements

When looking in more detail at the input/output data of gene expression analysis, various data structures may be identified. Medical data of patients is captured in a record set that is the result of joining patient data from different databases. Person-related data such as age, sex, body mass index (BMI) may be retrieved from clinical information systems. Disease-related data has its origin in clinical and pathological databases comprising anamnesis and diagnostic data. Much effort has been put on standardizing diagnoses which allows to categorize medical cases and make medical records searchable. The most established disease classification system in healthcare is the International Classification of Diseases (ICD) ² proposed by the World Health Organization. The ICD undergoes several revisions and is currently available in its 10th version ICD-10. In context of GATiB, diagnoses were standardized using the International Classification of Diseases for Oncology (ICD-O-3) which is an extension of the ICD-10 coding system. Further, the classification of tumors is accomplished by the TNM cancer staging system [92]. TNM staging is used for describing the size and behaviour of tumors. The characteristics of a tumor are assessed in three parameters: the size of tumors is specified in parameter T, the presence of regional lymph nodes

²International Classification of Diseases <http://www.who.int/classifications/icd/en/>

metastasis in parameter N and the presence of distant metastasis in parameter M. Further, medical records are linked with corresponding biological material of patients. Each human sample has a unique identifier that is used as a marker for gene expression profiles. Grouping criteria can be defined on both person- and disease related attributes.

Gene expression profiles created by the same microarray experiment are combined into a gene expression matrix. In order to remove systematic biases of the experiment (e.g. varying intensity in measured fluorescence), normalization of the gene expressions is required. A gene expression matrix consists of m rows and n columns whereas each column represents the gene expression profile of an individual and each row the measurement of genetic activity for a certain gene. Columns are labelled with sample identifier while genes are discriminated by gene identifiers that are defined by vendors of microarray chips. Genetic activity is captured in Log_2 ratio values indicating over-expressed genes in positive and under-expressed in negative values. The results of gene expression analysis are groups of genes that have turned out to be significantly differently expressed in the compared groups of gene expression profiles. The effects of genes are accomplished by genetic interactions. Genes may activate and deactivate genes and enhance or suppress their activity. The process of genetic interaction is typically represented by gene regulatory networks in which genes receive signals for initiating or terminating their activity and emit signals influencing the behaviour of other genes. Each biological process has an associated gene regulatory network that consists of a cascade of gene interactions. For instance, the genes FAS, FASLG and IFNG are known to participate in the process of inflammatory cell apoptosis. However, genes may control different processes in multiple gene regulatory networks. Genes are said to exhibit different functions in various biological processes. The assignment of functions to genes is considered as *gene annotation*. In order to standardize the annotation of genes, the

Gene Ontology ³ has been introduced. *"The Gene Ontology is a major bioinformatics initiative with the aim of standardizing the representation of gene and gene products across species and databases"* [17]. The gene annotations are supplied by the Gene Ontology Consortium in form of publicly available ontologies that may be downloaded in various data formats - e.g. as MySQL database dumps or RDF-files. The Gene Ontology integrates several gene annotation repositories. For instance, annotations are imported from the GeneDB of the Wellcome Sanger Trust Institute ⁴ and from gene databases of the European Bioinformatics Institute (EBI) ⁵. Since new gene functions are constantly discovered by genetic research, the annotations do have to be updated continuously which results in daily versions of the Gene Ontology. The Gene Ontology contains annotation of various species including homo sapiens and model organism such as *Drosophila melanogaster* or *Escherichia Coli*. Gene functions are represented by ontology terms and cover different domains: biological process, molecular function and cellular components. The ontology terms are structured hierarchically forming a directed acyclic path. Each ontology term has unique identifier starting with a GO: prefix. An example GO term, representing the apoptosis of inflammatory cells is illustrated in Figure 11. The hierarchical classification of inflammatory cell apoptosis is depicted in the right part of the figure. The term has an associated definition and synonyms. Term GO:0006925 is assigned three known gene annotations for the species homo sapiens which are the protein-coding genes FAS, FASLG and IFNG. Currently, over 34,000 Gene Ontology terms have been defined. For species homo sapiens, more than 18,000 gene products are assigned by over 230,000 annotations to ontology terms.

Gene expression analysis takes into account the annotation information supplied by the Gene Ontology. Instead of detecting single genes that are significantly differently expressed, the analysis strives to identify groups of

³The Gene Ontology www.geneontology.org

⁴GeneDB <http://www.genedb.org>

⁵European Bioinformatics Institute <http://www.ebi.ac.uk/>

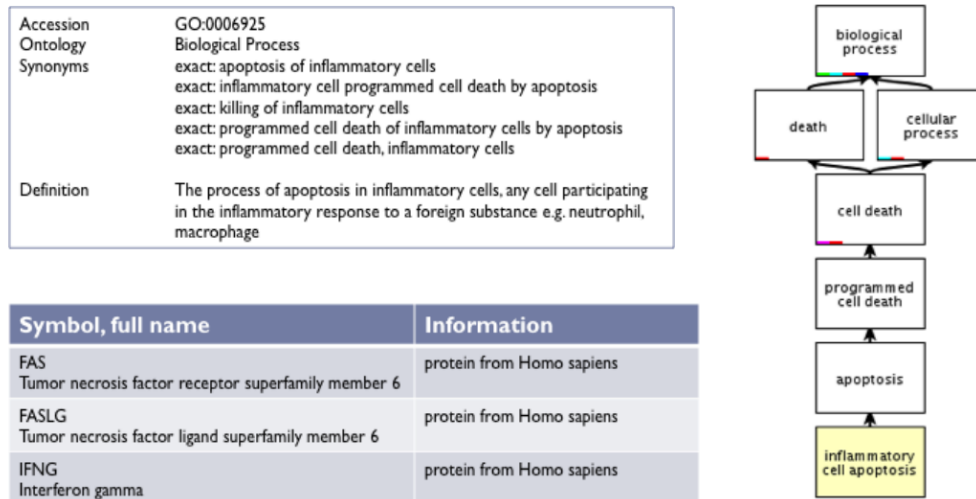


Figure 11: Gene Ontology Example

genes that are known for promoting a certain function. Gene groups are defined by associating genes of the microarray experiment with Gene Ontology terms. A detailed view on the involved input/output data of the gene expression analysis process is given in Figure 12.

While Figure 10 presents input/output data from a simple *file-oriented* perspective, Figure 12 gives a more fine-grained view on the data. Both input and output data may be considered as collections of data objects. Each record in the medical data set is a data object as well as each gene expression matrix entry and each significant gene group. Further, data dependencies between data objects are indicated by dotted lines. Patient records have associated biological samples which are linked to columns of the gene expression matrix. Microarray genes are linked by annotations to Gene Ontology terms. Since only a subset of gene expression profiles is selected for an analysis run, grouping parameters discriminate gene expression profiles into two groups. This *collection-oriented* view on input/output data facilitates traceability of individual objects in processes. The generated set of signifi-

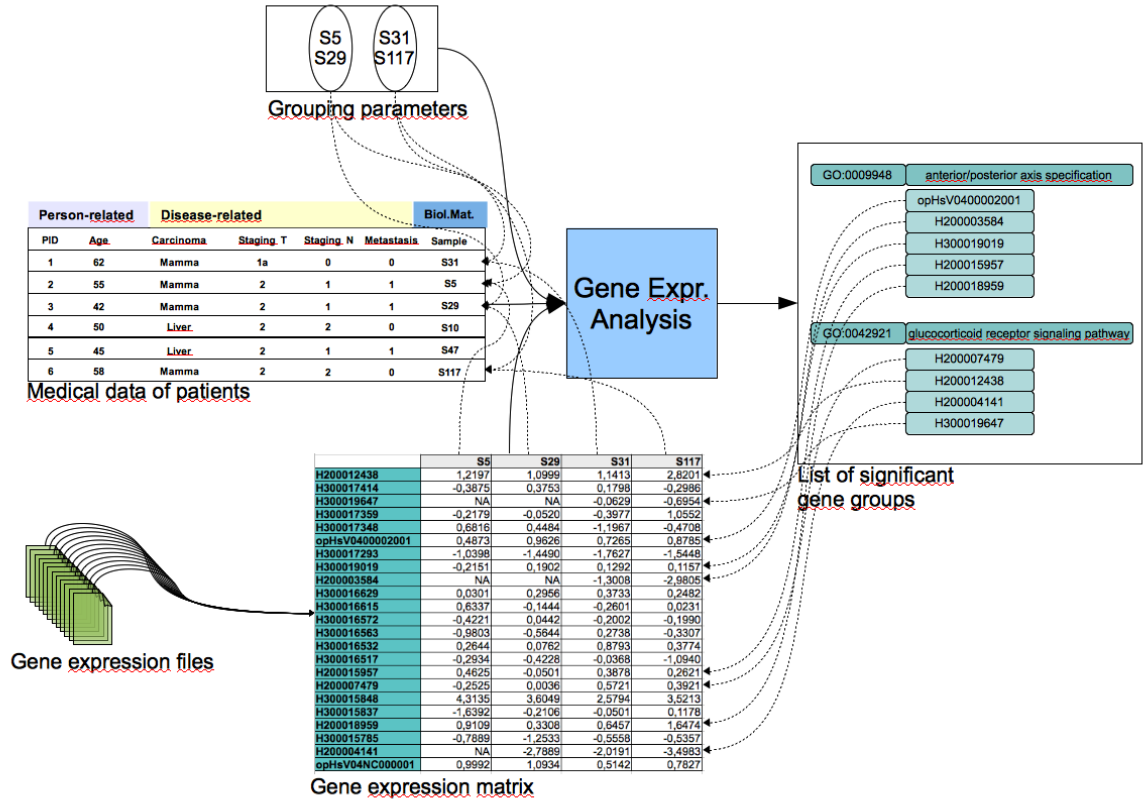


Figure 12: Gene Expression Analysis Workflow - Details

cant gene groups may be seen as the result of a data transformation process which takes individual objects as transformation arguments. The entire set of transformation arguments together with associated transformation steps that were necessary to produce a result set X is also considered as the *data provenance* of X. Data provenance and data lineage are synonyms, as both denote the origin of data in terms of accessing data and deriving new data by transformations. Data provenance may be represented by directed acyclic graphs, in which nodes constitute transformation arguments and arcs transformation steps. On the other hand, data objects may be used in multiple

processes as transformation arguments. A gene expression profile may be analyzed by different researchers in hundreds of analysis runs and significant genes groups may be used as inputs in various visualization tools. In this context, *data usage* of an object specifies in which processes the object was used as a transformation argument. Traceability of data provenance and data usage is vitally important for research processes. When reviewing and comparing research results, questions regarding the quality and validity of involved data do have to be answered. Therefore, it is essential to document which data contributed to research results and provide comfortable mechanisms for querying and re-accessing these data. Further, researchers are interested in research activities that overlap with their own work. If related research projects are based on the analysis of the same data (or part of it), researchers may strive to gain an insight in results of similar research activities. Answering of both data provenance and data usage queries has been a key requirement in the GATiB gene expression analysis. In the following, the set of required queries is outlined.

- Q1: Given a set of significant gene groups, find all relevant data and transformation steps that were necessary to generate the result. The query should return the set of involved gene expression profiles, the biological samples from which cDNA was taken for the microarray experiment and all associated patient data. Further, details of the analysis process such as grouping parameters or thresholds for the statistical analysis should be presented.
- Q2: Given a set of significant gene groups and user-defined filter criteria, return all matching data and transformation steps that produced the result set. Filter criteria restrict the result set by selecting certain data types (e.g. only sample or patient data) or by focusing on certain transformation steps: e.g. only data that was generated after normalization of gene expression profiles.

- Q3: Given a set of gene expression profiles, find all results of gene expression analyses that have been already executed on the set of profiles or on a subset of it. The query should return lists of identified significant gene groups together with related grouping and statistical parameters.
- Q4: Given a specific time interval and certain statistical method, return all results that were generated by the specified method within the time interval. If a researcher wants to extract analysis results, which were created by the statistical test Global Ancova within the last two weeks, lists of gene groups are to be returned. Parameter values of the analysis may be included as filter criteria as well as information regarding the type of medical data. Optionally, all involved input data (gene expression profiles, samples, medical data) should be presented.
- Q5: Given two finished analysis runs, identify the differences in the generated result sets. For instance, if an identical set of gene expression profiles is analyzed by two different methods (Global Test and Global Ancova), all gene groups should be returned that have turned out to be significant in one of the analysis runs but not in the other. Alternatively, all gene groups that have been proved to be significant in both runs should be identifiable.
- Q6: Input data may be enriched with user-defined annotations. Given a certain annotation value or a set of annotation values, return all analysis runs that use matching data as transformation arguments. For instance, a set of tissue samples is collected, taken from patients suffering from the rare liver cancer Angiosarcoma. The collection was established following high quality control standards and detailed diagnostic and clinical parameters were assessed. In order to highlight the quality of these samples, each sample is annotated with a high quality annotation. Another example could be the verification of a significant

gene group by applying a RT-PCR. If the over-expression of a gene group can be proven by a RT-PCR experiment, the gene group may be labelled with a verified-by-RT-PCR annotation.

5.2.3 Data Change Related Requirements

When executing scientific workflows the generated results are assumed to be valid, if all involved input data is valid. Thus, the outcome of scientific workflows strongly depends on the reliability and on the quality of input data. Regarding medical data, there are various factors influencing the quality of data. On the one hand, quality of medical data depends on the accuracy of laboratory tests. For instance, the validity of cancer blood tests may be affected by inaccurate laboratory techniques and instruments. Further, the used test method may be erroneous producing false positive and false negative results. On the other hand, personal skills and level of expertise of medical scientists have a significant impact on the quality of diagnostic data. While a physician may correctly classify the basic type of tumor, a more experienced colleague could be able to augment the diagnosis by identifying the correct subtype of tumor. Another circumstance that has to be considered is the incompleteness of data. In prospective medical cohort studies, a group of individuals suffering from a specific disease is monitored over a period of time. During the observation period, the effects of predefined factors on the outcome of the disease are investigated. Disease-related factors may be life-style parameters such as smoking, alcohol consumption or dietary habits. Further, different therapeutic measures and medications can be included as treatment factors that influence the course of disease. Consider a cancer cohort study, in which the responsiveness of patients to a certain therapy is investigated. After taking tumor tissue samples and generate gene expression profiles, patients are treated with the selected therapy and the long term effects are monitored. After two years, half of the patients responded positively to the therapy, while the therapy had no impact on the remaining

patients. After four years, 10 percent of the originally positively responding patients had a tumor recurrence. If the gene expression profiles of patients are analyzed after two years by comparing positive and negative responders, different gene expression patterns may be found than in an analysis after four years. Thus, relevant changes of input data have to be considered as they can provoke invalidation of analysis results. In the following, the effects of data change in the gene expression analysis workflow are illustrated by a few examples.

Figure 13 presents a record set of medical data that has been created by integrating data from five different sources. Aside from the already presented sample, person and disease-related data, survival and follow-up data has been merged with the record set. The survival data consists of two columns: disease-free and overall survival. Disease-free survival indicates the period of time a patient does not suffer from a disease after a particular treatment has been performed. Disease-free survival is typically measured in months. By contrast, the overall survival specifies the period of time a patient stayed alive after diagnosis of a disease. The overall survival is also measured in months. Based on the experience gained in related cases, an overall survival of more than a specific threshold may indicate a cure of the disease. In our example N/A values are assigned to all patients that are still alive and are assumed to be cured, whereas month values indicate that the patient has already died. Follow-up data concerns therapeutic measures that have been executed after the diagnosis of the disease. Cancers may be treated by different methods such as surgery, chemotherapy, chemoembolisation or radiation therapy. The collection of follow-up data is complicated, since after diagnosis, some patients choose to be treated in different hospitals. In our example, N/A values indicate that there is no follow-up data available for a specific patient.

The entire set of medical data is used as input for a gene expression analysis. Two filter criteria have been defined on the columns carcinoma type

and metastasis in order to define two groups of gene expression profiles that should be compared. Group 1 (S31,S29,S117) is defined on mamma carcinoma cases without metastasis while group 2 (S5,S20,S7) covers mamma carcinoma cases with metastasis. After a gene expression analysis was performed, the diagnosis of the third patient was revised, since metastasis have been found in other organs. The new metastasis value causes S29 to be removed from group 1 and added to group 2. In order to prevent incorrect results, the gene expression analysis has to be repeated. Optionally, the differences in the result sets of the invalidated run and the valid run could be displayed.

A similar example is presented in Figure 14. Two groups of mamma carcinoma patients are defined based on the age of patients. Samples S29 and S20 belong to the group of age interval 35-50, while samples S31, S5, S117 and S7 are part of the group of age interval 50-75. After comparing the gene expression profiles of these group, age interval group 50-75 is further subdivided into patients with positive survival (S31 and S117) and into patients with negative survival (S5 and S7). Again, significant differences in gene expression profiles are identified. Although patient S31 was assumed to belong to the positive survivor group, she dies after an unexpected tumor recurrence. Sample S31 now belongs to the group of negative survivors. As a consequence, the results of gene expression analysis must be reviewed as they are based on wrong classification of gene expression profiles.

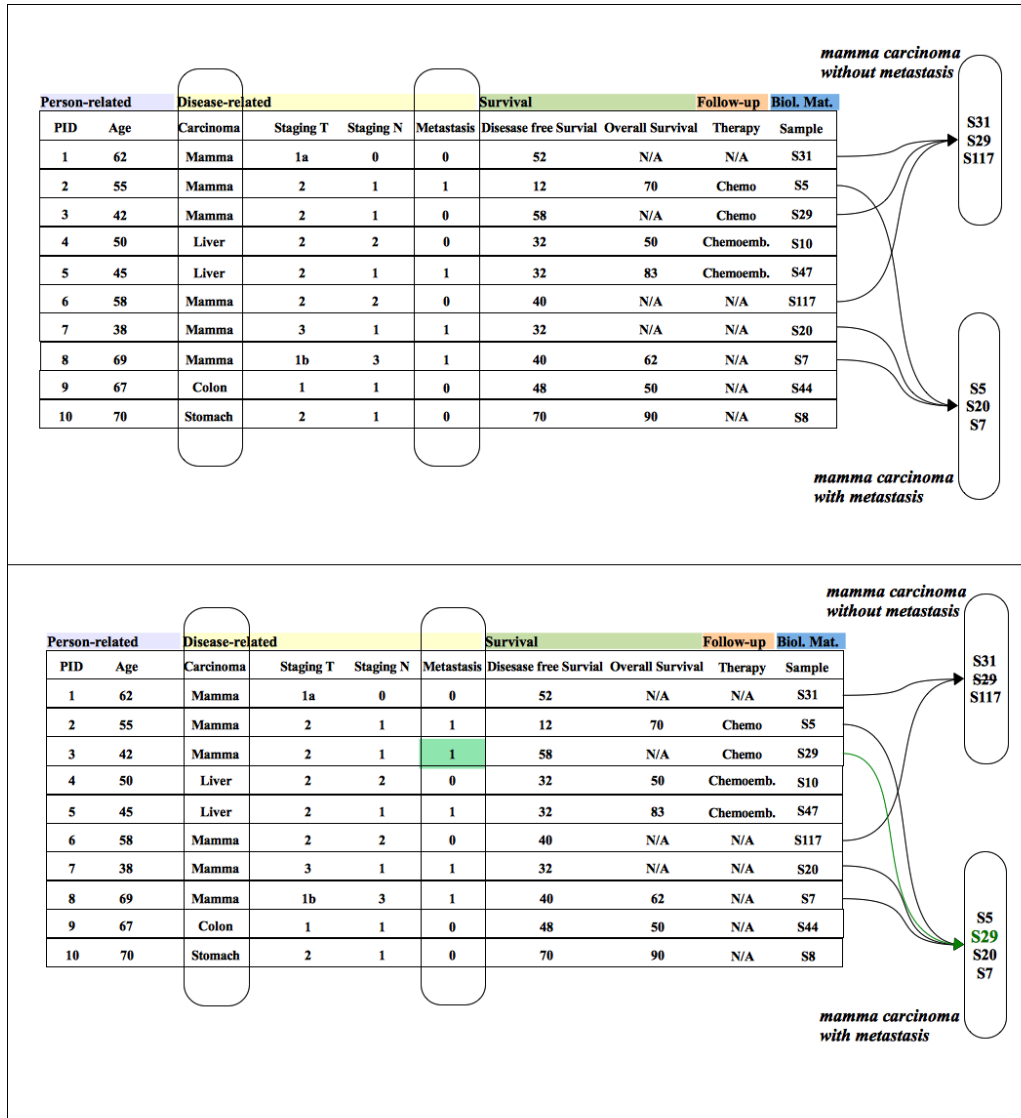


Figure 13: Data Changes in Medical Data I

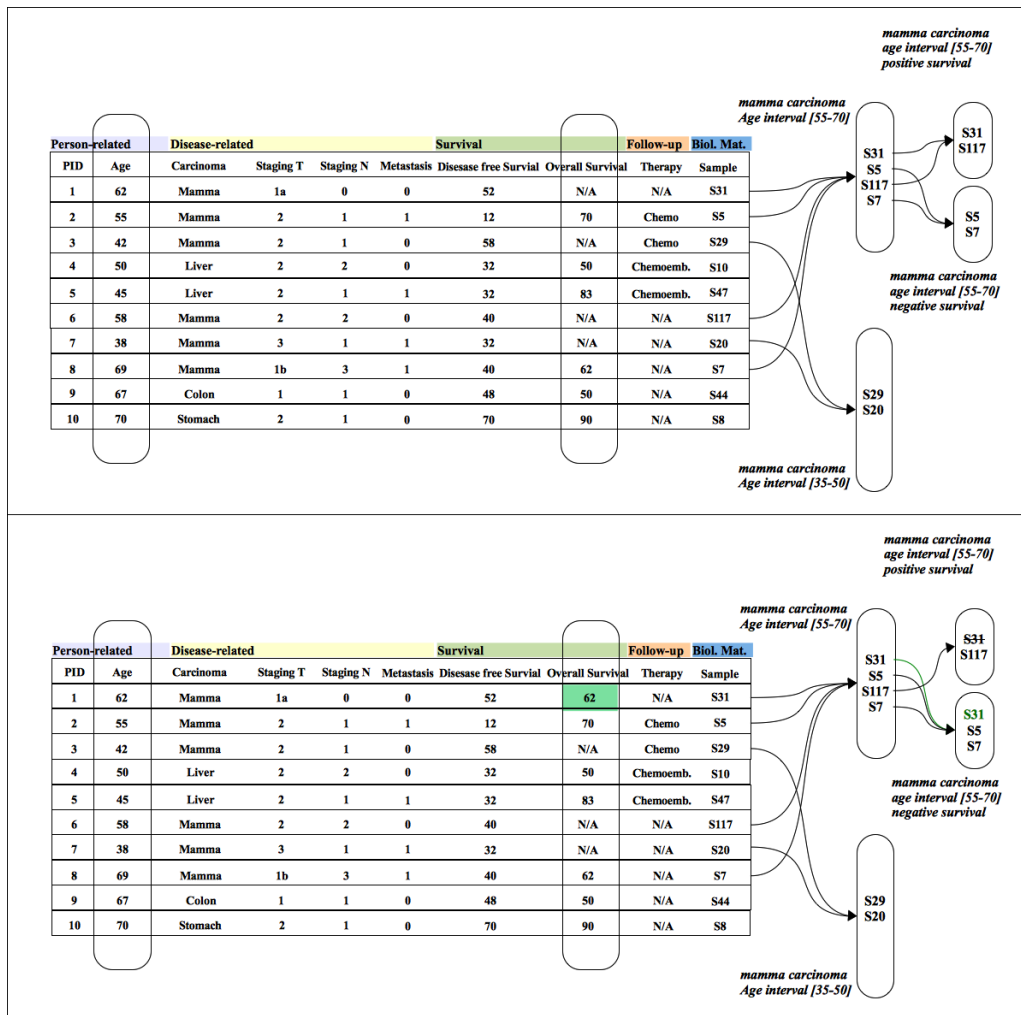


Figure 14: Data Changes in Medical Data II

Aside from medical data, genetic data is affected by data changes as well. For instance, the annotation of gene functions to genes is an incremental and corrective process. Genetic functions are discovered by research results which can be verified or falsified by following researches. The Gene Ontology repository faces several hundred changes every year resulting in different versions of the Gene Ontology. A Gene Ontology version may be considered as the state of the art knowledge of gene annotations at a certain point of time. In Figure 15, the effects of a changed gene annotation are illustrated. Genes of a gene expression matrix have associated gene annotations represented by Gene Ontology terms. Genes FAS, FASLG and IFNG are known to contribute to the inflammatory cell apoptosis (GO:0006925). The annotation is considered in a gene expression analysis process. If an additional gene is identified that is also involved in the inflammatory cell apoptosis, the results of the previous analysis have to be reviewed and verified.

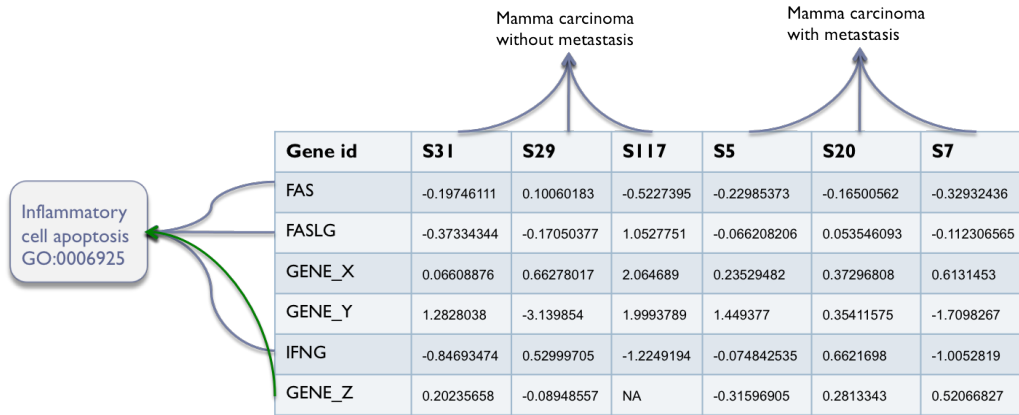


Figure 15: Data Changes in Genetic Data

Changes of input data in scientific workflows must be treated accurately in order to guarantee validity of workflow results. An efficient management of data changes is required allowing to automatically detect workflow results that are affected by data changes. Generally, handling of data changes should be capable of fulfilling the following requirements:

- **Change propagation:** Every change of a data object, that has been used as an input data in at least one workflow execution must trigger a validation of associated workflow results. A differentiation must be made between changes that cause invalidation of results and changes that do not have an impact on them. For instance, if a new synonym of a gene is defined, the change does not affect gene annotations and gene expression analysis results. If an output data object of a workflow depends on the data object that has been changed, it has to be invalidated. All transformation steps generating output data objects that have been invalidated have to be reexecuted. The result of change propagation is a set of newly generated output objects representing valid results of workflow executions.
- **Workflow Execution Plans:** Depending on the set of changed input data, optimized workflow execution plans should be deducible for all workflows that produce invalid results and have to be reexecuted. These workflow execution plans exclusively comprise transformation steps that are affected by data changes and skip steps which input data is still valid.
- **Explanation:** If workflow results are invalidated, a detailed explanation of the reasons for invalidation should be given. All changed input objects that caused the invalidation should be presented. Further, the set of reexecuted transformation steps and all newly generated output data objects are to be returned.
- **Transparency of differences:** If a workflow has to be reexecuted due to data changes, differences between invalidated and newly generated output objects should be identifiable. For instance, if a gene expression analysis has to be repeated because of a changed gene annotation, all significant gene groups that were not present in the first workflow should be detectable.

Person-related		Disease-related			Survival			Follow-up	Biol. Mat.
PID	Age	Carcinoma	Staging T	Staging N	Metastasis	Disease free Survival	Overall Survival	Therapy	Sample
1	62	Mamma	1a	0	0	52	N/A	N/A	S31
2	55	Mamma	2	1	1	12	70	Chemo	S5
3	42	Mamma	2	1	1	58	N/A	Chemo	S29
4	50	Liver	2	2	0	32	50	Chemoemb.	S10
5	45	Liver	2	1	1	32	83	Chemoemb.	S47
6	58	Mamma	2	2	0	40	N/A	N/A	S117
7	38	Mamma	3	1	1	32	N/A	N/A	S20
8	69	Mamma	1b	3	1	40	62	N/A	S7
9	67	Colon	1	1	0	48	50	N/A	S44
10	70	Stomach	2	1	0	70	90	N/A	S8
11	36	Mamma	3	1	1	62	N/A	N/A	S15
12	47	Mamma	2	1	1	50	N/A	Chemo	S248

Figure 16: Types of Data Changes

Different types of data changes may be derived from the presented examples. Figure 16 summarizes the main types of data changes on a set of medical records. Data may turn out to be wrong or outdated, which is illustrated by tumor staging values that result from wrong diagnoses and by survival data that is not valid any more. If data objects are linked wrongly to other objects, all workflow results that are affected by these wrong associations have to be identified. In our example, wrong mappings between patients and associated tissue samples may occur. Further, data may only be partially available resulting in incomplete data. For instance, therapeutic information of patients may be assessed by contacting cooperating hospitals. If gene expressions are compared based on a grouping on therapeutic methods and some of the medical records are amended with therapeutic data at a later date, the analysis may be repeated with a large sample size. If the medical data set is extended by adding further patient records and associated gene expression profiles, workflow results do not have to be invalidated.

However, the additional data may match filter criteria of past analyses. In this case, suggestions may be created showing which experiments could be reexecuted with the new patient records.

5.3 Data Provenance

5.3.1 General Aspects and Application Areas of Data Provenance

Data provenance may be seen from a process-oriented perspective. Data may be transformed by a sequence of processing steps which could be small operations (SQL joins, aggregations), the result of tools (analysis services) or the product of a human annotation. Thus, these transformations form a process which is a *construction plan* of a data object. An additional type of provenance was pointed out by Stevens et al. [79]: organizational provenance. Organizational provenance comprises information about who has transformed which data in which context. This kind of provenance is closely related to collaborative work. There are manifold application areas using data provenance information. Simmhan et. al. [90] proposed the following categorization of application fields of data provenance.

- **Data Quality:** Data provenance can be used as a validation mechanism for evaluating the quality of generated data. By storing the details of data transformation and all involved source data, the generated data is linkable to all associated input data. Since input data may adhere to different quality standards, the quality of transformed data can be critically estimated.
- **Audit Trail:** Monitoring of data transformations is vitally important for ensuring that each transformation has been executed as expected and for retracing transformation parameters and execution times. Further, usage of system resources may be assessed.
- **Replication Recipes:** Detailed provenance data may enable a repetition

of data derivation. If transformation steps are modified - for instance by changing parameters - data derivation can be repeated and generated result sets can be compared.

- **Attribution:** Tracing the usage of data, questions regarding data ownership can be answered and follow-up use of generated data can be identified. If research is based on data sets published by other research groups, results can be linked with corresponding external input data. Published results can be annotated with correct citations and liability of source data may be assigned to appropriate research groups.
- **Informational:** Data exploration can be facilitated by provenance data as metadata about how data was generated may assist users in understanding the process of data transformation. Further, if users are able to efficiently retrieve generated data based on filter criteria, unnecessary duplication of already present data can be avoided.

From the perspective of medical research, relevant use cases exist for all listed application areas. Research projects are frequently based on publicly available data set or refer to similar projects. Transparently coupling research results with external data sources permits efficient data validation and quality control. The execution of analysis processes can be supervised by audit trails which ensure that processes have been completed regularly. If derivation data of a certain analysis result is available, partial reexecution of the analysis is possible.

The attribution function of provenance data is important for tracking the usage of tissue samples in various research contexts. The value of biological material is tightly coupled with the amount, type and quality of associated data. A biospecimen may be used in various studies and projects and may be assigned different annotations in each of it. All generated data that is based on a tissue sample enhances its research value and should be accessible in related research projects. For instance, if gene expression profiles are created

for a collection of tissue samples, the newly generated data should be shared in all research contexts, in which the tissues are used. Finally, provenance data has an informational role in medical research projects by documenting and explaining all data derivation steps that were necessary to produce a certain result. That is, the construction plans of research results can be used as learning processes that may be reexecuted or adapted.

5.3.2 Open Provenance Model

The Open Provenance Model (OPM) is the result of a provenance standardization initiative, the First Provenance Challenge ⁶, which was started at the International Provenance and Annotation Workshop (IPAW'06). At this time, data provenance has become an emerging issue in the scientific workflow research community and has already been implemented in various systems. However, the representation of data provenance as well as provenance capturing techniques and provenance query functionality were realized in manifold ways. The provenance capabilities of the systems turned out to be hard to compare, since the respective realizations were strongly influenced by domain-specific requirements. The research community recognized the need of a common understanding of data provenance and initiated an informative process in which developers of scientific workflow management systems were encouraged to demonstrate the capabilities of their systems by answering a predefined set of provenance queries. The example scientific workflow brain atlases was taken as a use case for formulating the provenance queries. Though, the set of provenance queries may be defined on arbitrary scientific workflows which support annotations of workflow tasks, and of input/output data. In the following, the nine First Provenance Challenge queries are presented in a generalized form. Consider a workflow wf_y that is composed of a set of tasks $T_y = \{t_1, ..t_n\}$, whereas there

⁶First Provenance Challenge <http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge>

is an implicit execution time ordering $t_i < t_{i+1}$. Each task has an associated type which is expressed by $type(t_i, ty_{ident})$. A task type may be a SQL operation, a web service or a library function, identified by a unique ty_{ident} specifier. Tasks possess input and output ports which are filled with parameter values at execution time. Input ports of task t_i are notated as $\{iport_{i,1}, \dots, iport_{i,j}\}$, whereas j is the number of input ports and all k output ports of t_i are defined as $\{oport_{i,1}, \dots, oport_{i,k}\}$. Data flows are modelled by connecting output ports of data generating tasks with input ports of consuming tasks. That is, parameter passing is captured by a set of *connect* relations: $connect(oport_{i,m}, iport_{j,n})$ specifies that data generated at output port m of task i is consumed at the input port n of task j . Parameter values that are filled with external or constant data are not modelled by *connect* relations as well as output parameters that are not consumed by subsequent tasks.

At execution time, workflow instances may be created based on the workflow definition. Workflow instances have unique identifiers, such that $wf_{e,y,k}$ specifies the execution of wf_y and is identified by k . Workflow instances are composed of respective task instances for each of the predefined tasks. The set of task instances is defined as $TE = \{te_{1,k}, \dots, te_{n,k}\}$, whereas $te_{i,k}$ specifies the execution of task i in the workflow instance k . Data objects are used as input and output parameters of tasks. The entire set of all related data objects of $wf_{e,y,k}$ is specified as $D_{y,k} = \{d_1, \dots, d_m\}$. Every data object possesses a certain type that is used for classification. For instance, types may characterize data as patient data or gene expression profile. Data types are specified as follows: $type(d_x, ty_{ident})$.

The assignment of data objects to task executions is represented by *flow* relations: $flow(d_x, te_{i,k}, te_{j,k}, iport_{j,m}, oport_{i,n})$ defines that data object d_x is passed from task instance $te_{i,k}$ to task instance $te_{j,k}$ using input port $iport_{j,m}$ and output port $oport_{i,n}$. External or constant input data is assigned by $flow(d_x, te_{i,k}, iport_{i,m})$ relations. Data objects that are not consumed by

other task instances are specified by $flow(d_x, te_{i,k}, oport_{i,m})$ relations. Both tasks and data objects may be annotated with key-value pairs which are denoted by $annotation(t_i, key, text)$ and $annotation(d_x, key, text)$ relations. In the following, the nine core First Provenance Challenge Queries are defined formally.

- PC1: Based on output data d_x of workflow instance $wfe_{y,k}$, return the set of tasks $TS \subseteq T_y$, that had to be executed to generate d_x , together with all related input and output data objects of the corresponding task instances:

$$\begin{aligned}
 TS = \{ & t_i \bullet t_i \in T_y \wedge \\
 & t_j \in T_y \wedge flow(d_x, te_{j,k}, oport_{j,m}) \wedge i < j \} \\
 & \{ d_i \bullet t_j \in TS \wedge (\\
 & flow(d_i, te_{l,k}, te_{j,k}, iport_{j,m}, oport_{l,n}) \vee \\
 & flow(d_i, te_{j,k}, te_{l,k}, iport_{l,m}, oport_{j,n}) \vee \\
 & flow(d_i, te_{j,k}, iport_{j,m}) \vee \\
 & flow(d_i, te_{j,k}, oport_{j,m})) \}
 \end{aligned}$$

- PC2: Based on output data d_x and task t_y of workflow instance $wfe_{y,k}$, return the set of tasks $TS \subseteq T_y$, that had to be executed to generate d_x and which were executed before t_y , together with all related input and output data objects of the tasks:

$$\begin{aligned}
TS = \{ & t_i \bullet t_i \in T_y \wedge \\
& t_j \in T_y \wedge flow(d_x, te_{j,k}, oport_{j,m}) \wedge i < j \} \\
& \{d_i \bullet t_j \in TS \wedge j < y \wedge (\\
& flow(d_i, te_{j,k}, te_{l,k}, iport_{l,m}, oport_{j,n}) \vee \\
& flow(d_i, te_{l,k}, te_{j,k}, iport_{j,m}, oport_{l,n}) \vee \\
& flow(d_i, te_{j,k}, iport_{j,m}) \vee \\
& flow(d_i, te_{j,k}, oport_{j,m})) \}
\end{aligned}$$

- PC3: Based on output data d_x and tasks t_a, t_b, t_c of workflow instance $wfe_{y,k}$, return all related input and output data objects of the tasks:

$$\begin{aligned}
& \{d_i \bullet t_j \in \{ t_a, t_b, t_c \} \wedge (\\
& flow(d_i, te_{j,k}, te_{l,k}, iport_{l,m}, oport_{j,n}) \vee \\
& flow(d_i, te_{l,k}, te_{j,k}, iport_{j,m}, oport_{l,n}) \vee \\
& flow(d_i, te_{j,k}, iport_{j,m}) \vee \\
& flow(d_i, te_{j,k}, oport_{j,m})) \}
\end{aligned}$$

- PC4: Find all executions of task t_i in which the constant value *const* was used as an input parameter. Return the task instances and all associated data

$$\{te_{i,k} \bullet flow(const, te_{i,k}, iport_{i,m})\} \cup \{d_p \bullet (\\ flow(d_p, te_{i,k}, te_{j,k}, iport_{j,m}, oport_{i,n}) \vee \\ flow(d_p, te_{j,k}, te_{i,k}, iport_{i,m}, oport_{j,n}) \vee \\ flow(d_p, te_{i,k}, iport_{i,m}) \vee \\ flow(d_p, te_{i,k}, oport_{i,m}))\}$$

- PC5: Find all generated output data (excluding intermediate data) that was produced by workflow instances consuming data objects of type ty_{ident} .

$$\{d_q \bullet \\ type(d_p, ty_{ident}) \wedge \\ flow(d_p, te_{i,k}, te_{j,k}, iport_{j,m}, oport_{i,n}) \wedge \\ flow(d_q, te_{j,k}, oport_{j,m})\}$$

- PC6: Find all data objects that were created by task t_j , that was preceded directly or indirectly by an execution of task t_i with input parameter $const$

$$\{d_p \bullet \\ (flow(d_p, te_{i,k}, te_{j,k}, iport_{j,m}, oport_{i,n}) \vee \\ flow(d_p, te_{j,k}, oport_{j,m})) \wedge \\ \exists flow(const, te_{i,k}, iport_{i,m} \wedge i < j)\}$$

- PC7: Two similar workflows wf_y and wf_z are executed by workflow instances $wfe_{y,k}$ and $wfe_{z,l}$. Both workflows share a set of tasks of the

same type and both workflow instances operate on a common set of data objects. Find the set of different tasks of both workflows, and identify all data objects that are present in one of the workflow instances but not in both:

$$\begin{aligned} & \{ t_i \bullet \\ & (t_i \in T_y \wedge type(t_i, ty_{ident}) \wedge \neg (t_j \in T_z \wedge type(t_j, ty_{ident})) \vee \\ & (t_i \in T_z \wedge type(t_i, ty_{ident}) \wedge \neg (t_j \in T_y \wedge type(t_j, ty_{ident}))) \} \\ & \cup \{ d_p \bullet d_p \in D_{y,k} \setminus D_{z,l} \vee d_p \in D_{z,l} \setminus D_{y,k} \} \end{aligned}$$

- PC8: Find all generated data objects that were annotated with key_1 and text $annot_A$ and which were derived from input data that was annotated with key key_2 and text $annot_B$:

$$\begin{aligned} & \{ d_p \bullet annotation(d_p, key_1, annot_A) \wedge \\ & flow(d_p, te_{i,k}, oport_{i,m}) \wedge \\ & flow(d_q, te_{j,k}, iport_{j,m}) \wedge \\ & annotation(d_q, key_2, annot_B) \} \end{aligned}$$

- PC9: Find all data objects of type ty_{ident} that have been annotated with key_1 and text $annot_A$ and return all existing annotations of the objects:

$$\begin{aligned} & \{ d_p \bullet type(d_p, ty_{ident}) \wedge \\ & annotation(d_p, key_1, annot_A) \} \\ & \cup \{ annotation(D, K, A) \bullet \\ & annotation(d_p, K, A) \} \end{aligned}$$

This general representation allows to map the GATiB related data provenance requirements presented in Section 5.2.2 to the set of First Provenance Challenge queries.

GATiB Prov. Query	Prov. Challenge Query
Q1	PC1
Q2	PC2, PC3
Q3	PC4, PC5, PC6
Q5	PC7
Q6	PC8, PC9

Table 3: Mapping of First Provenance Challenge Queries to GATiB Provenance Queries

After the First Provenance Challenge was completed and the results of all participants were received, a follow-up initiative, the Second Provenance Challenge ⁷, was defined. This initiative focused on interoperability issues of provenance data. The participants were invited to deliver the results of pre-defined provenance queries in a standardized format which make them processable for other competing systems. The results of the Second Provenance Challenge were collected and compared. A set of core modelling entities could be condensed from all submitted provenance models, which constituted the base components of the Open Provenance Model [64].

The provenance of data objects is defined in form of directed acyclic graphs that reflect past workflow executions generating the objects. The graph is enriched by annotation which represent additional information about workflow executions. The main requirements of the Open Provenance Model are defined as follows:

- Provenance data has to be shareable between different workflow management systems. Thus, the Open Provenance Model is rather considered as an interoperability exchange format than as a standard for

⁷Second Provenance Challenge <http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge>

storing provenance data. Interoperability is to be accomplished by introducing a set of controlled vocabulary, relationships and inference rules. The use of OPM as a provenance interoperability layer is illustrated in Figure 17. Systems may use proprietary models for storing provenance data. Though, they should supply converter tools for transforming their internal provenance representation into OPM compatible data. If provenance data from various application should be integrated in a central repository, each application converts its provenance data into an OPM representation which may be stored in a central repository.

- The specification of the provenance model should allow developers to design software components interacting with the model. Various tools such as visualization, reasoning or conversion software may be created.
- A set of inference rules is to be defined that is applicable to the provenance graph. For instance, for deriving the data lineage of a single object, appropriate inference rules are required.

The Open Provenance Model does not specify an internal representation format, workflow management systems do have to comply to. Instead, systems may retain their specialized provenance repositories in relational databases or semi-structured data. The OPM is specified formally without being bound to a certain representation language [63]. There are several representation of the model (e.g. XML Schema, OWL Ontology) which can be found at the OPM website ⁸.

The general principle of the Open Provenance Model is to represent how *things* reached their actual states, respectively which actions have been executed to transform these *things* into their current states. The notion of things embraces all kinds of objects, such as physical entities or data objects. However, in the following description we use the concept *thing* as a synonym for

⁸Open Provenance Model <http://openprovenance.org/>

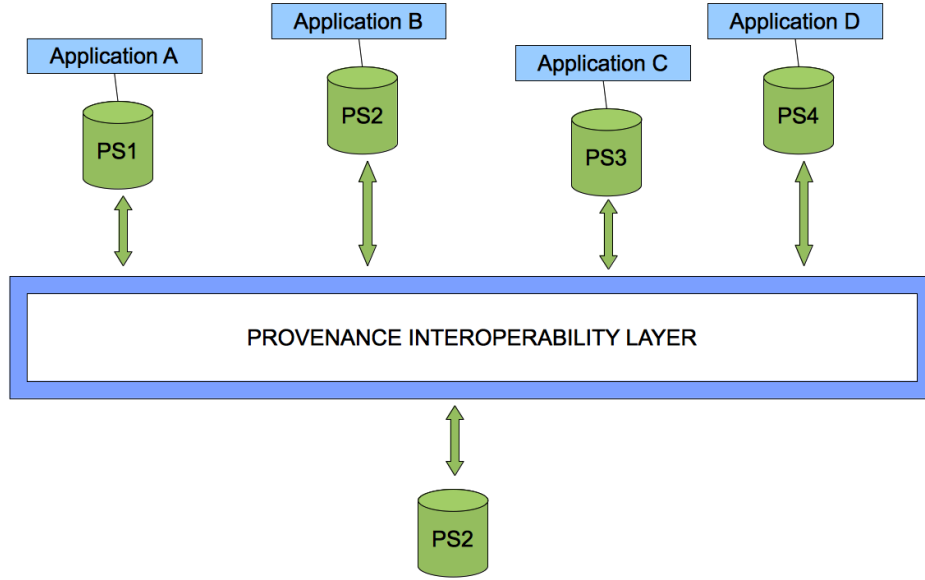


Figure 17: Open Provenance Model - Interoperability [65]

a data object. There are three types of basic entities in the Open Provenance Model: *artifact*, *process* and *agent*. Artifacts are things that are in a certain, immutable state. For instance, if a data file was processed by a service which appends a new line to file, the resulting file is an artifact of the original file. In this case, the artifact is a version of a data object. Generally, artifacts are all data objects that are involved in a workflow execution. A process is defined as an action or a series of actions that has been executed on or caused by artifacts. The execution of processes results in new artifacts. Agents act as enabler or controller of process executions. Agents may be individual persons or institutions.

The OPM represent causal dependencies between the entities artifact, process and agent in a dependency graph. The basic entities are modelled as nodes, while edges capture dependency relationships between the entities.

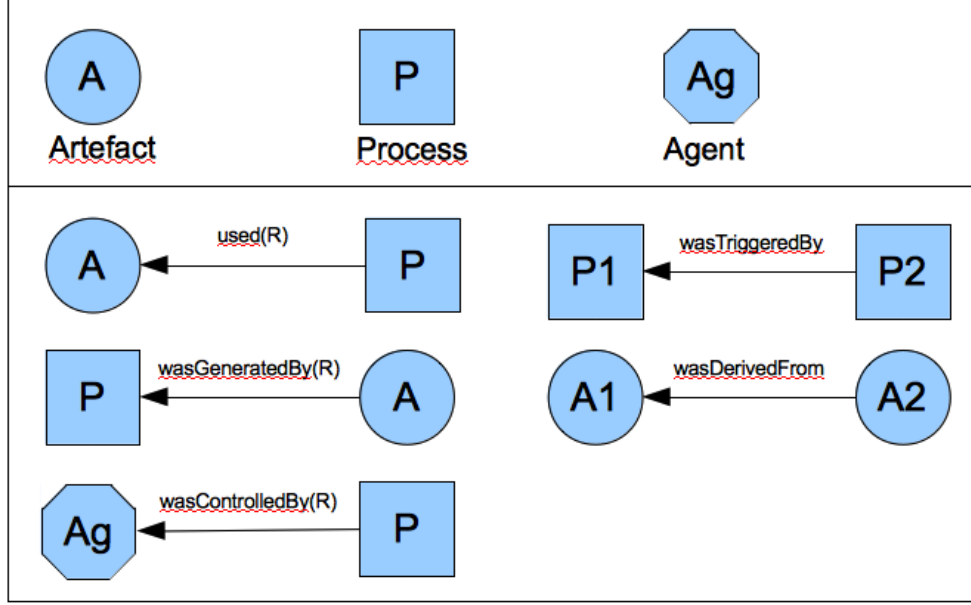


Figure 18: Open Provenance Model - Nodes and Dependencies [63]

There are various types of dependency relations which are differentiated by appropriate labels. The dependency relationships may be categorized in data derivation relationships and control relationships. The data derivation relationships *used(R)*, *wasGeneratedBy(R)*, *wasTriggeredBy*, *wasDerivedFrom* model data lineage of artifacts. The *wasControlledBy(R)* relationship is a control dependency and is not immediately related to data transformations. Figure 18 gives an overview of all dependency types of OPM. A *used(R)* dependency between an process *P* and an artifact *A* indicates that the artifact has to be available to allow the completion of the process. Thus, the artifact is an input data for the process. If a process requires several input data, multiple *used(R)* dependencies are created between the process and the corresponding artifacts. The *R* identifier of the *used* dependency specifies the role the artifact has in the dependency. For instance, a process performing a multi-criteria search in a text document requires two input data

files. One file contains a list of search criteria and the other one the text content that is to be searched. In this case, a *used(criteria)* dependency may be created for the first file and a *used(content)* dependency for the second. The *wasGeneratedBy(R)* dependency between process P and artifact A indicates that P has to be finished in order to generate the artifact A . This kind of relationship models output data of processes. Again, there is a role identifier for assigning a certain scope to the dependency relation. For instance, if a visualization service creates a boxplot output data, a dependency *wasGeneratedBy(boxplot)* could be created. Processes are controlled by agents which is expressed by *wasControlledBy(R)* dependencies. Role R of the dependency indicates in which context agent controlled the process, whereas a context may be a scientific user or an administrative user. As already mentioned, artifacts are generated by processes which use a set of input artifacts for creating new ones. In order to specify explicitly which output artifact depends on which input artifact, the *wasDerivedFrom* dependency is used. Consider a process that translates a set of nouns from a source into a destination language. Each translated noun of the destination language is an artifact which depends on exactly one noun of the source language. Thus, separate *wasDerivedFrom* dependencies may be created for all nouns capturing these fine-grained relationships. Dependencies between processes may be modelled explicitly by *wasTriggeredBy* dependencies. In some cases, artifacts that were used as input data for a process are unknown. When the input data of process $P2$ was generated by process $P1$, then $P2$ is said to be triggered by $P1$ and modelled by the dependency $P2$ *wasTriggeredBy* $P1$. This situation may occur, if historical provenance data which only contain an execution log of all involved processes is analyzed but no information about artifacts is available.

5.4 Implementation of the Gene Expression Analysis Workflow

The gene expression analysis was implemented as a scientific workflow composed of a set of tasks with predefined input and output parameters. Starting with a selection of medical records and associated gene expression profiles, the workflow runs through various transformation, mapping and filtering steps and finally results in the analysis and plotting of the results of a statistical analysis. The complete workflow is modelled in a business process model notation (BPMN) in Figure 19. In the following, a detailed task description

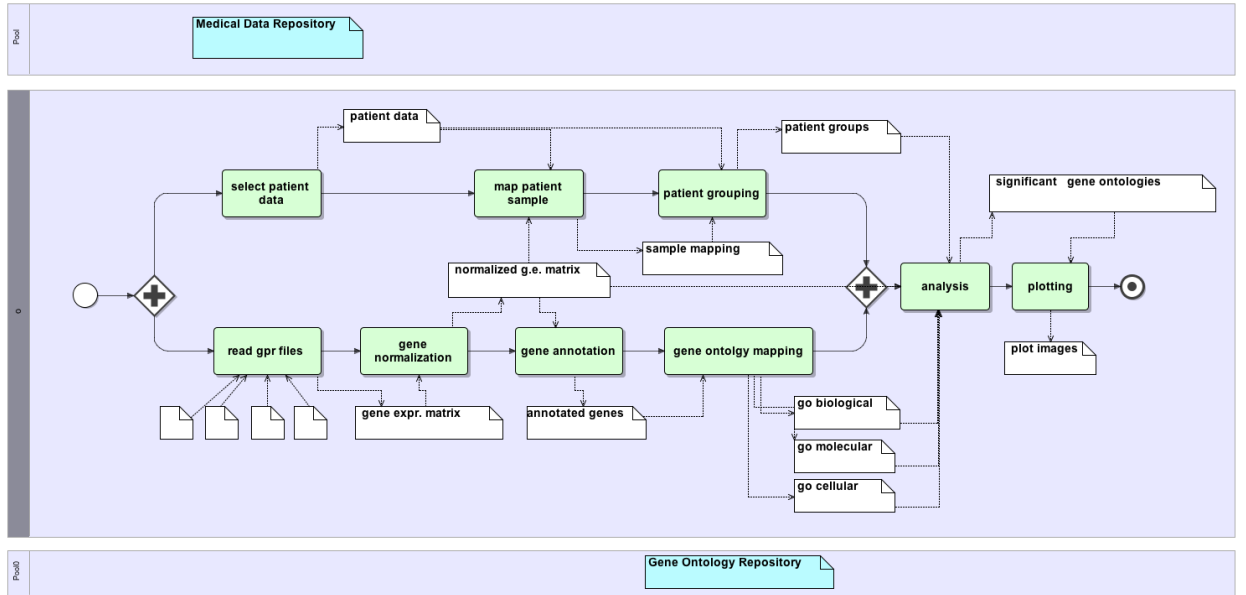


Figure 19: BPMN Model of Gene Expression Analysis Workflow

is given.

- **Select Patient Data:** The medical data repository is a database containing medical data of patients. The record set is the result of integrating diagnostic data from the pathology (e.g. tumor staging, ICD-O3 coded diagnoses), data assessed from the surgery and therapeutic data from

data sources of the oncology. Selection is limited to cases, for which biological material has been collected and gene expression profiles were created. The output of this task is a record set that is used for defining filter and grouping criteria in the subsequent analysis task. Every entry in the record set is uniquely identified by a *patient.id*.

- Read GPR Files: Gene expression profiles are provided as GenePix GPR files, which are tab-delimited files containing the measured signal intensities of a microarray experiment. The content of a GPR file is divided into an informative header section, listing metadata about the deployed scanning technology and into a GPR data section which contains the measured genetic activities. Each row in the GPR data section corresponds to the measured intensity of a certain gene. Every GPR file is linked with a tissue sample by an appropriate file name and may be thereby mapped to medical records in a later step. The result of the GPR reading task is a gene expression matrix of n rows and m columns, whereas n is the number of analyzed genes and m the number of selected gene expression profiles. By choosing a medical data set, the GPR reading task may be started in parallel as both task are based on the same selection criteria. In order to generate valid analysis results, only gene expression profiles that were generated on the same microarray platform with identical assay protocols can be combined in a gene expression matrix.
- Gene Normalization: The generated gene expression matrix is passed to the normalization task which is responsible for removing systematic errors resulting from the microarray experiment. The measured signal intensities are influenced by the efficiency of labelling cy5 and cy3 dyes and by various image scanning properties. We used the normalization libraries provided by the marray package of the statistical software

Bioconductor⁹. The resulting normalized gene expression matrix serves as input data for the tasks map patient samples, gene annotation and analysis.

- Map Patient Samples: Every gene expression profile in the gene expression matrix is identified by a unique *slide_id*. Each slide id is associated with a human tissue sample which in turn can be mapped to a patient record. Depending on the set of selected medical records, a mapping table is created, consisting of (*slide_id*, *patient_id*) entries.
- Gene Annotation: Microarray chip producers notate genes with vendor-specific identifiers, which are used in GPR files and consequently in the generated gene expression matrix. Gene annotation is the process of mapping these vendor-specific identifiers to gene identifiers that are used in public gene databases. For instance, unique gene symbols and names were assigned by the HUGO Gene Nomenclature Committee (HGNC)¹⁰ which is a subgroup of the Human Genome Organization. Microarray vendors provide gene annotation files for translating chip-specific identifiers (*chip_gene_id*) to publicly-known notations (*public_gene_id*). The output of the annotation task is a list of translations consisting of (*chip_gene_id*, *public_gene_id*) entries.
- Gene Ontology Mapping: With the help of Gene Ontologies, genetic functions may be assigned to genes. As already described in Section 5.2, the Gene Ontology Consortium maintains the definition and assignment of gene functions in the Gene Ontology Database. Gene Ontologies are identified by unique GO terms (*GO_id*). New versions of the Gene Ontology are released on a daily basis. Based on the gene annotation list, gene functions of the genes used in the microarray

⁹Exploratory analysis for two-color spotted microarray data <http://www.bioconductor.org/packages/2.8/bioc/html/marray.html>

¹⁰The HUGO Gene Nomenclature Committee <http://www.genenames.org/>

experiment can be determined. For each type of gene function: biological processes, cellular components and molecular functions, a mapping file is generated, containing mappings of GO terms to chip genes ($GO_id \mapsto chip_gene_id_1, \dots, chip_gene_id_n$).

- Patient Grouping: For hypothesis testing, the medical record set is divided into subsets based on filter criteria defined on data columns. For instance, long-term survivors of a certain cancer type are compared to short time survivors. Two groups of patients are defined, whereas both groups must have a sufficient number of group elements for the subsequent statistical analysis. This task takes the sample mapping list as an input and use the grouping criteria and the slide mapping information for generating a grouping file with $(slide_id, group_id)$ entries.
- Analysis: Gene expression analysis is conducted by one of the following analysis methods 'Global Test' [38] and 'Global Ancova' [44]. Both analysis techniques have been implemented in Bioconductor packages ¹¹, ¹². The task takes the normalized gene expression matrix, the three Gene Ontology mapping files and the grouping data of the patient grouping task as input data. The aim of the analysis is to identify sets of genes that are on average more associated with a grouping variable than expected. The assignment of genes to gene sets is accomplished by the Gene Ontology mapping. The identified sets of genes may be positively associated (up-regulated) or negatively associated (down-regulated) with the grouping variable. The null hypothesis to be tested is that there are no differences in the expression of gene sets in the compared groups. The null hypothesis may be rejected, if the calculated p-value is less than a predefined significance level, which is

¹¹Global Ancova: <http://www.bioconductor.org/packages/2.5/bioc/html/GlobalAncova.html>

¹²Global Test <http://www.bioconductor.org/packages/2.0/bioc/html/globaltest.html>

typically 0.05 or 0.01. A statistical test is performed for each gene set allowing to determine significant associations between sets of genes and grouping variable separately. The output of the task is a list of Gene Ontologies, that are proven to be over- or under-expressed in one of the compared groups. The returned Gene Ontology list is sorted by the calculated significance level. The list may be filtered by excluding entries exceeding the significance level limit.

- **Plotting:** The results of gene expression analysis are visualized by the plotting task which takes the list of significant Gene Ontologies as an input. A plot is created for each Gene Ontology, showing the relative influence of single genes on the outcome of the statistical test. For all genes belonging to a gene set of a Gene Ontology, an influence factor is calculated and plotted in a bar chart. Therefore, it is possible to detect and evaluate the influence of single genes contributing to a significant gene set.

Wizard-based Execution

In order to support user-controlled execution of the gene expression analysis workflow, a distributed information system was developed which is based on web applications, web services and various file and database resources. The aim of the system was to provide researchers a simple web user interface which allows to control and execute tasks of the analysis workflow in a wizard-based execution mode. The system architecture is illustrated in Figure 20.

The multi-tier architecture consists of a central coordination server, web-based clients, a set of experiment servers and data repositories. The web-based workbench for researchers may be opened by various web browsers. The user interface has been successfully tested with Microsoft Internet Explorer, Mozilla Firefox and Opera. Web clients communicate with the coordination server by sending HTTP requests and receiving HTTP responses. The coordination server has been realized on an Apache Tomcat web server, which has an embedded container for maintaining and executing Java Servlets. The

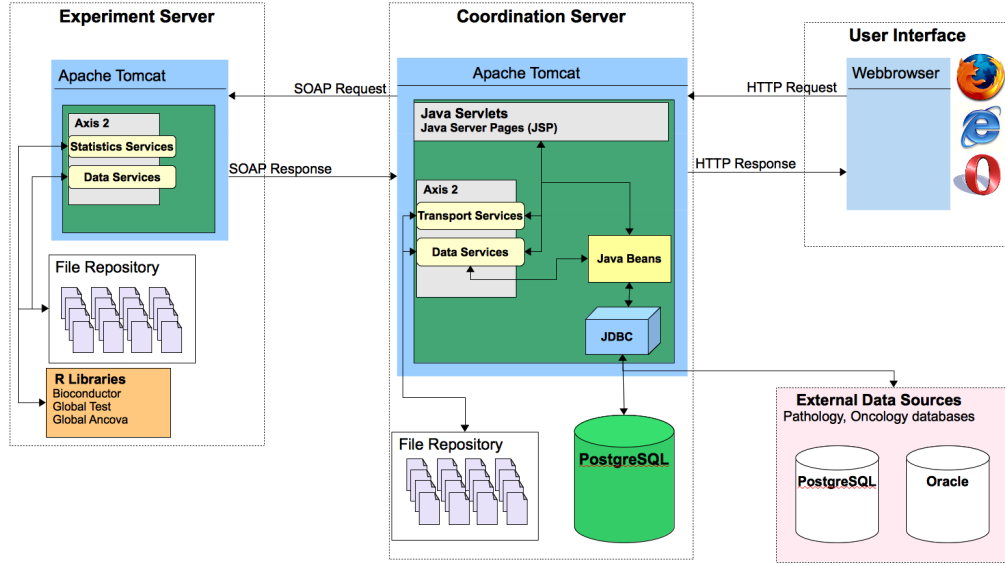


Figure 20: GATiB Scientific Workflow System Architecture

coordination server is responsible for generating HTML user interfaces depending on URLs and parameters of received HTTP requests. In order to create dynamic content in HTML responses, JavaServer Pages (JSP) and JavaServer Pages Standard Tag Library (JSTL) are used. Further, JavaBeans are deployed for encapsulating data in data objects and embed them in generated result pages. Access to data sources is accomplished by data services which provide access methods for an internal file repository and for data of relational databases. A JDBC bridge is used to access data from PostgreSQL and Oracle database systems. Generally, experiment-related data is located on the coordination server, that stores intermediate and final results of the workflow in the local file system and in a PostgreSQL database. Interfaces to external databases such as the oncology and pathology data repository allow to integrate and update patient records.

The coordination server reallocates data intensive tasks to experiment servers which have large computational resources and are capable of exe-

cuting tasks on demand. Both coordination and experiment servers deploy an Apache Axis2 web service engine and communicate by SOAP messages. Tasks are assigned to experiment servers by issuing web service calls containing a task identifier and all necessary parameters. Data files are transported by SOAP with attachments and Secure Copy (SCP) commands. If a task is to be executed on an execution server, the coordination server prepares all required data files and transports them by SCP commands to the destination system. Then a web service call is packed into a SOAP request and sent to the experiment service. Since computational-intensive task such as statistical analysis may last several minutes, an asynchronous communication protocol is required. Therefore, a callback is sent from the experiment server to the coordination server after the task has been completed. At the side of the experiment server, a callback handler waits for an incoming callbacks and resumes processing after it has been received. The file repository at the coordination server maintains data files of current user sessions. Each user session has a dedicated folder containing data files required for the current workflow execution of the user. A unique session id is used for labelling session folders. Data files are created by extracting relevant data from the local PostgreSQL database. For instance, the gene expression matrix and Gene Ontology mapping files are generated on demand and staged in the session folder for further processing. Data files remain in the session folder as long as the user session is active and are deleted after user logout. If single tasks are to be repeated, data files may be reaccessed without additional database access. The file repository at the experiment server is also structured in session-related folders and are filled with data required for the current task. While session folders at the coordination server contain data files of all user sessions and tasks, session folders at experiment servers store data files of single tasks. After a task has been successfully completed, result files are transported to the coordination server and the session folder is deleted. Statistical analyses are executed at experiment servers by calling R

library functions. The R library Bioconductor provides a set of analysis and visualization methods for microarray experiments including implementations of the GlobalTest and GlobalAncova approaches. Calls to R libraries have been realized by implementing R functions which load files from the session folders into the R execution environment, perform internal data transformation and filtering operations and call analysis and plot libraries. Parameters received by web service calls are handed over to these R utility functions. After completion of the R functions, results are packed into a tar file and are embedded into a SOAP response message which is sent back to the coordination server.

In the following the main user interface components are presented. Grouping of patients is supported by the user interface presented in Figure 21. The user interface is split in two parts, whereas each part allows to specify a group of patients based on selection criteria on medical data. Users may switch between group definitions by selecting the group tabs at the top of the interface. The user interface is created dynamically based on the previously selected data set.

Group medical data
Go Logout
Gene Group Analysis
GEN-AU
GENOME RESEARCH IN AUSTRIA

Group 1
Group 2
Save
Reset

Year_of_Diagnosis	Gender
pT	pN
pM	Grading
Number_of_asported_lymphnodes	Number_of_positive_lyphnodes
Estrogen_receptor_IHC	Progesteron_receptor_IHC
ER_(fg)	PR_(fg)
Amplification_date_agilent_kit_AC	Autopsy_
Diagnosis	Kind_of_border
Grading_Tubulus_Formation	Nuclear_Grading
Mitotic_Grading	In_situ_Components
Grading_is	Fibroblast_contentent
Fibrose_Grade	Elastase_grade
Lymphocyte-infiltration	Epithel_Percentage
Presence_of_other_cryoslices	Localization_primary_tumor
Menopausal_status_at_First_Tumor_Diagnosis	Surgical_M
Her2neu_DAKO	Neoadjuvant_PCT_with_Anthracyclin
Neoadjuvant_PCT_without_Anthracyclin	Postoperative_Radiation
Postop_adjuvant_Hormonotherapy	Adjuvant_PCT_with_Anthracyclin
Adjuvant_PCT_without_Anthracyclin	Lokalisation_der_Erstmetastase
Status	Reason_of_death
DFS_(Months)	OS_(Months)

A001	A002	A003	A004	A005	A006	A007	A008	A009	A010	A011	A012	A013	A014	A015	A016	A017	A018	A019	A020	A021	A022	A023	A024	A025	A026	A027
A028	A030	A031	A032	A033	A035	A036	A037	A038	A039	A040	A042	A043	A044	A045	A046	A047	A049	A050	A051	A052	A053	A054	A056			
A057	A058	A059	A061	A062	A063	A064	A066	A067	A068	A069	A070	A074	A075	A077	A078	A079	A080	A081	A082	A083	A084	A085	A087			
A088	A090	A093	A094	A095	A096	A097	A099	A101	A102	A103	A104	A105	A106	A107	A108	A11	B012	B013	B014	B015	B016	B017	B019			
B022	B023	B024	B025	B026	B027	B028	B029	B030	B031	B032	B033	B034	B035	B036	B037	B038	B039	B040	B041	B042	B043	B044	B045			
B046	B047	B048	B049	B050	B051	B052	B053	B054	B055	B056	B057	B058	B059	B060	B061	B062	B063	B064	B065	B066	B068	B069	B070			
B071	B072	B073	B074	B075	B076	B078	B079	B080	B081	B082	B083	B084	B085	B086	B087	B088	B089	B091	B092	B093	B094	B095	B096			
B097	B098	B099	B100	B101	B102	B103	B104	B105	B106	B108																

Figure 21: Gene Expression Workflow - Patient Grouping User Interface

For instance, a mamma carcinoma data set of 179 patients is presented in Figure 21. The example mamma carcinoma data set has been annotated with 42 attributes, including diagnostic attributes such as tumor staging (pT, pM, pN), progesteron and estrogen receptors and survival data (disease free and overall survival). For each attribute, the set of possible values is extracted and filled into a drop-down list. By choosing values of the drop-down list, a new filter criteria for an attribute is defined. Filter criteria for attributes are connected by logical AND operators.

The associated gene expression profiles are illustrated in the bottom part of the user interface. Gene expressions are labelled according to their slide names in microarray experiments. Whenever a new filter criteria is defined, all matching gene expression profiles are marked with coloured ellipses. After group definition has been accomplished, the analysis parameters may be specified. Before the analysis step of the wizard is displayed, the gene expression profiles of both groups are extracted from the PostgreSQL database and combined in a gene expression matrix file. Additionally, a mapping file assigning slide names to groups is generated. Both files are stored in the session folder of the user.

Figure 22 depicts the user interface for specifying the analysis parameters. The grouping criteria is summarized in the top section of the user interface. Users may choose one of the analysis methods GlobalTest or Global Ancova. Further, the gene group of interest may be selected: GO:biological_processes, GO:cellular_components or GO:molecular_functions. Depending on the selected gene group type, a mapping file assigning corresponding GO terms to genes of the microarray experiment is created. The last parameter, top gene groups, specifies the number of most significant gene groups, that should be displayed in the result set. The result of the gene analysis is a list of significant gene groups that is sorted by their significance levels.

In Figure 23 an example of an analysis result is given. In the first column of the list the Gene Ontology term of the gene group is displayed. Each GO

Analysis Go Logout Gene Group Analysis GEN-AU
GENOME RESEARCH IN AUSTRIA

Group 1 criteria: pT=1;1(m);1A;1B;1C;1C-2
Group 2 criteria: pT=2

Analysis method: Global Test
Gene groups: GO:biological_processes
Number of top gene groups: 50

Start analysis
<<< Go back

Figure 22: Gene Expression Workflow - Gene Group Analysis User Interface

term is linked with the corresponding entry at the Gene Ontology web page. In the second column, a short description of the term is given. GO terms have a variable number of associated genes. Some terms only link to single genes while others link to several hundreds. The result list comprises, how many genes are associated with a certain GO term, and how many of the genes have been used in the analysis. In most cases, all known genes participate in the analysis, as microarray chips include nearly exhaustive sets of genes. In the significance column, the significance level of the test is presented. Significance levels below 0.05 indicate that there is a statistically relevant difference in the gene expressions of a gene group in the compared patient groups. The details of the statistical test of a gene group may be assessed by clicking on the icon in the details column. In the detail view, the influence of single genes of the gene group on the analysis result is illustrated in a plot. Further, a CSV file containing single gene influences may be downloaded. The entire gene group list may be exported into an Excel file and saved in the database.

Generate plots Export results Save experiment					
GO category	Description	# genes	# tested genes	Significance	Details
GO:0006516	glycoprotein catabolic process	9	9	0.000138	Details
GO:0006349	genetic imprinting	13	13	0.000194	Details
GO:0000132	establishment of mitotic spindle orientation	1	1	0.000305	Details
GO:0006259	DNA metabolic process	24	24	0.000336	Details
GO:0051154	negative regulation of striated muscle cell differentiation	2	2	0.000448	Details
GO:0032235	negative regulation of calcium ion transport via store-operated calcium channel	1	1	0.000467	Details
GO:0009441	glycolate metabolic process	2	2	0.000489	Details
GO:0035264	multicellular organism growth	3	3	0.000527	Details
GO:0016048	detection of temperature stimulus	1	1	0.000613	Details
GO:0021940	positive regulation of granule cell precursor proliferation	5	5	0.000635	Details
GO:0042488	positive regulation of odontogenesis of dentine-containing tooth	3	3	0.000702	Details
GO:0021675	nerve development	4	4	0.000848	Details
GO:0048009	insulin-like growth factor receptor signaling pathway	10	10	0.001209	Details
GO:0006281	DNA repair	171	171	0.001235	Details
GO:0010001	glial cell differentiation	11	11	0.001253	Details
GO:0002318	myeloid progenitor cell differentiation	1	1	0.001353	Details
GO:0017055	negative regulation of transcriptional preinitiation complex assembly	2	2	0.001363	Details
GO:0000079	regulation of cyclin-dependent protein kinase activity	46	46	0.001493	Details
GO:0045075	regulation of interleukin-12 biosynthetic process	1	1	0.001509	Details
GO:0048515	spermatid differentiation	1	1	0.001509	Details
GO:0006977	DNA damage response, signal transduction by p53 class mediator resulting in cell cycle arrest	8	8	0.001743	Details
GO:0006479	protein amino acid methylation	9	9	0.001792	Details
GO:0009081	branched chain family amino acid metabolic process	2	2	0.002062	Details
GO:0009082	branched chain family amino acid biosynthetic process	2	2	0.002062	Details

Figure 23: Gene Expression Workflow - Gene Group Analysis Results User Interface

Examples of influence plots are given in Figures 24 and 25. In the left part of Figure 24, the influence plot for GO term GO:0006418 indicates a dominating influence of three genes which are overexpressed in the patient group with tumor staging pT=2. The remaining six genes do not show a markable over- or underexpression. In the influence plot of GO term GO:0048468, one of twelve genes has a dominating influence on the significance level. Figure 25 illustrates the influence plots of larger gene groups. While gene identifiers may be represented in the influence plot of GO term GO:0007059, the number of genes in gene group term GO:0007067 is too large for a compact

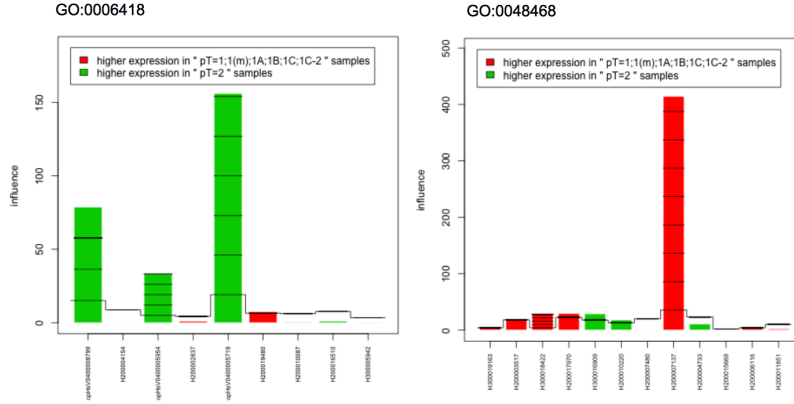


Figure 24: Gene Expression Workflow - Gene Group Analysis Plots

representation. Therefore, gene identifiers are faded in as tooltips.

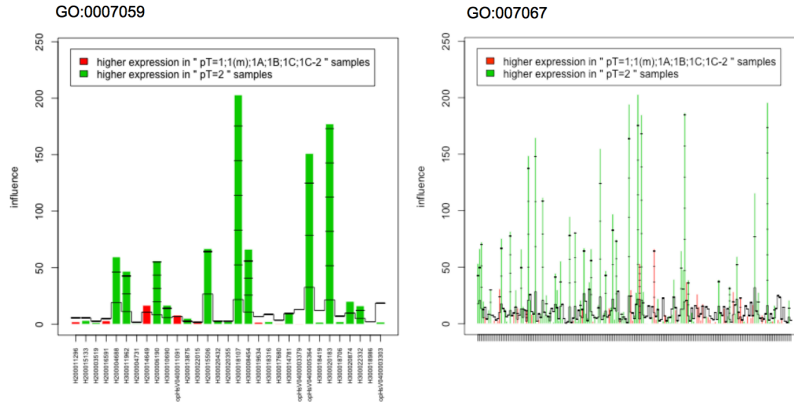


Figure 25: Gene Expression Workflow - Gene Group Analysis Plots

All necessary data for conducting gene expression analyses is stored in an experiment database hosted at a PostgreSQL database server. The main relations of the database schema of the experiment database are given in the class diagram illustrated in Figure 26. Medical data is extracted from routine clinical information systems and stored in relations patient_data, medical_data and medical_attribute. Patient_data solely contains a unique patient identi-

fier without any person-related data. Patient data is represented by assigning medical attributes and attribute values to patient entries. For instance, if patient P_A has an age of 55 and suffers from liver cancer, the two medical attributes *age* (identifier ma_1) and *cancer_type* (identifier ma_2) are linked by the following *medical_data* entries: *medical_data*(1, ma_1 , P_A , 55, *pathology*) and *medical_data*(2, ma_2 , P_A , *liver cancer*, *pathology*).

The *medical_data* relation stores values of medical attributes and the source from which the information was extracted. Medical attributes have a unique id, a name and an associated attribute type indicating one of the following data types: String, Date, integer, float, double, text or binary data. Depending on the specified data type, different rendering of data values in the user interfaces is supported. Further, appropriate selection and filter criteria may be defined on each attribute - e.g. interval selection for numeric types, regular expression selection for character types. Datasets are subsets of patient data that are used in a certain context. For instance, a collection of medical data of mamma carcinoma patients is established in a prospective cohort study. During observation of the patient group, different data such as tumor characteristics, recurrence of tumor, lifestyle and treatment data is assessed in various medical attributes. Different medical studies may base on the same patient cohort, each focusing on specific aspects. If a study investigates the effects of lifestyle habits on disease outcome, a dataset consisting of disease-specific and lifestyle data may be defined. Relation dataset stores information about selected patient data in attributes name and description. A dataset consists of a set of medical data entries which is represented by relation *dataset_mapping*. Microarray experiments are conducted on selected samples of patients. For each microarray experiment, the platform (e.g. Affymetrix) on which it was executed and a detailed experiment protocol are stored in corresponding attributes. A microarray experiment consists of a set of microarray slides whereas each slide is linked to a patient data entry. Generated gene expression profiles are stored in

relation `gene_expression`. The relation specifies which $\log_2$ gene expression values were assessed for a certain oligonucleotide probe on a specific slide. Oligonucleotide probes have unique `oligo_id` identifiers. Attributes `row` and `column` define the position of probes on the microarray slide and attribute `value` stores the gene expression value. Gene annotation data are provided by microarray chip vendors. Relation `gene_annotation_data` maps oligonucleotide probes to sequence and genetic data. Each oligonucleotide has an associated gene name - e.g. the gene symbol defined by the HUGO Gene Nomenclature Committee¹³. Further, annotation data concerning exons, transcripts and chromosomes is available. Another type of annotation is accomplished by the `refseq` relation which maps oligonucleotides to reference sequences (RefSeq) entries. The RefSeq database [78] is a public repository of protein and nucleotide sequences which is maintained and released by the National Center for Biotechnology Information (NCBI). The RefSeq repository is an international standard for gene annotation, which is wide spread in genomic research. We use RefSeq entries as annotations for results of gene expression analysis. For each identified significant gene group, all related RefSeq entries are extracted and displayed. Gene Ontology terms are integrated by downloading and importing database dumps from the Gene Ontology repository¹⁴. The Gene Ontology database consists of 36 relations. However, only a subset of four relations is required for the gene expression analysis: `go_term`, `go_term2term`, `term_dbxref` and `dbxref`. Gene Ontology terms are represented by the `go_term` relation. The unique Gene Ontology identifier is specified in attribute `acc`, for instance identifier GO:0006925 defines the apoptosis of inflammatory cells. Attribute `term_type` is used for specifying one of the ontology domains molecular function, biological process or cellular component as well as Gene Ontology specific types such as relationship,

¹³The HUGO Gene Nomenclature Committee (HGNC) <http://www.genenames.org/>

¹⁴Gene Ontology repository <http://www.geneontology.org/GO.downloads.ontology.shtml>

subset, synonym type, synonym scope and universal. Obsolete Gene Ontology terms are marked by the `is_obsolete` attribute. The root node of ontology hierarchy is identified by value 1 in attribute `is_root`. Relationships between ontology terms are represented by entries in the `go_term2term` relation. A relationship of type `relationship_type_id` is defined between child term `term1_id` and parent term `term2_id`. Generally, hierarchical dependencies are modelled by `is_a` relations and part-whole relationships are represented by `part_of` relations. Additionally, Gene Ontology terms are cross-referenced with other genetic annotations from external resources by entries in relations `dbxref` and `term_dbxref`. For instance, genetic pathway annotations are supplied by the Kyoto Encyclopaedia of Genes and Genomes (KEGG) ¹⁵. Attribute `xref_dbname` specifies the name of the external data source and attribute `xref_key` and the key in the linked source. Gene Ontology terms are linked to oligonucleotides of microarray experiments by relations `go_molecular`, `go_cell` and `go_biological` which are mapping tables for each ontology domain.

Results of completed gene expression analyses are stored in relation `virtual_experiment`. On the one hand, user-defined information such as `experiment_name` and `description` are stored. On the other hand, all relevant input and output parameters of the experiment are assessed. Attribute `method` specifies, which analysis algorithm has been executed. Attributes `group1_sql_criteria` and `group2_sql_criteria` store the SQL filter criteria for defining the compared groups of patients. `Gene_group` specifies the used ontology domain, and `group_limit` the user-defined limit on significant gene groups. All generated output files of an experiment (plots, csv files) are saved as a tar file in attribute `result_data`. Further, the current state of the R workbench is exported in a `.RData` image file and saved in attribute `r_data`. Virtual experiment data is used for loading completed gene expression analyses, viewing the results and reexecute analyses with slightly modified parameters. A simple user management has been integrated, allowing to

¹⁵KEGG Pathway Database: <http://www.genome.jp/kegg/pathway.html>

authenticate users and keep track on their executed experiments.

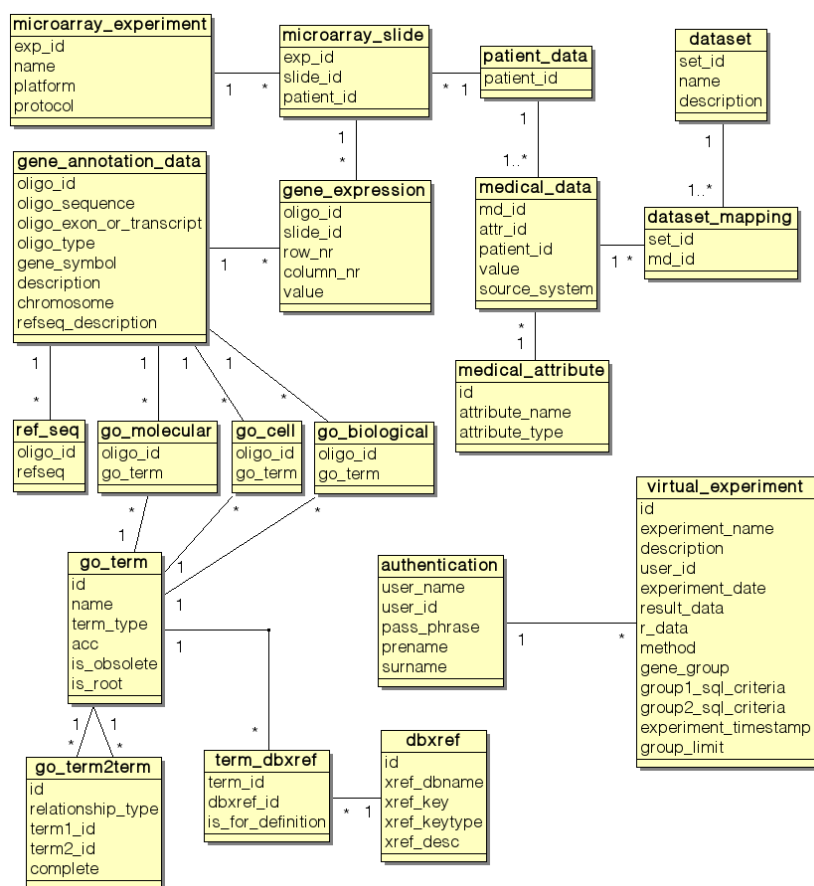


Figure 26: Microarray Experiment Database Schema

5.5 Integration of Scientific Workflow Execution in GAMECS-CSCW

The user-interactive execution of the above-mentioned gene expression analysis allows to iteratively specify execution parameters and immediately monitor intermediate results. Further, the process of selecting and grouping patient data is supported by graphical representations of patient cohorts. The visual feedback of patient grouping is particularly reasonable as a part of data exploration. Medical researcher become aware of the characteristics of the underlying data set and of its grouping possibilities. Additionally, a batch execution mode for workflows was required allowing to preselect input data and parameters for all tasks and execute the entire workflow without further user intervention. Batch execution of workflows is deployed in cases, where all required input data and parameters are known and no manual interaction is required. Moreover, it is possible to define series of workflow executions and view their results at a later date.

The main idea was to enhance the capabilities of the GAMECS-CSCW system (presented in Section 4) and integrate workflow-execution functionality into the system. Thus, the resulting system combines the strengths of both types of information systems: it allows to collaboratively contextualize data resources in knowledge spaces and enrich them with semantic annotations, and it supports the definition and enactment of scientific workflows based on these resources. A workflow definition and a workflow execution component had to be designed for executing general scientific workflows. In the following, the main capabilities of the extended CSCW system are listed. The features of the system have been published at the DEXA 2010 [96].

Semantic annotation: We encourage collaboration by creating virtual knowledge spaces that are filled with data resources and services. A knowledge space is a predefined context or container allowing for structuring shared data and cooperatively perform operations. A knowledge space may be created for a particular project or study. As the cooperation proceeds, new

data may be added or deduced by applying services. Finally, if the project is finished, the context data is retained. That enables both documentation of the collaboration and resuming the cooperation at a later time. Projects and studies frequently overlap or depend on each other. That is, data and services may be reused in similar contexts. Consider a set of diseased human tissues that is used in two parallel medical studies. While the first study uses gene expression analysis of microarrays, the second one uses polymerase chain reaction in order to detect genes relevant for the disease. Both studies can benefit from each other if they are aware of each other, are able to exchange generated data, and have knowledge about how the results were created. We support various ways of annotating contexts semantically. Firstly, a categorized tag repository allows for assignment of categories (and subcategories) that characterize contexts. Further, free text annotations containing textual descriptions may be assigned to contexts. Finally, the content of contexts is searchable. Thus, it is possible to look for all contexts, where the same set (or subset) of data resources or services is used [96].

Process Definition: Processes are assigned to collaborative knowledge spaces and may be executed by members of the corresponding knowledge space. The deployment of process templates has various objectives: If a certain procedure has been approved as efficient for reaching a certain goal, it may be reused in similar contexts. Well established procedures can be implemented as standardized processes. For instance, gene expression data created by microarray experiments have to be normalized and standardized before an analysis of the gene activities may begin. Therefore, the measured gene expression values are transformed appropriately eliminating systematic bias of hybridization experiments. Thus, adhering to certain transformation techniques, gene expression experiments become comparable to each other. Dedicated processes may be provided to untrained persons for improving the learning process. A sandbox like testing environment enables to trace, reexecute and modify processes and immediately view the created results.

These training processes allow a deeper understanding of complex procedures and a steeper learning curve for beginners [96].

Process Traceability: We designed appropriate data structures for logging the execution of web service and store input parameters and generated data. The execution logs allow a detailed view on how data resources were created. For each generated data source, the generating service as well as input parameters are detectable. The transparency of processes is useful for documentation purposes, for instance, for the materials and methods section of publications [96].

Process Repeatability: Since the execution sequence of services is stored as well as the parameter settings, it is possible to repeat the execution of processes or even subprocesses. Verification of analysis results may be accomplished as well as process reexecution with slightly modified parameters. Consider a time-consuming service that is frequently executed with equal parameter values. If the results generated by the service are stored, they may be reused in future requests [96].

The system architecture of the adapted GAMECS-CSCW system is presented in the following Figure 27. The original CSCW system is based on a JBoss Application Server which provides full Enterprise JavaBeans functionality, including support of transactions, scalability of applications and integration of distributed services. The GAMECS-CSCW server offers collaborative services such as data sharing and data annotation in virtual knowledge spaces and communication services, e.g. integration of instant messaging and mail. A web-based user interface fulfilling the collaborative demands of medical research in context of biobanks has been implemented. The CSCW User Interface allows to contextualize biomedical data according to specific research questions or medical studies. Further, data enrichment by annotations, tags and discussion boards is supported. Relevant data from biomedical resources (clinical patient data, gene expressions, research data) has been extracted from external sources and materialized in the experiment

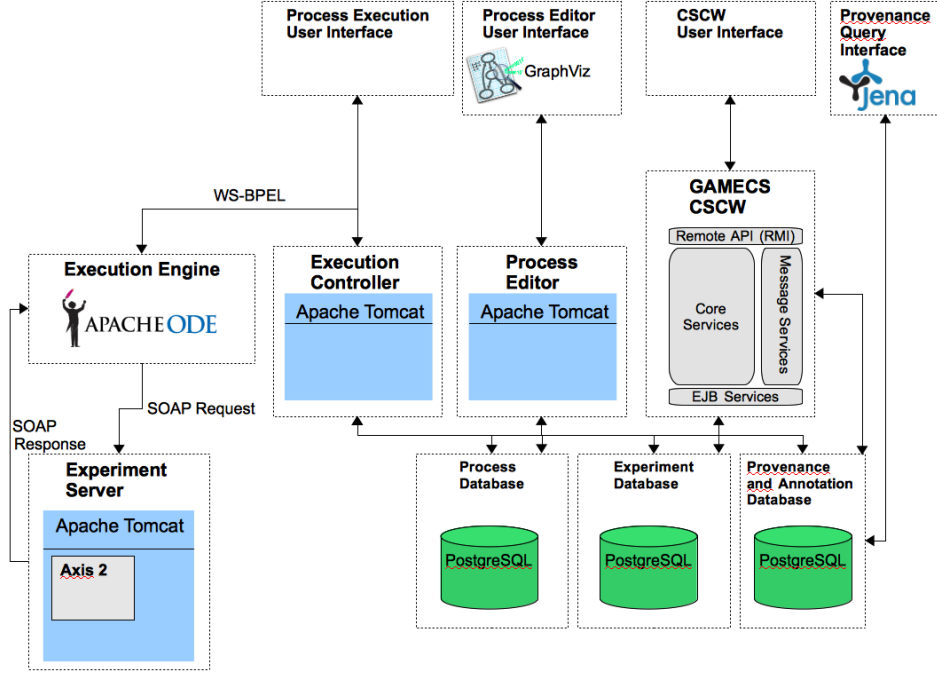


Figure 27: Adapted System Architecture of GAMECS-CSCW

database. In order to integrate scientific workflow functionality, several new components have been developed. The process editor allows the creation and management of process templates. The process editor user interface allows medical users to generate process definitions by arranging services in acyclic graphs. These process templates are persistently stored in a separate process database. The Execution Controller is the runtime component for execution of process templates. Process definitions are loaded from the process database, instantiated as new processes and filled with data from the experiment database and user-defined parameter values. Processes consist of sequences of web service calls. The execution of web services is coordinated by the web service orchestration engine Apache ODE. The Execution Controller transforms the internal process representation into a valid WS-BPEL

business process specification which is sent to the Execution Engine Apache ODE. By executing the received WS-BPEL specification, Apache ODE issues web service calls, handles parameter value passing and returns the generated results to the Execution Controller. Web services are executed on the web service stack Axis2 on separate experiment servers. Provenance data of process executions is recorded in the Provenance Database. The provenance model was designed as an ontology by using the knowledge representation language OWL-DL. The Jena framework is deployed for storing provenance data in a relational database. Provenance queries may be formulated by SPARQL expressions. However, no graphical user interface has been realized for defining provenance queries. A command line Java application delivers result sets for provenance queries written in SPARQL. In the following two subchapters 5.5.1 and 5.5.2, details about the new components for process definition and process execution are presented.

5.5.1 Workflow Definition Component - Process Editor

The process editor is a graphical user interface for designing and editing execution templates (processes) of scientific workflows. The implementation of the process editor was done by Carina Christ in context of her bachelor thesis [16]. The process editor has been realized as a web application consisting of three parts: a web user interface using Java Seam, a Tomcat application server and a PostgreSQL database schema. Processes are defined in the web user interface by arranging services as tasks in a directed acyclic graph $G = (N, E)$. The node set N is the set of tasks and the set of edges E model transitions between the tasks. Users may select services from a service repository (e.g. data access service or statistical service) and assign them to task nodes. The acyclic graph is visualized using the open source GraphViz¹⁶ library. GraphViz is capable of generating graphs based on the descriptive Dot language. Starting from an initial task node, additional task nodes may

¹⁶GraphViz www.graphviz.org

be inserted incrementally, whereas parallel task execution sequences can be realized by splits and joins. Data flow between tasks is captured by mapping output parameters to input parameters of the corresponding services. After a process has been completely specified, it may be stored persistently in the process database. Further, existing processes may be loaded, modified and saved as new templates.

In Figure 28, the user interface of the process editor is displayed. In the upper part of the interface, the current process definition is represented as a GraphViz graph. The process definition may be modified by using the interaction tabs *edit process*, *edit node*, *edit edge*, *add node* and *add edge*. Selected task nodes can be relabelled or removed and new task nodes or edges may integrated. If an edge is defined between two task nodes, data flow has to be specified from the preceding service to the subsequent service. That is, output parameters of the preceding service are mapped to input parameters of the subsequent service. However, parameter mapping is restricted by type compatibility of parameter types. If an output parameter of a service is not consumed by the subsequent service, it can be skipped. If an output parameter should be reused as an input for multiple services, a separate edge has to be defined for each output-input parameter mapping.

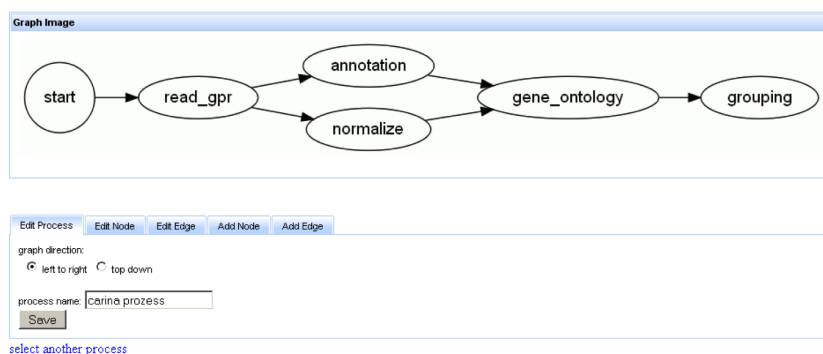


Figure 28: Process Editor - User Interface

There are two representation formats for processes. A detailed process

specification including parameter mappings is stored in the process database. This representation is used for both workflow definition and workflow execution. Further, a simplified graphical representation of the process consisting of task nodes and edges is given in the GraphViz Dot language. Whenever a process definition is modified, all changes are committed into the process database. Then, the database definition is translated into a valid Dot file which is in turn used as an input for generating a GraphViz image of the current process. In the following, an example Dot specification of the gene expression analysis workflow is presented. A GraphViz graph consists of an enumeration of labelled nodes and edge definitions between nodes. Each node is specified by a node label and each edge is represented by a directed dependency between two nodes. Joins and splits in the control flow of the process are modelled implicitly by adding the required dependency relationships. Both nodes and edges may have attached properties which are defined by key-value pairs enclosed by squared brackets. Node start has an attached shape property indicating that the node is drawn as a circle. URLs are attached as properties to nodes and edges allowing to trigger user interaction events. Thus, it is possible to select graphical components in the user interface and modify the process definition interactively.

Listing 1: Example Dot Process Specification [16]

```
digraph G {
  rankdir="LR"
  start[shape=circle URL="editProcessGraph.jspx?node=1"]
  patient_data[URL="editProcessGraph.jspx?node=3"]
  read_gpr[URL="editProcessGraph.jspx?node=2"]
  mapping[URL="editProcessGraph.jspx?node=5"]
  normalize[URL="editProcessGraph.jspx?node=4"]
  annotation[URL="editProcessGraph.jspx?node=6"]
  gene_ontology[URL="editProcessGraph.jspx?node=8"]
  grouping[URL="editProcessGraph.jspx?node=7"]
  analysis[URL="editProcessGraph.jspx?node=9"]
  plotting[URL="editProcessGraph.jspx?node=10"]
}
```

```

end[shape=doublecircle URL="editProcessGraph.jspx?node=11"]
start  -> patient_data [URL="editProcessGraph.jspx?edge=14"]
start  -> read_gpr [URL="editProcessGraph.jspx?edge=15"]
read_gpr  -> normalize [URL="editProcessGraph.jspx?edge=16"] ;
normalize  -> mapping [URL="editProcessGraph.jspx?edge=17"]
patient_data  -> mapping [URL="editProcessGraph.jspx?edge=18"]
normalize  -> annotation [URL="editProcessGraph.jspx?edge=19"]
annotation  -> gene_ontology [URL="editProcessGraph.jspx?edge=20"]
mapping  -> grouping [URL="editProcessGraph.jspx?edge=21"]
gene_ontology  -> analysis [URL="editProcessGraph.jspx?edge=22"]
grouping  -> analysis [URL="editProcessGraph.jspx?edge=23"]
analysis  -> plotting [URL="editProcessGraph.jspx?edge=24"]
plotting  -> end [URL="editProcessGraph.jspx?edge=25"]
}

```

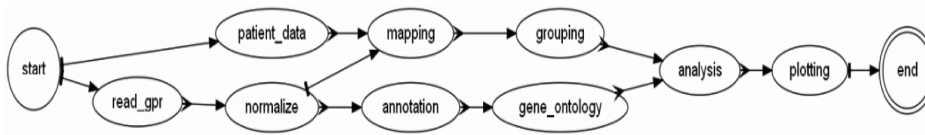


Figure 29: Process Editor - Example Process

The resulting process graph of the example dot specification is given in Figure 29. By selecting task nodes, task labels may be changed and new successor task nodes may be inserted. Process definitions are persistently stored in the process database which schema is illustrated in Figure 30.

All process definitions are captured in the process table. Each process has a unique process identifier and a user-defined process name. A process definition is composed of a set of tasks which are in turn realized by web services. Attributes `start_task` and `end_task` refer to corresponding start and end nodes in the process graph. Web services are captured in the service table which has attributes for defining service names and descriptions. Further, the physical address of the web service, specified as a URI, is captured in the endpoint attribute.

The functionality of the web service is described by a WSDL (Web Service

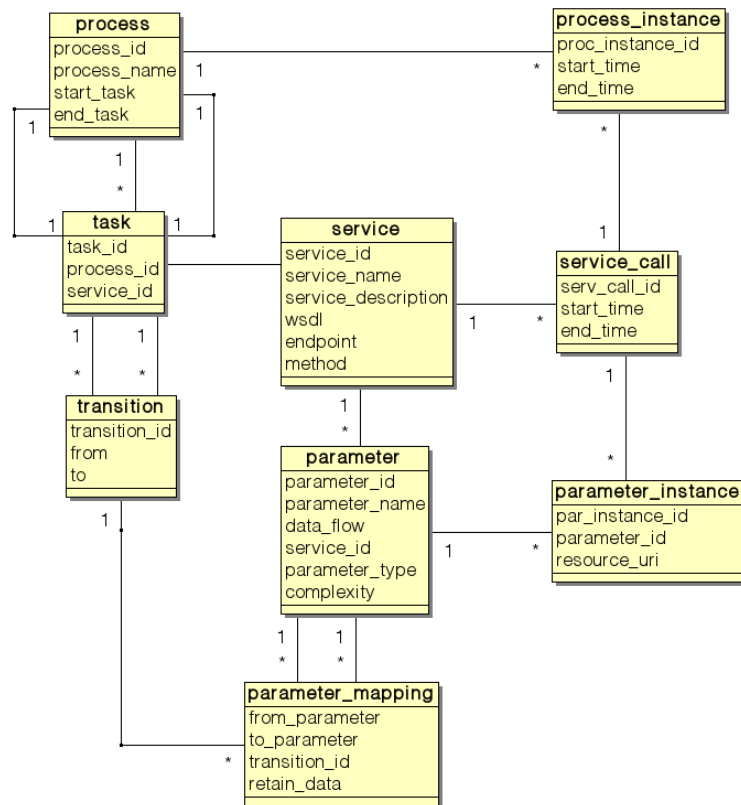


Figure 30: Process Database Schema

Description Language) file. The WSDL file is a XML document specifying all operations the web service exposes, all involved data types and message formats. Web services are registered in the process database by importing the corresponding WSDL file and automatically extracting all relevant information such as communication protocol and encoding, method name, name and type of parameters and endpoint address. For instance, the transportation of files between coordination and experiment servers is accomplished by the Java web service AttachmentService. The web service provides a set of methods for data file transfer, including the method uploadFile for staging files on the experiment servers. The corresponding signature of the imple-

menting Java method is:

Listing 2: AttachmentService

```
public String uploadFile(String name,  
    String attachmentID,String session_id)  
    throws IOException;
```

In the listing below, a fragment of the WSDL file of the AttachmentService is given. A web service may be provided on multiple ports. For instance, separate ports for SOAP 1.1, SOAP 1.2 and HTTP may be defined. In our example, the AttachmentService is provided for the SOAP 1.2 protocol. The web service endpoint is encoded in the location attribute of the soap12:address tag. For each port, the message formats used for transportation are declared in wsdl:binding tags. In the binding for AttachmentService, HTTP is defined as transport protocol for SOAP messages. The AttachmentService provides a set of operations - for simplicity, only the uploadFile operation is illustrated. The operation is encapsulated in a soap12:operation element. The style attribute is set to document indicating a transport style accomplished by messages. Soap messages are used as inputs and outputs of a web service. In our case, uncoded literal messages are deployed. Details of the uploadFile operation are given in the wsdl:portType element which defines the types of input, output and fault messages. Finally, the message types are specified in wsdl:message elements and the operation uploadFile in an own type definition. The three String input parameters are declared as a sequence of elements.

Listing 3: WSDL file of AttachmentService

```
<xs:element name="uploadFile">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element minOccurs="0" name="name" nillable="true" type="xs:string"/>  
      <xs:element minOccurs="0" name="attachmentID" nillable="true" type="xs:string"/>
```

```

<xs:element minOccurs="0" name="session_id" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<wsdl:message name="uploadFileRequest">
<wsdl:part name="parameters" element="ns:uploadFile"/>
</wsdl:message>
<wsdl:message name="uploadFileResponse">
<wsdl:part name="parameters" element="ns:uploadFileResponse"/>
</wsdl:message>
<wsdl:message name="IOException">
<wsdl:part name="parameters" element="ns:IOException"/>
</wsdl:message>

<wsdl:portType name="AttachmentServicePortType">
<wsdl:operation name="uploadFile">
<wsdl:input message="ns:uploadFileRequest" wsaw:Action="urn:uploadFile"/>
<wsdl:output message="ns:uploadFileResponse" wsaw:Action="urn:uploadFileResponse"/>
<wsdl:fault
  message="ns:IOException" name="IOException" wsaw:Action="urn:IOException"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="AttachmentServiceSoap12Binding"
  type="ns:AttachmentServicePortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
  style="document"/>
<wsdl:operation name="uploadFile">
<soap12:operation soapAction="urn:uploadFile" style="document"/>
<wsdl:input>
  <soap12:body use="literal"/>
</wsdl:input>
<wsdl:output>
  <soap12:body use="literal"/>
</wsdl:output>

```

```

<wsdl:fault name="IOException">
  <soap12:fault use="literal" name="IOException"/>
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="AttachmentService">
<wsdl:port name="AttachmentServiceHttpSoap12Endpoint"
  binding="ns:AttachmentServiceSoap12Binding">
<soap12:address
  location="[url]:[port]/axis2/services/AttachmentService.
AttachmentServiceHttpSoap12Endpoint/">
</wsdl:port>
</wsdl:service>

```

When registering new web services in the process database, the operation name is extracted from the WSDL definition and stored in the column method of the web service table. For each parameter of an operation, a new entry in the parameter table is created. Attribute data_flow defines, whether the parameter is an input or output parameter. Parameters may either have a simple data type (boolean, integer, float, String), a file data type or a collection data type. Currently, the collection data types ArrayList and Vector are supported. In order to model the data flow between tasks, the parameter_mapping table connects task transitions with web service parameters. Hence, output parameters of services can be consumed as input parameters of subsequent services. The retain_data flag specifies, whether parameter values are to be retained at execution time. Enactment of processes is accomplished by the process execution component presented in the next Section 5.5.2. The execution component captures protocol data in tables process_instance, service_call and parameter_instance during enactment of processes. A process definition may be instantiated multiple times, whereas the execution time is assessed in attributes start_time and end_time. of table process_instance. Similarly, service executions are logged in table service_call

whereas new entries in the `parameter_instance` table are created for each input and output parameter of the service. Parameter values of simple and collection data types are retained in table `parameter_instance`, while files are saved at the file system and the corresponding URIs are captured.

5.5.2 Process Execution Component

The Process Execution Component has been realized for enacting scientific workflows defined by the Process Editor. The component allows to load predefined process templates, instantiate processes and execute them based on user-defined input parameter values. A user interface was required that displays the process graph and allows the assignment of parameter values to individual tasks. Further, two execution modes had to be supported. On the one hand, a simple batch-execution mode was required in which the entire workflow is enacted without user interaction. On the other hand, a set of selected tasks should be executable in a partial execution mode. While batch-execution mode is preferred in long lasting execution runs with fixed parameter values, the partial execution mode supports more variable execution variants. For instance, if the execution of a workflow is to be repeated with changed parameter values of the last two tasks, a partial execution of these last steps may be initiated by using the output values of the previous run. The process execution component was implemented in context of the bachelor thesis of Erwin Mann [57].

The system architecture of the Process Execution Component consists of three main parts: the process database presented in the previous section, the execution controller web application and the execution engine. The execution controller was implemented as a Java web application that may be deployed on an application server such as Oracle GlassFish ¹⁷ or Apache Tomcat ¹⁸. The execution controller provides a web-based user interface in

¹⁷Oracle GlassFish <http://glassfish.java.net/>

¹⁸Apache Tomcat <http://tomcat.apache.org/>

order to select process templates from the process database, assign parameter values to individual tasks and manage the execution of processes. Based on user-defined input data, a new process execution template is generated and sent to the execution engine, which is responsible for the enactment of processes by consecutively calling web services. The execution engine has been realized by using the web service orchestration engine Apache Ode (Orchestration Director Engine) ¹⁹, which is capable of communicating with web services by message exchange, executing data manipulations and handling of execution errors. Apache ODE is deployed as a WAR file (web application archive) on a Java application server and provides a web-based management interface for monitoring process executions and message exchanges. Apache ODE supports two kinds of communication layers: Java Business Integration standard (JBI) [19] and HTTP transport based on Axis2 [32]. As all services were implemented as Axis2 web services in the GATiB project, only this type of communication layer was used. The Apache ODE engine is able to execute processes defined in the Web Service Business Process Execution Language (WS-BPEL), which is a specification for business processes that are based on web services. The latest version of the specification is WS-BPEL 2.0, released in 2007 by the Organization for the Advancement of Structured Information Standards (OASIS) [30]. The task of the execution controller is to create a valid WS-BPEL specification based on the internal process representation format stored in the process database and on the effective parameter values entered by the user. That is, a transformation mechanism was required for dynamically generating a WS-BPEL specification which is used as input for the Apache ODE engine. The execution controller extracts all relevant information about used Java web services (WSDL-files, endpoint references) from the process database and combines the information with process structure data and user-defined parameter values.

WS-BPEL was invented as a language for defining business process be-

¹⁹Apache Ode <http://ode.apache.org/>

behaviour based on web services. Generally, WS-BPEL strives to support two types of business processes: executable business processes and abstract business processes. Abstract business processes have a descriptive function for assessing the communication pattern of multiple parties participating in a business use case. They omit details of internal behaviour and are not intended to be executed. In contrast, executable business processes are fully specified and model the actual behaviour of all parties in a use case. For executable business processes, WS-BPEL is used as an execution plan that is solely based on web service resources and XML data [30].

A few prerequisites are necessary for successfully executing a WS-BPEL specification with the help of Apache Ode. Firstly, a valid WS-BPEL definition has to be present in a .bpel file, which is a XML document containing a root element process, which has various XML namespaces declarations. The default namespace is represented by the WS-BPEL 2.0 namespace provided by OASIS. Further, WSDL and SOAP namespaces are referenced as well. For each web service, that is used in the WS-BPEL process, a namespaces declaration of its corresponding WSDL files is defined. A simplified WS-BPEL specification of a process consisting of two web services (Patient_Data_Reader, and Link_Gene_Expression) is presented in the listing below.

Listing 4: Example WS-BPEL Specification [57]

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:tns="http://www.cs.univie.ac.at/gatib/bpel/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
  xmlns:nsschema="http://services/xsd"
  xmlns:Main.wSDL="http://www.cs.univie.ac.at/gatib/Main_wSDL"
  xmlns:Patient_Data_Reader.wSDL=
    "http://localhost:8080/axis2/services/Patient_Data_Reader"
```

```

xmlns:Link_Gene_Exprs.wsdl=
  "http://localhost:8080/axis2/services/Link_Gene_Exprs"
name="gene_expression_analysis"
targetNamespace="http://www.cs.univie.ac.at/gatib/bpel/">
<import location="Main.wsdl"
  namespace="http://www.cs.univie.ac.at/gatib/Main_wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/" />
<import location="Patient_Data_Reader.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/"
  namespace=
    "http://localhost:8080/axis2/services/Patient_Data_Reader" />
<import location="Link_Gene_Exprs.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/"
  namespace="http://localhost:8080/axis2/services/Link_Gene_Expr" />

```

The name of the example process is captured in the name attribute of the process element. The target namespace of the declaration assigns a unique identifier to the WS-BPEL specification. The WSDL files of the used web services are referenced by import elements. A coordination web service Main is embedded in every generated WS-BPEL specification. The coordination service is responsible for initializing and preparing the execution of the WS-BPEL process. The process element has four embedded subsections: partnerLinks, variables, faultHandlers and sequence definition. However, the section faultHandlers, which is responsible for dealing with errors resulting from service invocation, has not been considered in the automatic generation of WS-BPEL process definitions.

Listing 5: WS-BPEL Process Structure

```

<process>
  <partnerLinks> </partnerLinks>
  <variables> </variables>
  <faultHandlers> </faultHandlers>
  <sequence> </sequence>
</process>

```

Business processes communicate with each other using a peer-to-peer interaction pattern. That is, if a relationship between two business processes is established, each process may act as both service provider and service consumer. Similarly, business processes may communicate directly with web services. Both business processes and web service act as communication *partners*. For each service that is consumed or provided in a process, a `partnerLink` is declared, which is an instance of a typed connector. A `partnerLink` has a dedicated name and refers to a `partnerLinkType`, that is defined in the WSDL of the involved web service through the WSDL extensibility mechanism. A `partnerLinkType` encapsulates ports of web services and defines the type of communication (synchronous or asynchronous) between the communication partners. In a synchronous communication, the service consumer sends an *invoke* message to the service provider, which accepts the message through a *receive* activity and executes the requested operation. Afterwards, a *reply* message is generated and sent back to the service consumer. By contrast, in an asynchronous communication pattern, an *invoke* message is sent from the service consumer and received by service provider. After the operation was executed, a callback is issued by sending an *invoke* message from the service consumer to the service provider. The roles of service consumer and service provider in the communication pattern are defined by `plink:role` elements inside the `partnerLinkType` declaration. Up to two roles may be given for each `partnerLinkType`. If only one role is specified, a synchronous communication pattern is defined, in which the `plink:role` element identifies the role of the service requester. In case of two `plink:role` elements, separate roles for service requester and service provider are specified for an asynchronous communication pattern. In the example below, a `partnerLinkType` for the `Patient_Data_Reader` service is given. This `partnerLinkType` exposes the `Patient_Data_Reader` service at the port `axis2:Patient_Data_Reader_PortType` for service requesters in a synchronous communication pattern. Since Axis2 web services do not include `partnerLinkType` elements in the WSDL, they

have to be generated separately and added to the WSDL definition.

Listing 6: WS-BPEL PartnerLinkType

```
<plink:partnerLinkType
  name="Patient_Data_Reader_PartnerLinkType">
  <plink:role name="Patient_Data_Reader_Role"
    portType="axis2:Patient_Data_Reader_PortType"/>
</plink:partnerLinkType>
```

The set of used partnerLinks of the WS-BPEL process is embedded inside a partnerLinks element. The roles the current process and the service provider take on in a partnerLink are given by attributes myRole and partnerRole.

Listing 7: WS-BPEL PartnerLinks [57]

```
<partnerLinks>
  <partnerLink name="MainPartnerLink"
    partnerLinkType="main.wsdl:MainPartnerLinkType"
    myRole="me"/>
  <partnerLink name="Patient_Data_Reader_PartnerLink"
    partnerLinkType="Patient_Data_Reader.wsdl:Patient_Data_Reader_PartnerLinkType"
    partnerRole="Patient_Data_Reader_Role"/>
  <partnerLink name="Link_Gene_Expr_PartnerLink"
    partnerLinkType="Link_Gene_Expr.wsdl:Link_Gene_Expr_PartnerLinkType"
    partnerRole="Link_Gene_Expr_Role"/>
</partnerLinks>
```

In the above example, two partnerLink elements are listed for web services Patient_Data_Reader and Link_Gene_Expression and the partnerRole is set to the role names of the related partnerLinkType roles. Both services are called in simple request-reply communication patterns. The WS-BPEL itself serves as a service provider which may be called by other web services or business processes. Therefore, the partnerLink MainPartnerLink is defined with a myRole attribute. This partnerLink refers to the coordination service Main, which is executed as an initialization service. The Main service is required

in order to start the WS-BPEL process by sending a SOAP message to it.

The current state of a process is based on the set of messages that are exchanged between partners and on the set of intermediate data that is generated during process execution. WS-BPEL variables are used to maintain the state of a process and to model the data flow between partners. Variables are specified by variable element declarations inside of a variables block. WS-BPEL supports three different types of variables: WSDL message type, XML Schema type and XML Schema element. The variable type is assessed by attributes `messageType`, `type` and `element` [30]. WS-BPEL variables follow the following life cycle: declaration, initialization and usage. In the listing below, two variables for the web service `Patient_Data_Reader` are declared, one for the request message and one for the returned response message.

Listing 8: WSDL `MessageType`

```
<variable name="Patient_Data_ReaderRequest"
  messageType="Patient_Data_Reader.wsdl:read_patient_dataRequest"/>
<variable name="Patient_Data_ReaderResponse"
  messageType="Patient_Data_Reader.wsdl:read_patient_dataResponse"/>
<xs:element name="read_patient_data">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="directory_name" nillable="true"
        type="xs:string"/>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="attributes"
        nillable="true" type="xs:string"/>
      <xs:element maxOccurs="unbounded" minOccurs="0"
        name="filter_parameter" nillable="true" type="xs:string"/>
      <xs:element maxOccurs="unbounded" minOccurs="0"
        name="filter_parameter_values" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="output_file_name" nillable="true"
        type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The message types are declared in the linked WSDL file of the service. Message type `read_patient_dataRequest` has an embedded element `read_patient_data`. This element encapsulates the input parameters of the web service. Some of the input parameters such as `directory_name` or `output_file_name` have simple data types, while others such as `filter_parameter` have composed data types, indicated by `maxOccurs=unbounded` attributes. Complex data types correspond to Java ArrayLists or Java Vectors. An example of instantiated input parameters is given in Figure 31.

Listing 9: WSDL Message Parameters

```
<wsdl:message name="read_patient_dataRequest">
<wsdl:part name="parameters"
  element="ns0:read_patient_data"/>
</wsdl:message>
<wsdl:message name="read_patient_dataResponse">
<wsdl:part name="parameters"
  element="ns0:read_patient_dataResponse"/>
</wsdl:message>
```

Input parameter values `session1010` and `patient_list.txt` constitute single data types, while composed values are assigned to input parameters attributes, `filter_parameter` and `filter_parameter_values`.

The initialization of WS-BPEL variables occurs by assign activities, which allow to copy data values between variables as well as assigning new data values to variables. In order to initialize the variable `Patient_Data_ReaderRequest`, the type information of the embedded element `read_patient_data` is extracted from the WSDL and separate elements for all defined input parameters are generated. The entire element is placed into the from-part of the assign statement while the to-part is filled with the message variable `Patient_Data_ReaderRequest`. The `read_patient_data` element is copied as a message part into the message variable. The newly generated message part is labelled `parameters`.

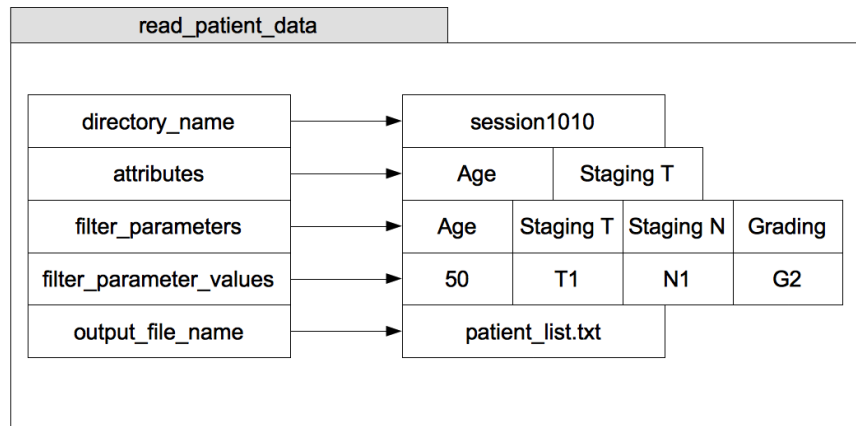


Figure 31: Process Execution Component - Parameter Passing

Listing 10: WS-BPEL Initialize Variables [57]

```
<assign name="Initialize_variable_Patient_Data_ReaderRequest">
  <copy>
    <from>
      <literal xml:space="preserve">
        <nsschema:read_patient_data
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <nsschema:directory_name></nsschema:directory_name>
          <nsschema:attributes></nsschema:attributes>
          <nsschema:filter_parameter></nsschema:filter_parameter>
          <nsschema:filter_parameter_values></nsschema:filter_parameter_values>
          <nsschema:output_file_name></nsschema:output_file_name>
        </nsschema:read_patient_data>
      </literal>
    </from>
    <to variable="Patient_Data_ReaderRequest" part="parameters"/>
  </copy>
</assign>
```

Although the message variable has been initialized, all embedded input parameter elements do not include values. Input parameter values may be

assigned to web services in two ways: either by collecting values from the web user interface or by receiving parameter values from output parameters of completed web services. Both parameter value assignments are represented by assign activities. Single values may be copied by declaring a literal inside the from element. The copy destination element is specified using XPath 1.0 addressing patterns. The destination message variable is referenced by a leading '\$' character, while the delimiter '.' separates the message variable from its message parts. Embedded elements are accessed using typical XPath location paths. In the example below, element `directory_name` is filled with the String literal `session1010`.

Listing 11: WS-BPEL - Assign Simple Parameter Value

```
<assign name="simple_param_directory_name_3">
  <copy>
    <from>
      <literal>session1010</literal>
    </from>
    <to>
      $Patient_Data_ReaderRequest.parameters/nsschema:directory_name
    </to>
  </copy>
</assign>
```

Similarly, elements having composed data types are filled with assign activities. Apache ODE provides an extension of the WS-BPEL copy element allowing to create sequences of elements without overwriting existing ones. The copy element has an attribute `insert` which specifies, where the new element should be inserted in a sequence of other elements. The extension allows a comfortable handling of collections of data values. In the example below, a new filter parameter element is generated and appended at the end of the sequence of filter parameters. Passing of parameter values between web services is also accomplished by assign activities. Within copy elements, the output parameters of completed web services are extracted from wsdl

output messages and passed as input parameters of subsequent services.

Listing 12: WS-BPEL - Assign Complex Parameter Value [57]

```
<assign name="complex_param_filter_parameter_3">
  <copy insert="last">
    <from>
      <literal>Staging T</literal>
    </from>
    <to>
      $Patient_Data_ReaderRequest.parameters/nsschema:filter_parameter
    </to>
  </copy>
</assign>
```

In the following listing, web service Patient_Data_Reader returns a file name of a patient list which is used as an input parameter in web service Link_Gene_Expression.

Listing 13: WS-BPEL - Extract Output Parameter Value [57]

```
<assign name="assign_output_3">
  <copy>
    <from>
      $Patient_Data_ReaderResponse.parameters/nsschema:return
    </from>
    <to>
      $Link_Patients_Gene_ExpressionsRequest.parameters/nsschema:patient_data_file
    </to>
  </copy>
</assign>
```

The execution order of web services is defined inside sequence elements of the WS-BPEL specification. Parallel executions are represented by flow elements, which are embedded within sequence elements. Based on the process graphs of the process database, the execution controller derives the execution order of web services and arranges web service invocations in sequence and flow elements. A simplified execution sequence is illustrated in the list-

ing below. The execution sequence consists of the invocation of the Main service and the Patient_Data_Reader service. The initial receive activity is used to capture an initialization message for the Main service. Afterwards, the initialization of all input parameters of service Patient_Data_Reader occurs. The set of input parameters is declared and stored in the message variable Patient_Data_ReaderRequest. After assigning actual values to input variables, the invocation of service Patient_Data_Reader is triggered, using corresponding partnerLink, portType and operation values. Finally, the output value resulting from the service execution is passed as an input for the subsequent Link_Gene_Expression service.

Listing 14: WS-BPEL - Execute Process

```
<sequence>
  <receive name="start" partnerLink="MainPartnerLink"
    portType="main.wsdl:MainPortType"
    operation="main" variable="start_var" createInstance="yes"/>

  <!-- Variable initialization -->
  <assign name="Initialize_variable_Patient_Data_ReaderRequest">
    <copy>
      <from>
        <literal xml:space="preserve">
          <nsschema:read_patient_data>
            <nsschema:directory_name/>
            <nsschema:attributes/>
            <nsschema:filter_parameter/>
            <nsschema:filter_parameter_values/>
            <nsschema:output_file_name/>
          </nsschema:read_patient_data>
        </literal>
      </from>
      <to variable="Patient_Data_ReaderRequest" part="parameters"/>
    </copy>
  </assign>
```

```

<!-- Fill variable values -->
<assign name="simple_param_directory_name_3">
  <copy>
    <from>
      <literal>session1010</literal>
    </from>
    <to>
      $Patient_Data_ReaderRequest.parameters/nsschema:directory_name
    </to>
  </copy>
  .....
  .....
</assign>

<!-- Invoke web service -->
<invoke name="patient_data_invoke"
  partnerLink="Patient_Data_ReaderPartnerLink"
  portType="Patient_Data_Reader.wsdl:Patient_Data_ReaderPortType"
  operation="read_patient_data" inputVariable="Patient_Data_ReaderRequest"
  outputVariable="Patient_Data_ReaderResponse"/>

<!-- Pass output values to subsequent service -->
<assign name="assign_output_3">
  <copy>
    <from>
      $Patient_Data_ReaderResponse.parameters/nsschema:return
    </from>
    <to>
      $Link_Patients_Gene_ExpressionsRequest.parameters/nsschema:patient_data_file
    </to>
  </copy>
</assign>
  .....
  .....
</sequence>

```

The user interface for process execution is presented in Figure 32. In the upper part of the user interface, the process graph of the current process is displayed. Below, a sequential listing of all services used in the process definition is given. Due to space restrictions, only the first two services of the process are illustrated. For each service, an input block for specifying input parameters is generated. Some of the parameters have default values suggested by the system. For instance, the output file name of service `read_gpr` is set to `gepx_matrix.txt`. User-defined parameter values may be set by opening context-dependent selection panels. Thus, a list of selectable GPR files is displayed for defining filter criteria for the `read_gpr` service. The information, which input value may be specified for which service is extracted from the experiment database. For instance, if a certain patient dataset is to be used in an experiment, metadata about the dataset allows to select columns and available values. Output values of every service may be retained for process reexecutions by activating the retain data checkbox. Retention of intermediate results is activated per default, in order to enable capturing of data provenance. If the generated output is memory-intensive, or the service result should be excluded from data provenance recording, a deactivation is recommended. Standalone execution of services is supported, if all required user-input parameters are filled with values and throughput parameter values from preceding services are available.

Alternative processes may be loaded from the process database by using the Load Process link. Two execution modes are supplied: normal process execution and process reexecution. In the normal execution mode, a new WS-BPEL specification is created based on the current process definition and all specified user-defined input parameter values. In the process reexecution mode, a comparison between the current input parameter values and the one of the previous execution run is made. Services must be only reexecuted, if they have changed input parameter values or immediately or transitively depend on output parameter values of services with changed input param-

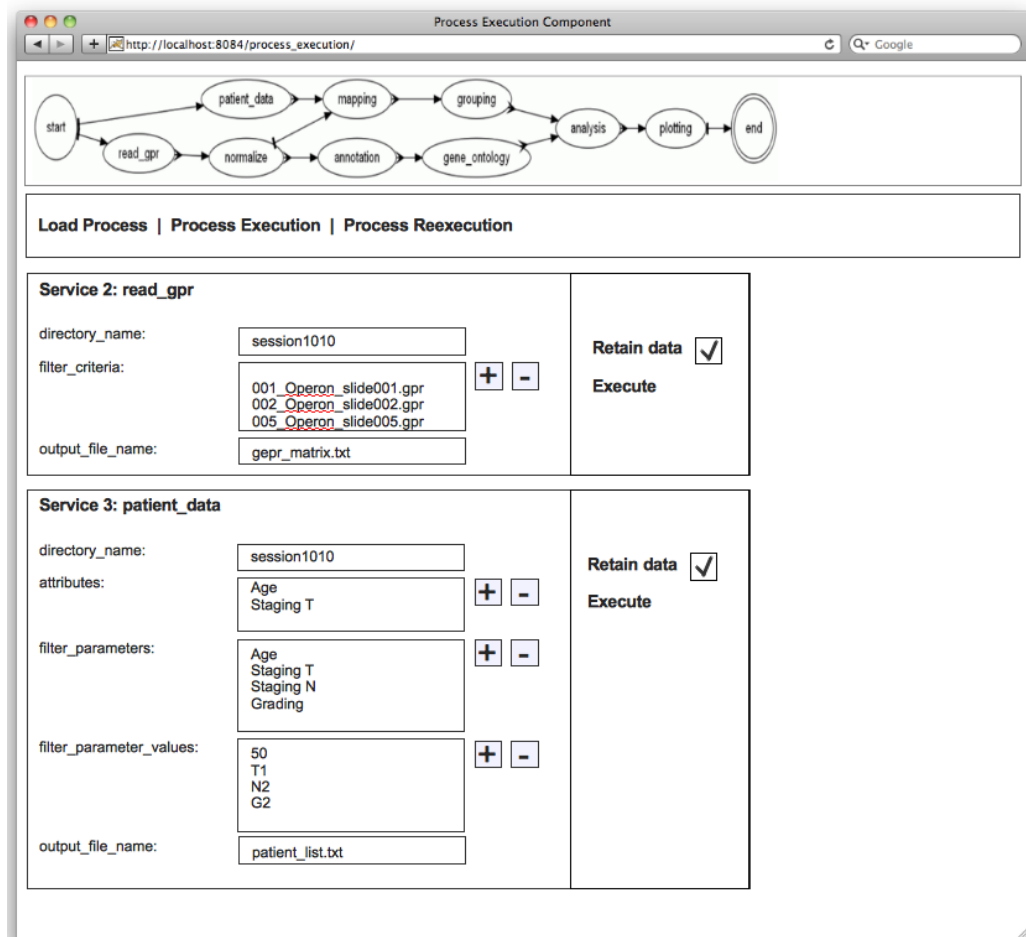


Figure 32: Process Execution - User Interface

eter values. After successful process execution, all produced output data is displayed on a result screen. Generated output files may be downloaded or persistently saved in the experiment database.

5.6 Data Provenance Model

In this Section, a data provenance model is presented that fulfils the GATiB-related provenance requirements presented in Section 5.2.2. Initially, a rough overview of the provenance model is given followed by a detailed formal specification of the model in Subsection 5.6.1. In Subsection 5.6.3, a selected set of provenance queries from the biomedical research domain is answered using the presented provenance model. Further, in Subsection 5.6.2, generalized provenance queries presented in 5.3.2 are answered formally in order to prove that the presented provenance model is compliant to the Open Provenance Model (OPM).

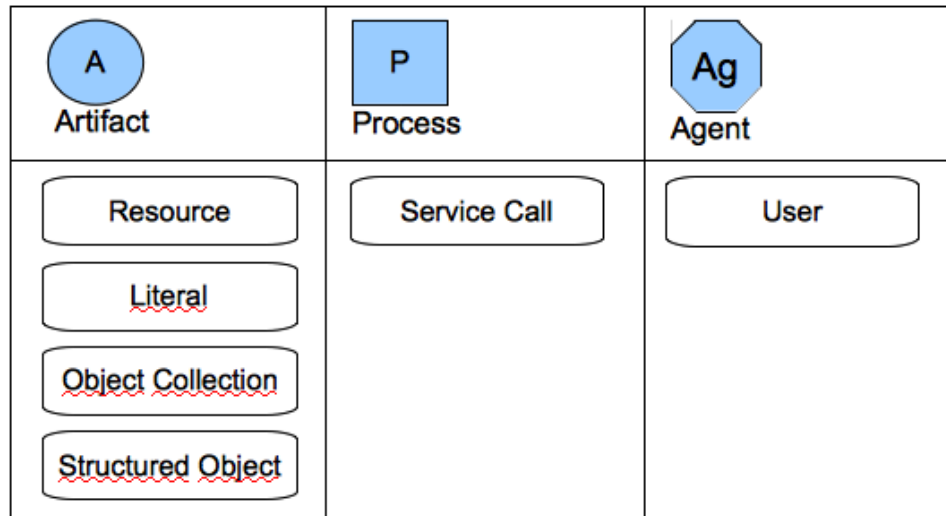


Figure 33: OPM - Semantic Provenance Model Match - Entities

In the following, the compliance of the semantic provenance model with the OPM is shown. The basic entities of the OPM may be mapped to the main classes of the developed provenance model, as illustrated in Figure 33. Entity Artifact is represented by class Resource and its subclasses Literal, Object_Collection and Structured_Object. The provenance model

captures different types of artifacts: Literals are typical simple parameter values such as Strings or numbers. Structured_Objects are data objects composed of attribute values and Object_Collections are sets of Literals or Structured_Objects. Class Service_Call is equivalent to the OPM entity Process. A Service_Call represents the execution of a certain Service taking Resources as input parameter values and produce Resources as output parameter values. OPM entity Agent models the controller of a Process execution. In the semantic provenance model, Users execute Services or sequences of Services.

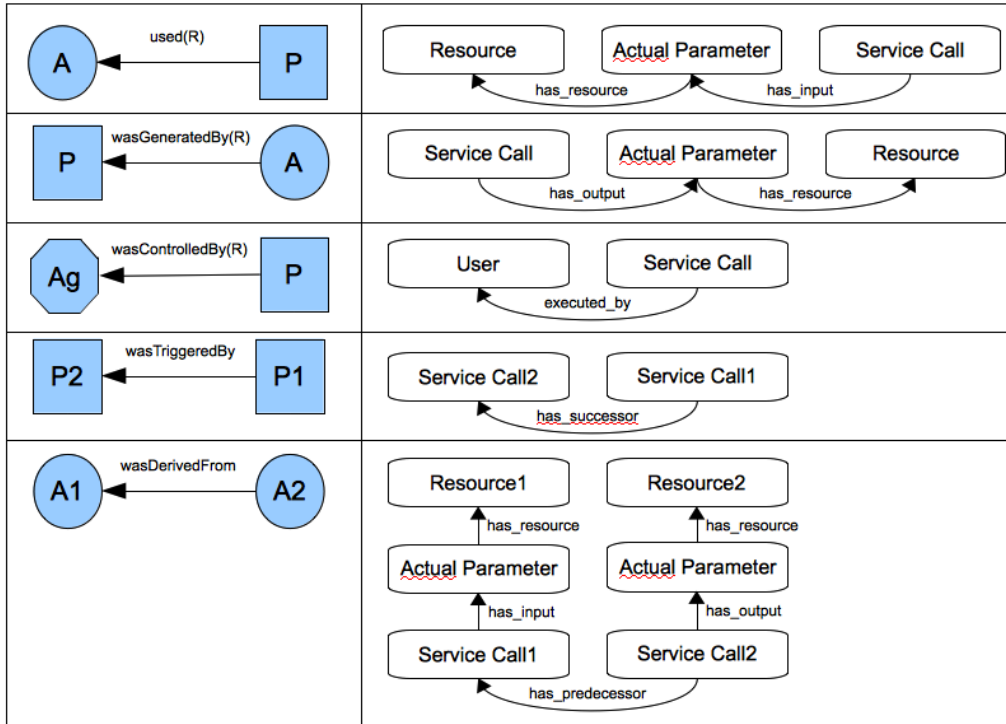


Figure 34: OPM - Semantic Provenance Model Match - Relationships

OPM postulates a set of data derivation relationships and control relationships. The corresponding relationships in the semantic provenance model are presented in Figure 34. Artifacts that are used by Processes are represented by connecting Resources with Actual_Parameters by has_resource re-

relationships and by connecting Actual_Parameters of has_input relationships with Service_Calls. Similarly, the OPM relationship wasGeneratedBy, that specifies an Artifact as a generated product of a Process, is represented. An output parameter of a Service_Call is an Actual_Parameter connected by a has_output relationship and a Resource is assigned to the Actual_Parameter by a has_resource relationship. The control relationship wasControlledBy between a Process and an Agent corresponds to the executed_by relationship between a Service_Call and a User. If a Process P1 consumes output data of a Process P2, P2 is said to be triggered by P1. In the semantic provenance model, this dependency is captured by connecting two Service_Calls with a has_successor relationship. Data derivation dependencies are modelled by wasDerivedFrom relationships in the OPM. In order to identify all Resources that were involved in the generation of a certain Resource, several relationships are required in the semantic provenance model. First, the Service_Call that created the Resource is detected by following the has_resource relationship to the corresponding Actual_Parameter and the has_output relationship to the Service_Call. Then all preceding Service_Calls are identified by has_predecessor relationships. Finally, all Resources used as input parameter values of these Service_Calls are found by has_input and has_resource relationships.

The data provenance model was specified in the knowledge representation language OWL-DL [18]. On the one hand OWL-DL is a powerful knowledge representation language that may be used to capture domain expertise, which is particularly complex in the field of medical research. OWL-DL allows to model detailed and restrictable relationships and facilitates verification of instance data as well as classifying data by subsumption. On the other hand, our model includes several transitive relationships that may be efficiently handled by reasoners. Moreover, we wanted to benefit from the powerful query languages that may be applied to ontologies (e.g. SPARQL). The provenance model is captured and maintained by using the Apache Jena

framework [31]. Jena provides a set of tools and interfaces for accessing and processing OWL ontologies and stores them persistently in relational databases (e.g. MySQL or PostgreSQL). Moreover, Jena has an integrated query engine allowing to formulate SPARQL queries against ontologies.

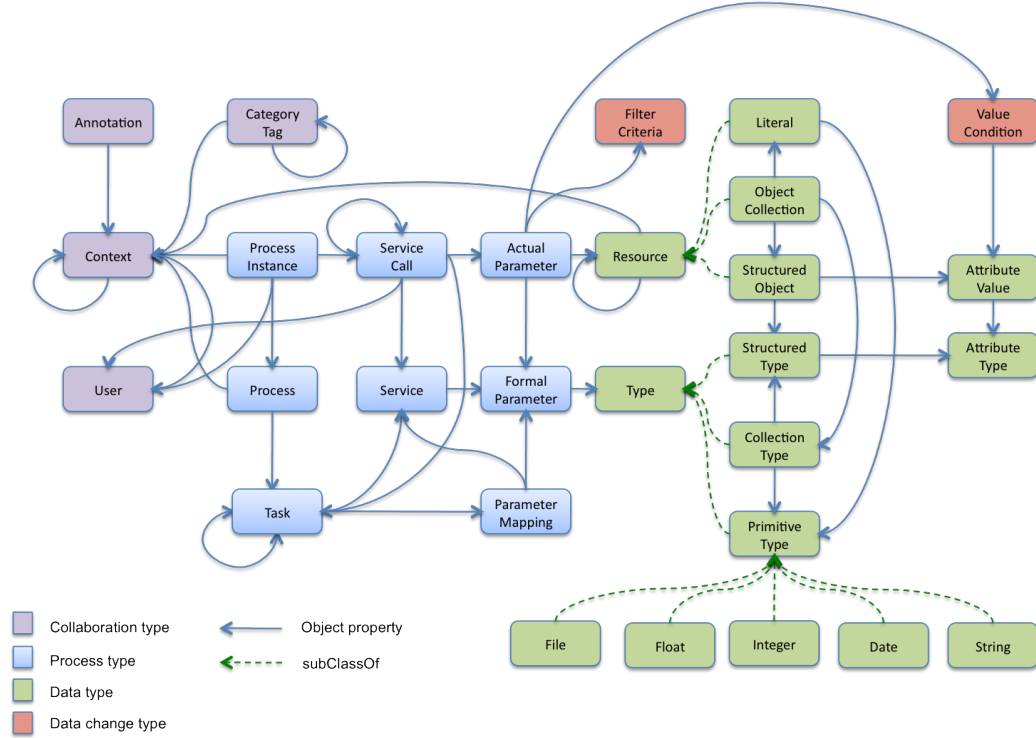


Figure 35: Data Provenance Model Overview

Figure 35 depicts the class structure of the ontology model. For simplicity, only the main relationships between the classes are illustrated. Subclass relationships between classes (`rdfs:subClassOf`) are represented by green arrows and object properties by blue arrows. The entire model including all object and data properties is specified in the following section 5.6.1. Generally, the provenance model combines collaboration-related data with data resulting from process execution and detailed metadata about accessed resources. Classes `context`, `annotation`, `user` and `category_tag` capture data about collab-

orations in virtual knowledge spaces of GAMECS-CSCW. A virtual knowledge space is a dedicated area, in which data and knowledge is shared for a group of collaborating persons. In context of medical research, knowledge spaces may be created for specific studies or projects. Each knowledge space is modelled by an instance of class `Context`, whereas contexts may be nested into each other and form context hierarchies. Semantic enrichment of knowledge is accomplished by assigning textual `Annotations` and `Category` tags. Collaborating persons are modelled by instances of class `User`. Data items that are shared in knowledge spaces are represented by instances of class `Resource`. Resources may either be simple `Literals`, `Structured_Objects`, which are database records transformed into OWL instances and properties or `Object_Collections`. Each database record has a root `Structured_Object` and every database attribute of the record is represented by an instance of class `Attribute_Value` and connected by object properties (`has_attribute_value`) with the root object. For instance, a patient record with attributes `Age`, `Staging_T`, `Staging_N` and `Staging_M` is mapped to a root `Structured_Object` that has four `Attribute_Value` objects containing the corresponding attribute values. `Object_Collections` are lists of objects. For instance, if a set of medical records is to be analyzed, the records are passed as a Java Array to a web service and recorded as an `Object_Collection` in the provenance model. Collections may be composed of `Literals` or `Structured_Objects`. `Object_Collections` are a key aspect of the model, since they allow to trace single objects that have been processed by services together with other objects of the same type. Class `Type` captures the metadata of `Resources`. A `Literal Resource` has one of the `Primitive_Types` `String`, `Date`, `Integer`, `Float` or `File`. The metadata of `Structured_Objects` is stored in `Structured_Types` which is composed of a set of `Attribute_Types`. In the previous example, a `Structured_Type` `Patient` with four `Attribute_Types` for attributes `Age`, `Staging_T`, `Staging_N` and `Staging_M` would be created. `Object_Collections` have a `Collection_Type` which refers either to a `Structured_Type` or to a `Prim-`

itive_Type. Several process-related classes exist which are responsible for recording process enactment and access of resources. Process definitions are stored in classes Process, Task and Service. A process definition consists of a sequence of Tasks, whereas the functionality of a Task is implemented by a dedicated Service. Thus, services may be used in different process definitions. Services have predefined input and output parameters of a certain Type. All parameters of a Service are stored as Formal_Parameters. The execution order of Tasks is implicitly defined by predecessor and successor object properties. When executing a Process, a Process_Instance is created and all invoked Services are documented as Service_Calls. Service_Calls access Resources as input parameters and generate new Resources as output parameters. The relationship between Service_Calls and Resources is modelled by Actual_Parameters. In order to handle data changes in resources classes Filter_Criteria and Value_Condition are used. Both classes allow to define validity constraints on Resource values. If Resource values are changed - due to erroneous or incomplete data - violations of validity constraints may be detected by OWL reasoners. In Section 5.6.5, handling of data changes and deploying of compensation techniques are described in detail.

5.6.1 Formal Representation

Generally, we notate class names in italic letters. Data properties are specified as data_property_name:{type}. Object properties are notated as property_name{Class_name} and transitive properties are marked as Trans.

Context	context_name:{string}	Name of context (knowledge space)
	creation_time:{dateTimeStamp}	Time of context initialization
	is_subcontext{ <i>Context</i> } Trans.	Context is nested in other context
continued on next page		

	related_context{ <i>Context</i> } Trans. has_annotation{ <i>Annotation</i> } has_resource{ <i>Resource</i> }	Context is related to other context Assignment of annotations Reference to resource that is accessed in context
<i>Annotation</i>	annotation_text:{string} annotation_key:{string} is_assigned_to{ <i>Context</i> }	Free text annotation Key value of annotation Reference to enclosing context
<i>Category_Tag</i>	tag_name:{string} is_assigned_to{ <i>Context</i> } has_subtag{ <i>Category_Tag</i> } Trans.	Name of category tag Reference to tagged context Subtag relationship for defining tag hierarchies
<i>User</i>	user_id:{string} prename:{string} surname:{string} is_assigned_to{ <i>Context</i> }	Unique user identifier Prenome Surname Reference to enclosing context
<i>Process</i>	process_name:{string} process_description:{string} creation_time:{dateTimeStamp} is_assigned_to{ <i>Context</i> } initial_task{ <i>Task</i> } end_task{ <i>Task</i> }	Unique process identifier Free text description of process scope Time of process definition Reference to enclosing context First task in execution sequence Last task in execution sequence
<i>Process_Instance</i>	start_time:{dateTimeStamp} end_time:{dateTimeStamp} invalid:{boolean} additional_data_available:{boolean} is_assigned_to{ <i>Context</i> } executed_by{ <i>User</i> } starts_with{ <i>Service_Call</i> }	Time the process was started Time the process was finished Specifies validity Reexecution yields extended result set Reference to context the process instance is embedded into Specifies the executing user First service call in execution sequence
continued on next page		

	ends_with{ <i>Service_Call</i> }	Last service call in execution sequence
	has_process_definition{ <i>Process</i> }	Reference to process template
Task	task_name:{string} has_defined_successor{ <i>Task</i> } Trans. has_defined_predecessor{ <i>Task</i> } Trans. has_service{ <i>Service</i> }	Task identifier, for instance read_patient_data Reference to succeeding task Reference to preceding task Reference to service that implements the requested functionality
Service	service_name:{string} service_description:{string} state:{string} runtime:{string} endpoint_reference:{string} record_provenance_data:{boolean} has_input_parameter{ <i>Formal_Parameter</i> } has_output_parameter{ <i>Formal_Parameter</i> } has_annotation{ <i>Annotation</i> }	Unique service identifier Description of service functionality Identical (stateless) or different (stateful) results at each run Runtime categories for service execution (immediate, short, medium, long) Reference to physical web service instance Enable or disable provenance recording Reference to input parameter (web service signature) Reference to output parameter (web service signature) Assignment of annotations
Service_Call	service_call_id:{string} start_time:{dateTimeStamp} end_time:{dateTimeStamp} invalid:{boolean} additional_data_available:{boolean}	Unique service call identifier Time the service call was started Time the service call was finished Specifies validity Reexecution yields extended result set
continued on next page		

	executed_by{ <i>User</i> } has_input{ <i>Actual_Parameter</i> } has_output{ <i>Actual_Parameter</i> } has_succesor{ <i>Service_Call</i> } Trans. has_predecessor{ <i>Service_Call</i> } Trans. has_service{ <i>Service</i> } has_task{ <i>Task</i> }	Specifies the executing user Reference to input parameter value Reference to output parameter value Reference to succeeding web service call Reference to preceding web service call Specifies the service definition of the call Reference to corresponding process step (task)
<i>Formal_Parameter</i>	parameter_id:{string} parameter_name:{string} has_type{ <i>Type</i> }	Unique parameter identifier Parameter name Specifies parameter type
<i>Parameter_Mapping</i>	mapping_id:{string} from_service{ <i>Service</i> } to_service{ <i>Service</i> } from_parameter{ <i>Formal_Parameter</i> } to_parameter{ <i>Formal_Parameter</i> }	Unique mapping identifier Source service Destination service Source parameter Destination parameter
<i>Actual_Parameter</i>	parameter_id:{string} has_formal_parameter{ <i>Formal_Parameter</i> } has_resource{ <i>Resource</i> }	Unique identifier Reference to formal parameter Reference to resource used as input or output value
<i>Filter_Criteria</i>	criteria_id:{string}	Unique identifier
<i>Value_Condition</i>	value_condition_id:{string}	Unique identifier
<i>Type</i>	type_id:{string} type_name:{string} type_description:{string}	Unique type identifier Name of type Description of type
<i>Structured_Type</i>	subClassof{ <i>Type</i> } has_attribute_type{ <i>Attribute_Type</i> }	Assignment of attribute types
<i>Collection_Type</i>	subClassof{ <i>Type</i> }	
continued on next page		

	composed_of{ <i>Primitive_Type</i> , <i>Structured_Type</i> }	
<i>Primitive_Type</i>	subClassof{ <i>Type</i> }	
<i>Attribute_Type</i>	attribute_id:{string} attribute_name:{string} attribute_data_type:{string}	Unique attribute identifier Name of attribute Type of attribute
<i>String</i>	subClassof{ <i>Primitive_Type</i> }	
<i>Boolean</i>	subClassof{ <i>Primitive_Type</i> }	
<i>Date</i>	subClassof{ <i>Primitive_Type</i> }	
<i>Integer</i>	subClassof{ <i>Primitive_Type</i> }	
<i>Float</i>	subClassof{ <i>Primitive_Type</i> }	
<i>File</i>	subClassof{ <i>Primitive_Type</i> }	
<i>Resource</i>	resource_id:{string} resource_id:{string} resource_origin:{string} version_nr:{integer} invalid:{boolean} is_subversion{ <i>Resource</i> } Trans. has_annotation{ <i>Annotation</i> }	Unique type identifier Physical location of resource Documents origin of resource Version number of resource Specifies validity Reference to previous version Assignment of annotations
<i>Structured_Object</i>	subClassof{ <i>Resource</i> } has_type{ <i>Structured_Type</i> } has_attribute_value{ <i>Attribute_Value</i> }	Reference to type Assignment of attribute values
<i>Object_Collection</i>	subClassof{ <i>Resource</i> } has_type{ <i>Collection_Type</i> } has_member{ <i>Literal</i> , <i>Structured_Object</i> }	Reference to type Specifies members of collection
<i>Literal</i>	subClassof{ <i>Resource</i> } has_type{ <i>Primitive_Type</i> }	Reference to type
<i>Attribute_Value</i>	subClassof{ <i>Resource</i> } has_value:{string,integer,float,dateTime}	Encapsulates attribute value

5.6.2 Answering General Data Provenance Queries

In the following, the First Provenance Challenge Queries are answered by formulating SPARQL queries against the semantic provenance model. For simplicity, all input and output values of service executions are considered as simple Literals. However, Structured_Objects and Object_Collections may be integrated in the provenance queries as well.

- PC1: Based on output data d_x of workflow instance $wfe_{y,k}$, return the set of tasks $TS \subseteq T_y$, that had to be executed to generate d_x , together with all related input and output data objects of the corresponding task instances. Let the output data d_x be the Literal "resource31", then the sequence of service calls including all involved input and output parameter values may be extracted by the following query:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT ?service ?start_time ?in_literal_value ?out_literal_value
WHERE
{
  {
    ?service_call0 rdf:type m:Service_Call ;
      m:has_output/m:has_resource/m:literal_value "resource31".
    ?service_call0 m:has_predecessor* ?service_call1 .
    ?service_call1 m:has_service ?service ;
      m:start_time ?start_time ;
      m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
      m:has_output/m:has_resource/m:literal_value ?out_literal_value.
  }
}
```

```
ORDER BY ASC(?start_time)
```

Since tasks are used to simply represent process execution steps in our

model, the query returns a sequence of executing services, which is more informative. Starting with the service call that generated "resource31", all preceding service calls are determined by using the transitive has_predecessor property. The workflow instance does not have to be included as a parameter, as only one service call may have "resource31" as output parameter. For each service call, all Literals that where used as input or output parameter values are extracted. The result set is ordered by the start time of service calls.

- PC2: Based on output data d_x and task t_y of workflow instance $wfe_{y,k}$, return the set of tasks $TS \subseteq T_y$, that had to be executed to generate d_x and which where executed before t_y , together with all related input and output data objects of the tasks. Again "resource31" is considered as output data d_x . Starting from the service call that generated "resource31", all preceding service calls are identified. Let the task t_y be the service "service2", then only those preceding service calls are returned that are successors of the service call of "service2". The transitive properties has_predecessor and has_successor are used to filter the service calls. Input and output parameter values are extracted as for PC1. If a task name is used as a restriction criteria, the following property chain can be deployed:

$m : has_successor + /m : has_task /m : task_name.$

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT ?service ?start_time ?in_literal_value ?out_literal_value
WHERE
{
  {
    ?service_call10 rdf:type m:Service_Call ;
      m:has_output/m:has_resource/m:literal_value "resource31".
  }
}

```

```

?service_call0 m:has_predecessor+ ?service_call1 .
?service_call1 m:has_service ?service ;
    m:start_time ?start_time ;
    m:has_successor+/m:has_service/m:service_name "service2" ;
    m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
    m:has_output/m:has_resource/m:literal_value ?out_literal_value.
}
}
ORDER BY ASC(?start_time)

```

- PC3: Based on output data d_x and tasks t_a, t_b, t_c of workflow instance $wfe_{y,k}$, return all related input and output data objects of the tasks. This query is based on filtering dedicated execution steps of a workflow. Assuming that d_x is either the result of one of the specified execution steps or that all execution steps were executed before d_x was generated, the execution steps may be identified by $m : has_predecessor$ properties. d_x is represented by "resource31" in the test data set. The result set is composed of three subqueries for each specified execution step that are combined by a UNION operator. Service names "service1", "service2" and "service3" are used as filter criteria for tasks t_a, t_b, t_c . Alternatively, the execution steps could be determined by task names.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT ?service ?start_time ?in_literal_value ?out_literal_value
WHERE
{
    {
        ?service_call0 rdf:type m:Service_Call ;
            m:has_output/m:has_resource/m:literal_value "resource31".
        ?service_call0 m:has_predecessor+ ?service_call1 .
        ?service_call1 m:has_service ?service ;
    }
}

```

```

        m:start_time ?start_time ;
        m:has_service/m:service_name "service1";
        m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
        m:has_output/m:has_resource/m:literal_value ?out_literal_value.
    }
    UNION
    {
        ?service_call0 rdf:type m:Service_Call ;
        m:has_output/m:has_resource/m:literal_value "resource31".
        ?service_call0 m:has_predecessor+ ?service_call1 .
        ?service_call1 m:has_service ?service ;
        m:start_time ?start_time ;
        m:has_service/m:service_name "service2";
        m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
        m:has_output/m:has_resource/m:literal_value ?out_literal_value.
    }
    UNION
    {
        ?service_call0 rdf:type m:Service_Call ;
        m:has_output/m:has_resource/m:literal_value "resource31".
        ?service_call0 m:has_predecessor* ?service_call1 .
        ?service_call1 m:has_service ?service ;
        m:start_time ?start_time ;
        m:has_service/m:service_name "service3";
        m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
        m:has_output/m:has_resource/m:literal_value ?out_literal_value.
    }
}
ORDER BY ASC(?start_time)

```

- PC4: Find all executions of task t_i in which the constant value *const* was used as an input parameter. Return the task instances and all

associated data. Assuming task t_i is identifiable by its task name, all executions of the task may be detected by consequently reading all service calls that are associated with the task. In the test data set, all executions of task "TaskB" having the constant input parameter value "resource20" are extracted. The result set is a list of service calls with all related parameter values.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT ?service_call ?in_literal_value ?out_literal_value
WHERE
{
  {
    ?task m:has_service ?service ; m:task_name "TaskB" .
    ?service_call m:has_service ?service ;
      m:has_input/m:has_resource/m:literal_value "resource20" ;
      m:has_input/m:has_resource/m:literal_value ?in_literal_value ;
      m:has_output/m:has_resource/m:literal_value ?out_literal_value.
  }
}

```

- PC5: Find all generated output data (excluding intermediate data) that was produced by workflow instances consuming data objects of type ty_{ident} . The semantic provenance model offers two conceptual types for resources. There are data types of resources and semantic types that may be assigned to resources by annotations. The following query shows how resources are selected based on an annotation type "ident". The query returns all generated Literal values of process executions that use at least one resource of annotation type "ident" as an input parameter value. As intermediate results should be eliminated, all resources that are used as input values in other service calls

are excluded. Starting from the service call reading a resource of type "ident", the query collects output parameter values from preceding and succeeding service calls.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT DISTINCT ?out_literal_value
WHERE
{
  {
    ?service_call m:has_input/m:has_resource ?resource ;
      m:has_predecessor* ?p_service_call.
    ?resource m:has_annotation/m:annotation_text "ident".
    ?p_service_call m:has_output/m:has_resource ?r_out.
    ?r_out m:literal_value ?out_literal_value.
    FILTER NOT EXISTS {?p_service_call1 m:has_input/m:has_resource ?r_out.}
  }
  UNION
  {
    ?service_call m:has_input/m:has_resource ?resource ;
      m:has_successor* ?p_service_call.
    ?resource m:has_annotation/m:annotation_text "ident".
    ?p_service_call m:has_output/m:has_resource ?r_out.
    ?r_out m:literal_value ?out_literal_value.
    FILTER NOT EXISTS {?p_service_call1 m:has_input/m:has_resource ?r_out.}
  }
}

```

- PC6: Find all data objects that were created by executing task t_j , that was preceded directly or indirectly by an execution of task t_i with input parameter *const*. Let t_i be a task with name "TaskA" and t_j task "TaskC". The constant value is represented by Literal "resource10". Firstly, all executions of "TaskA" having "resource10" as

an input parameter value are detected. Then all executions of "TaskC" that occurred after the filtered "TaskA" executions are identified and the generated output parameter values are extracted.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT DISTINCT ?out_literal_value
WHERE
{
    ?service_call m:has_input/m:has_resource/m:literal_value "resource10" ;
        m:has_task/m:task_name "TaskA" ;
        m:has_successor+ ?s_service_call.
    ?s_service_call m:has_task/m:task_name "TaskC" ;
        m:has_output/m:has_resource ?r_out .
    ?r_out m:literal_value ?out_literal_value .
}

```

- PC7: Two similar workflows wf_y and wf_z are executed by workflow instances $wfe_{y,k}$ and $wfe_{z,l}$. Both workflows share a set of tasks of the same type and both workflow instances operate on a common set of data objects. Find the set of different tasks of both workflows, and identify all data objects that are present in one of the workflow instances but not in both. Execution of workflows are represented by Process_Instances in the semantic model. In the following query, Process_Instances example.Process.1.1345122394386 and Process.2.1345122394387 are compared. Tasks of the same type are modelled by services with equal annotations ("ident"). As SPARQL does not support symmetric differences of sets, different tasks and resources must be detected by nested MINUS and UNION operators. The first subquery extracts all services and accessed resources that are present in the first process instance but not in the second. In the second subquery, a converse selection

is executed. Finally, a UNION operator combines the results of both queries.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT ?service ?i_value ?o_value
WHERE
{
  {
    {
      m:example.Process_1.1345122394386 m:starts_with ?s_service_call.
      ?s_service_call m:has_successor* ?f_service_call .
      ?f_service_call m:has_service ?service .
      ?f_service_call m:has_input/m:has_resource/m:literal_value ?i_value .
      ?f_service_call m:has_output/m:has_resource/m:literal_value ?o_value .
      ?service m:has_annotation/m:annotation_text "ident".
    }
  }
  MINUS
  {
    {
      m:example.Process_2.1345122394387 m:starts_with ?s1_service_call.
      ?s1_service_call m:has_successor* ?f1_service_call .
      ?f1_service_call m:has_service ?service .
      ?f1_service_call m:has_input/m:has_resource/m:literal_value ?i_value .
      ?f1_service_call m:has_output/m:has_resource/m:literal_value ?o_value .
      ?service m:has_annotation/m:annotation_text "ident".
    }
  }
}
UNION
{
  {
    {
      m:example.Process_2.1345122394387 m:starts_with ?s1_service_call.
      ?s1_service_call m:has_successor* ?f1_service_call .
      ?f1_service_call m:has_service ?service .
      ?f1_service_call m:has_input/m:has_resource/m:literal_value ?i_value .
```

```

        ?f1_service_call m:has_output/m:has_resource/m:literal_value ?o_value .
        ?service m:has_annotation/m:annotation_text "ident".
    }
MINUS
{
    m:example.Process_1.1345122394386 m:starts_with ?s_service_call.
    ?s_service_call m:has_successor* ?f_service_call .
    ?f_service_call m:has_service ?service .
    ?f_service_call m:has_input/m:has_resource/m:literal_value ?i_value .
    ?f_service_call m:has_output/m:has_resource/m:literal_value ?o_value .
    ?service m:has_annotation/m:annotation_text "ident".
}
}
}

```

- PC8: Find all generated data objects that were annotated with key_1 and text $annot_A$ and which were derived from input data that was annotated with key key_2 and text $annot_B$. Key-value annotations are modelled by data properties `annotation_key` and `annotation_text`. Starting with all resources that were used as input parameter values and were annotated with key "key2" and annotation value "annotationB", all involved service calls are extracted. Then, for each service call, a succeeding service call is searched that generated a resource annotated with key "key1" and annotation value "annotationB".

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT ?out_literal_value
WHERE
{
    ?service_call m:has_input/m:has_resource ?i_resource ;
        m:has_successor* ?p_service_call.

```

```

    ?i_resource m:has_annotation/m:annotation_key "key2";
        m:has_annotation/m:annotation_text "annotationB" .
    ?p_service_call m:has_output/m:has_resource ?o_resource .
    ?o_resource m:has_annotation/m:annotation_key "key1";
        m:has_annotation/m:annotation_text "annotationA" ;
        m:literal_value ?out_literal_value .
}

```

- PC9: Find all data objects of type ty_{ident} that have been annotated with key_1 and text $annot_A$ and return all existing annotations of the objects. The query returns all resources that were annotated with key "key1" and annotation value "annotationA" and have the semantic type "ident". Since annotation keys are not mandatory in annotations, the annotation_key property is embedded into an OPTIONAL clause.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT ?literal_value ?annotation_key ?annotation_text
WHERE
{
    ?resource rdf:type m:Literal ; m:has_annotation/m:annotation_key "key1" ;
        m:has_annotation/m:annotation_text "annotationA" ;
        m:has_annotation/m:annotation_text "ident" ;
        m:literal_value ?literal_value .
    ?resource m:has_annotation ?annotation .
    ?annotation m:annotation_text ?annotation_text .
    OPTIONAL{?annotation m:annotation_key ?annotation_key } .
}

```

5.6.3 Answering Use Case Driven Data Provenance Queries

The following set of GATiB-specific provenance queries has already been published at the DEXA 2010 [96]. In the first example, all contexts (knowledge spaces), in which a certain resource *document_1* is shared, should be returned. However, we are only interested in contexts which have an associated annotation text containing “liver cancer” or which have subcontexts with this annotation. SPARQL offers path-oriented filter patterns for graphs allowing to select elements along predefined traversal paths. Since contexts may be nested into each other up to an arbitrary depth, all subcontexts of a context may be identified by the inverted $\hat{m} : is_subcontext^*$ property pattern. No explicit reasoning step is executed in this case, as only the transitive closure of a single property is required, which may be determined by the lightweight internal reasoner of Jena. Alternatively, an external reasoner can be applied to infer all *is_subcontext* relationships [96].

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
SELECT ?context ?annotation_text
WHERE
{
    ?context ^m:is_subcontext*/m:has_resource ?resource .
    ?resource m:literal_value "document_1" .
    ?context m:has_annotation/m:annotation_text ?annotation_text .
    FILTER regex(?annotation_text, "liver_cancer", "i")
}
```

When working on documents collaboratively, several versions of documents may be created by cooperating partners. While different document versions are retained as backup and for tracking the development process, they may

also be shared in related contexts. For instance, in medical studies collecting life style and follow-up data for patients can be cumbersome. Life style data is collected in questionnaires, and disease treatment data is retrieved from various institutes: oncology, surgery, external hospitals etc.

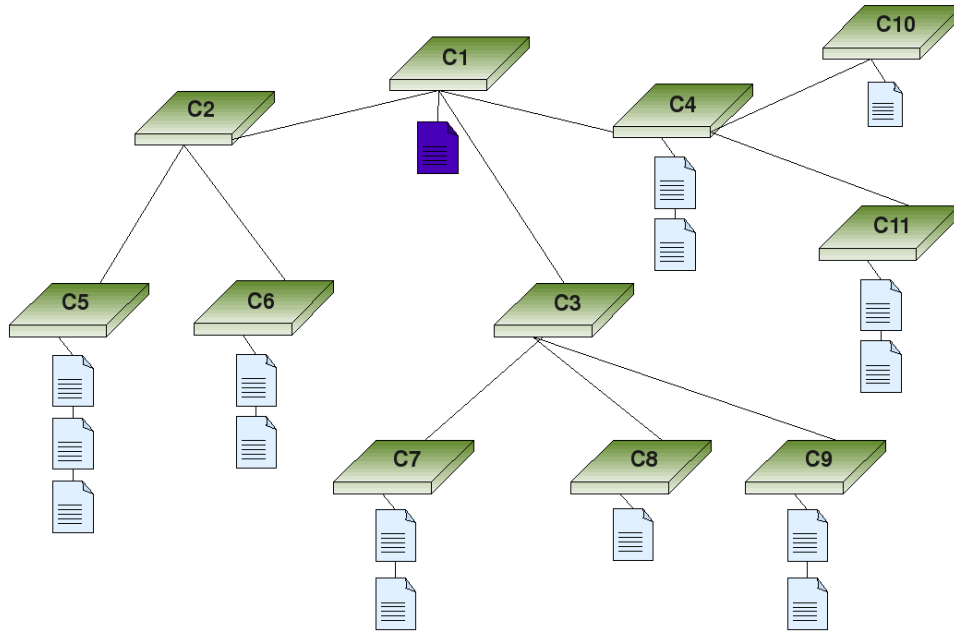


Figure 36: Different Versions of a Resource in Various Contexts

The collected data is precious for medical research and it can be shared and reused in several related studies. If follow-up data of a patient collective suffering from mamma carcinoma was collected, various breast cancer research projects may benefit from the data. Further, each project may create own versions of the follow-up data by including additional parameters and by adding further treatment data. If a medical scientist wants to find out how the original follow-up data was extended in related projects, he should be presented all consecutive versions of the follow-up document, as illustrated in Figure 36. The corresponding SPARQL query returns an ordered list of

document versions of related contexts. Note that there is a restriction on the transitive *related_context* property filtering out contexts that are either immediately related to the original context of the document or may be reached by a two-level *related_context* relationship. Thus, the search space is pruned filtering contexts that are semantically close to each other [96].

```

PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  m: <http://www.sem_model.org#>
PREFIX  sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT  ?context ?document_name
WHERE
{
    m:C1 m:related_context{1,2} ?context .
    ?context m:has_resource ?resource .
    ?resource m:is_subversion+ ?base_document ;
        m:literal_value ?document_name.
    ?base_document m:literal_value "document_2" .
    ?resource m:version_nr ?version_number.
}
ORDER BY ASC(?version_number) ?context

```

Process- and context-related data may be combined in complex queries. Consider three research projects in the field of breast cancer research, depicted in Figure 37. In projects “Mamma Carcinoma” and “Breast Cancer” the gene expression profiles of mamma carcinoma patients are analyzed by the bioconductor software packages ‘Global Test’ [38] and ‘Global Ancova’[44] which are executed as web services. These tools identify a set of functional gene groups which may influence the course of disease. Each functional gene group is described by an entry of the Gene Ontology, which is a categorization system for biological processes, cellular components and molecular functions. Within the third project “Breast Cancer Research”, several publications in the area of breast cancer research are collected and reviewed, whereas rele-

vant gene groups are extracted from the papers and added as annotations. If a medical scientist is interested in all gene groups that are associated with breast cancer, a query returning both the analysis results and the manual annotations may be formulated. The corresponding query searches within the analysis results of services of type *Analysis_Service* and combines the results with annotations containing Gene Ontology groups. The input files of the services have to be annotated with “Mamma Carcinoma” as well as the publications [96].

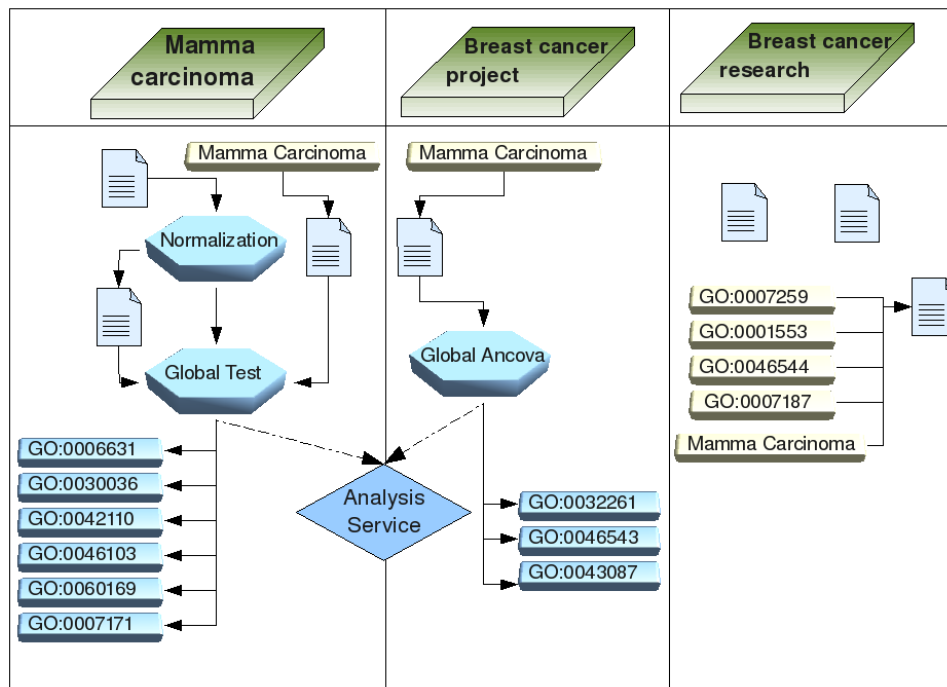


Figure 37: Gene Ontologies as Service Call Outputs or Annotations

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
```

```

PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT DISTINCT ?context ?document_name ?gene_group
WHERE
{
    {
        ?context rdf:type m:Context;
        m:has_resource ?resource .
        ?resource m:has_annotation/m:annotation_text ?annotation_text ;
            m:literal_value ?document_name .
        FILTER regex(str(?annotation_text), "Mamma_Carcinoma") .
        ?service_call m:has_input/m:has_resource ?resource ;
            m:has_service ?service ;
            m:has_output ?output_parameter .
        ?service m:has_annotation/m:annotation_text "Analysis_Service" .
        ?output_parameter m:has_resource ?object_collection .
        ?object_collection m:has_member/m:literal_value ?gene_group .
    }
    UNION
    {
        ?context rdf:type m:Context;
        m:has_resource ?resource .
        ?resource m:has_annotation/m:annotation_text ?annotation_text;
            m:literal_value ?document_name .
        FILTER regex(str(?annotation_text), "Mamma_Carcinoma") .
        ?resource m:has_annotation/m:annotation_text ?gene_group .
        FILTER regex(?gene_group, "GO:") .
    }
}
ORDER BY ?context

```

5.6.4 Provenance Model - Java Programming Interface

A set of maintenance operations for the provenance model have been realized by implementing a Java framework that offers various methods for recording provenance data, execute provenance queries and handle changes in provenance data. Based on Apache Jena, the Java framework allows to fill the provenance model by API calls and is capable of answering prove-

nance queries formulated as SPARQL queries. The entire Java code has been released in the Sourceforge project Semantic Provenance Model <http://sourceforge.net/p/sem-prov-model/>. Further, an OWL-DL representation of the model is integrated in the project. Generally, the implemented Java Programming Interface is a lightweight, standalone component that may be integrated in other scientific workflow management systems as a provenance capturing tool. It may be considered as a proof-of-concept implementation showing the capabilities of the semantic provenance model.

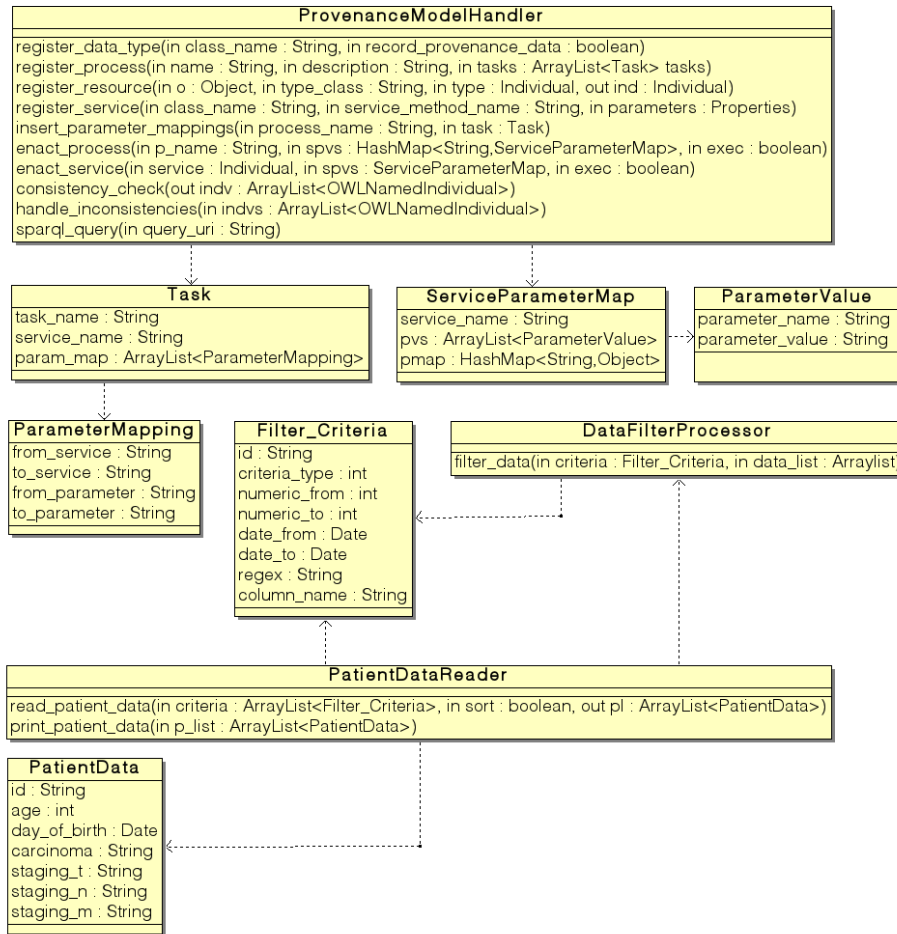


Figure 38: Provenance Model - Java API

The framework may be run in a demonstration mode, in which scientific processes consisting of web service calls are defined and executed. However, the execution of web services is limited to Java classes found in the classpath of the application. A simulated database interface was realized that returns Java objects based on passed data filter criteria. This section presents a continuous example for registering and executing a simple scientific process and capturing all generated provenance data.

In Figure 38, the main classes of the framework are illustrated in a UML class diagram. The ProvenanceModelHandler is the central component for interaction with the provenance model. It provides all relevant data access and manipulation methods for creating new ontology classes and instances and answering SPARQL-queries. Classes Task and ParameterMapping are internal data structures for defining process definitions. Enactment of processes and services requires the use of data structures of classes ServiceParameterMap and ParameterValue. Classes Filter_Criteria and DataFilterProcessor are utility classes for formulating selection criteria on data resources. In the presented example, patient data records are filtered by their attribute values patient age, tumor staging and day of birth. Classes PatientDataReader and PatientData have been integrated for demonstration purposes and are not required to run the framework. PatientDataReader offers a service which simulates a data extraction out of a fictive patient record database and another service for printing the result set of the extraction. Finally, PatientData is a simple Java Bean class encapsulating attributes of a patient data database table.

Reading and Filtering Example Record Set: The presented use case is a simple scientific workflow consisting of two tasks. In the first task a set of patient records is extracted from the database interface, whereas filter criteria are used for restriction. The resulting records are passed to the second task which is responsible for a pretty-formatted output of the record set. A BPMN representation of the process is given in Figure 39. Using the example

of this short workflow, access to data resources, parameter value passing and handling of data collections are illustrated. The patient data reader task is realized by service `read_patient_data` of class `PatientDataReader`. A set of filter criteria is passed to service `read_patient_data` together with a parameter specifying whether the result set should be sorted. Printing of patient records is accomplished by service `print_patient_data` of the same class.

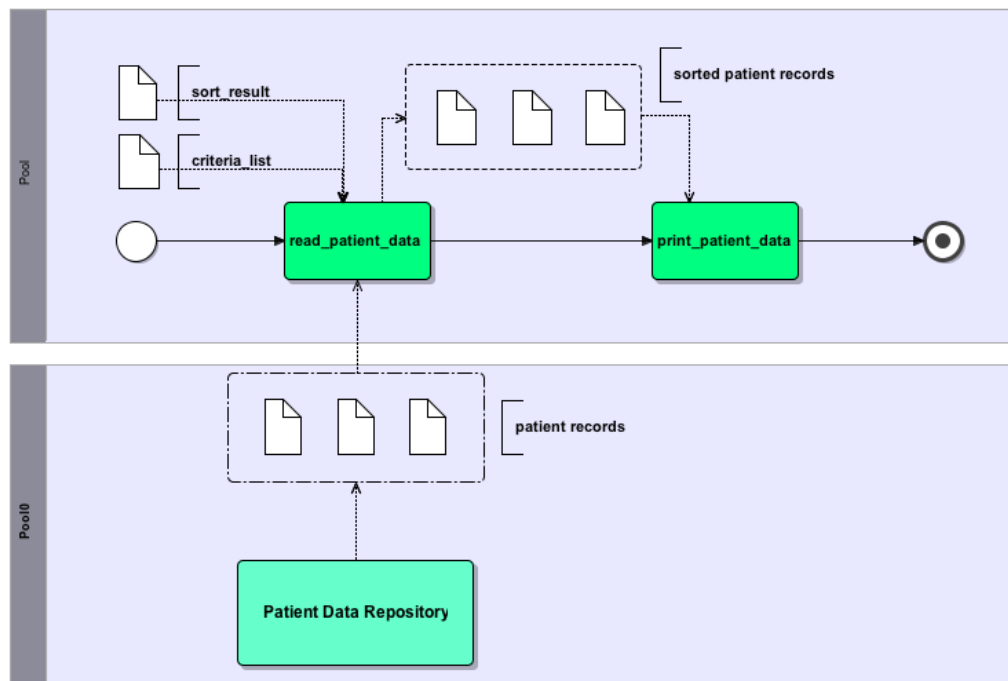


Figure 39: Example Process as BPMN

Figure 40 presents a set of three patient records. Each record has a unique ID and various person- and disease related attributes. Filter criteria for data selection may be defined on all attributes. Currently, the following types of filter criteria are supported:

- Numeric intervals - Based on lower and upper bounds of a numeric interval, all matching entries are returned.

- Date intervals - Lower and upper bounds of a date interval are used to restrict the result set of entries.
- String patterns - All entries containing the defined String pattern are included in the result set. The selection is equivalent to the SQL LIKE `% [pattern]` command.

Additional filter criteria for data types boolean, float and double can be easily integrated by extending class `Filter_Criteria` and adding further private fields.

Person-related			Disease-related			
ID	Age	Day_of_Birth	Carcinoma	Staging T	Staging N	Staging M
p3	50	1962-02-05	Mamma	T1	N1	M0
p4	49	1963-04-11	Mamma	T2	N2	M1
p5	48	1964-06-01	Mamma	T1	N2	M1

Figure 40: Example Patient Record Set

Code Listing 15 shows how service `read_patient_data` and `print_patient_data` may be executed as standalone services. After initializing a new `Patient-DataReader` object, a numeric filter criteria is generated by the static method `create_criteria` of class `Filter_Criteria`. A `Properties` object is used to encapsulate parameter values required for the filter criteria. In the present example, all patient records having age values in the interval `[49,51]` should be filtered. The created filter criteria is added into a list of filter criteria and passed to service `read_patient_data`. If more than one filter criteria is defined, all filter criteria are evaluated by using logical AND operators. Filter criteria that are connected by OR operators are currently not supported. Service `read_patient_data` reads all patient records from the simulated patient data repository and extracts all records matching the age filter criteria. In our example, patient records p3, p4 und p5 are extracted. The resulting

record set is buffered in an ArrayList of PatientData and passed to the printing service print_patient_data. In Listing 16, an example for a date-based filter criteria is given. All patient records having a day of birth between [1962 – 01 – 01, 1963 – 12 – 31] are extracted.

Listing 15: Read and Print Patient Data

```
PatientDataReader patient_reader = new PatientDataReader();
Properties properties = new Properties();
properties.put("column_name", "age");
properties.put("numeric_from", "49");
properties.put("numeric_to", "51");
Filter_Criteria criteria =
    Filter_Criteria.create_criteria(
        properties, Filter_Criteria.NUMBER_CRITERIA);
ArrayList<Filter_Criteria> criteria_list =
    new ArrayList<Filter_Criteria> ();
criteria_list.add(criteria);
ArrayList<PatientData> p_list1 =
    patient_reader.read_patient_data(criteria_list, false);
patient_reader.print_patient_data(p_list1);
```

Listing 16: Filter Criteria Defined by Date Intervals

```
properties = new Properties();
properties.put("column_name", "day_of_birth");
properties.put("date_from", "1962-01-01");
properties.put("date_to", "1963-12-31");
criteria = Filter_Criteria.create_criteria(
    properties, Filter_Criteria.DATE_CRITERIA);
criteria_list = new ArrayList<Filter_Criteria> ();
criteria_list.add(criteria);
```

Finally, in Listing 17 a filter criteria is defined by a String pattern. All patient records having a tumor staging value of T1 are extracted.

Listing 17: Filter Criteria Defined by String Patterns

```
properties = new Properties();
properties.put("column_name", "staging_t");
properties.put("regex", "T1");
criteria = Filter_Criteria.create_criteria(
    properties, Filter_Criteria.STRING_CRITERIA);
criteria_list = new ArrayList<Filter_Criteria> ();
criteria_list.add(criteria);
```

Register structured data types: Before a process template may be defined, all involved services and structured data types have to be registered. Simple data types such as integer or boolean are already included in the provenance model. A structured data type maps a database table or view to an object-oriented representation. The provenance framework uses Java Beans to represent structured data types. Alternatively, Structured_Types may be defined using XML schema complex types. In this case, XSD:Attributes are mapped to Attribute_Types and the built-in XSD data types are represented by appropriate Simple_Types. A Java Bean may be registered as a structured data type, if it has a private attribute id that has a String data type. The id attribute is used as unique identifier for distinguishing individuals of the ontology class Structured_Data_Type. All attributes of the Java Bean must have primitive data types or types String or Date. Attributes with other reference types are not supported.

The registration of new structured data types is accomplished by method `register_data_type` of class `ProvenanceModelHandler`. The method takes the class name of the Java Bean as an input and uses Java Reflection for dynamically reading the definitions of all declared private attributes. The Java Bean class must be included in the class path of the application, so it can be found by Java Reflection methods. A new `Structured_Data_Type` individual is instantiated and for every found attribute a new `Attribute_Type` individual is generated and linked with the new type. The second parameter `record_provenance_data` indicates, whether provenance data should be

recorded for resources of the type. Thus, resources that are not required for provenance queries can be excluded. An example call of method `register_data_type` is given in Listing 18. A new `ProvenanceModelHandler` object is generated and the `StructuredData.Types` `example.PatientData` and `util.Filter.Criteria` are registered. `Record_provenance_data` is enabled for both types. The generated OWL individuals and properties are illustrated in Figure 41. The class name is used as an identifier for the new `StructuredData.Type` individual. `Attribute.Type` individuals are identified by a concatenation of class name and attribute name.

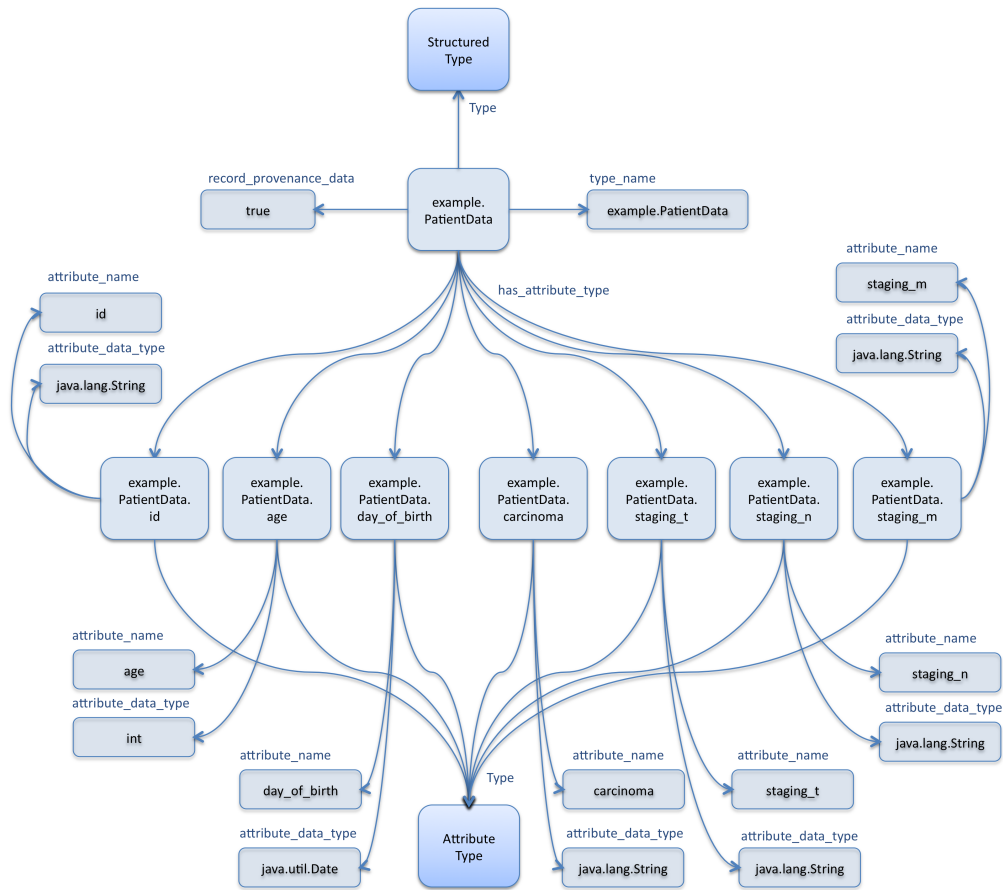


Figure 41: Provenance Model - Register Structured Data Type

Listing 18: Register Structured Data Type

```
ProvenanceModelHandler model_handler = new ProvenanceModelHandler();  
model_handler.register_data_type("example.PatientData",true);  
model_handler.register_data_type("util.Filter_Criteria",true);
```

Register service: Services are realized by either local Java services or remote web services provided by arbitrary frameworks. If remote web services are used, the service URI is required and parameters types have to be extracted from WSDL definitions. XSD types are mapped to Simple_Types and XSD complex types to Structured_Types and Collection_Types. For local Java services, the corresponding class name and service name have to be specified. The prerequisite for a successful service registration is that all related Simple_Types and Structured_Types of service parameters are present in the model. Collection_Types are dynamically added if they can not be found. Collection_Types are used to model lists of objects that are passed to or generated by service calls. Currently, Collection_Types are represented by typed ArrayLists. Alternative data structures such as Vectors, HashMaps or LinkedLists will be integrated in future releases. Object collection types in remote services are specified by XML schema complex types, having the maxOccurs attribute of an embedded element set to unbound. These collection types may be defined by firstly registering the nested type in the XSD complex type and then creating a Collection_Type for the complex type.

Services may be registered in the provenance model by using the method `register_service` of class `ProvenanceModelHandler`. In Listing 19, the API calls for registering services `read_patient_data` and `print_patient_data` are presented. After instantiating a new `ProvenanceModelHandler` object, a `Properties` object is generated. For each input parameter of the service, a new property mapping the parameter name to the parameter type is added to the property list. `Object_Collection_Types` are notated as `ArrayList(Structured_Type type_name)`. `Simple_Types` and `Structured_Types` are defined by their `type_name`. Output parameters are specified by adding

a property return $\mapsto$ type. If the service does not deliver a return value, the return property is omitted. In our example, service `read_patient_data` has a `criteria_list` of type `ArrayList(util.Filter.Criteria)` and `sort_result` of type `boolean` as input parameters. The service returns an `Object_Collection` of type `ArrayList(example.PatientData)`. After registering the `read_patient_data` service, a set of individuals and properties has been inserted into the model, as shown in Figure 42. A new Service individual and FormalParameters for all parameters have been created. Each FormalParameter is connected with its associated Type. Local Services are identified by a concatenation of class name and method name, remote Services by concatenating their URI with the WSDL service name. The URI of FormalParameters is composed of service name and the parameter name.

Listing 19: Register Service

```
ProvenanceModelHandler model_handler = new ProvenanceModelHandler();
Properties property_list = new Properties();
property_list.put("return", "ArrayList(example.PatientData)");
property_list.put("criteria_list", "ArrayList(util.Filter.Criteria)");
property_list.put("sort_result", "Boolean");
model_handler.register_service("example.PatientDataReader",
    "read_patient_data", property_list);
property_list = new Properties();
property_list.put("p_list", "ArrayList(example.PatientData)");
model_handler.register_service("example.PatientDataReader",
    "print_patient_data", property_list);
```

Register Process: A process template consists of a sequence of tasks which are realized by service calls at process runtime. While control flow is determined by the order of the task sequence, data flow has to be modelled separately. Output parameter values, generated by service calls, may be passed as input parameter values of subsequent service calls. The registration of the example process 'Patient_Data_Processor' is illustrated in code Listing 20. Since the list of PatientData records is passed from service

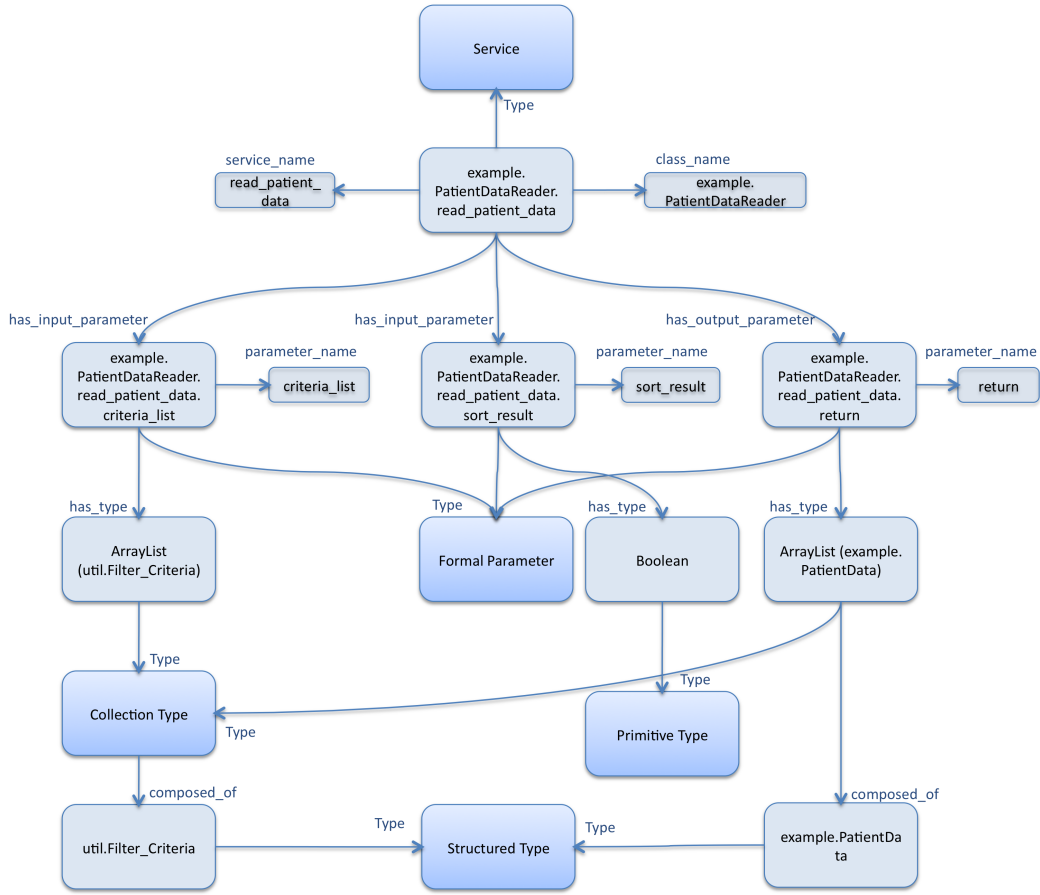


Figure 42: Provenance Model - Register Service

read_patient_data to service print_patient_data, the data flow has to be modelled by a ParameterMapping object. This object specifies that the output parameter value read_patient_data.return of the reading service is used as the input parameter value print_patient_data.p_list of the printing service. As the output value of a service may be used as input of multiple services, all defined parameter mappings are encapsulated in a list of mappings (parameter_mappings). Task objects represent single tasks of a process and are initialized with a task label ('Task1'), the executing service and the list of parameter mappings. After defining both tasks t1 and t2, they are added into a

task list (tasks). The registrations occurs by calling method `register_process` of class `ProvenanceModelHandler`. The method takes a unique process identifier, a process description and the list of tasks as inputs. Figure 43 presents the results of the process registration. A new `Process` individual and two `Task` individuals have been created. The `Process` individual references the `Task` individuals by `initial_task` and `end_task` properties. Further, predecessor and successor relationships model the execution order and both `Task` individuals are connected with the already registered `Service` individuals. Finally, a new `Parameter_Mapping` individual was generated, referencing the reading and the printing service as well as the corresponding `Formal_Parameters`.

Listing 20: Register Process

```
ParameterMapping parameter_mapping = new ParameterMapping(
    "example.PatientDataReader.read_patient_data",
    "example.PatientDataReader.print_patient_data",
    "example.PatientDataReader.read_patient_data.return",
    "example.PatientDataReader.print_patient_data.p_list"
);
ArrayList<ParameterMapping> parameter_mappings
    = new ArrayList<ParameterMapping>();
parameter_mappings.add(parameter_mapping);
Task t1 = new Task("Task1",
    "example.PatientDataReader.read_patient_data",parameter_mappings);
Task t2 = new Task("Task2",
    "example.PatientDataReader.print_patient_data",parameter_mappings);
parameter_mappings = new ArrayList<ParameterMapping>();
    "example.PatientDataReader.print_patient_data",parameter_mappings);
ArrayList<Task> tasks = new ArrayList<Task>();
tasks.add(t1);
tasks.add(t2);
model_handler.register_process("Patient_data_processor",
    "Reads_Patient_records_and_prints_them", tasks);
```

Record Provenance Data - Enact Process: When executing predefined process templates, provenance data may be captured by using API calls

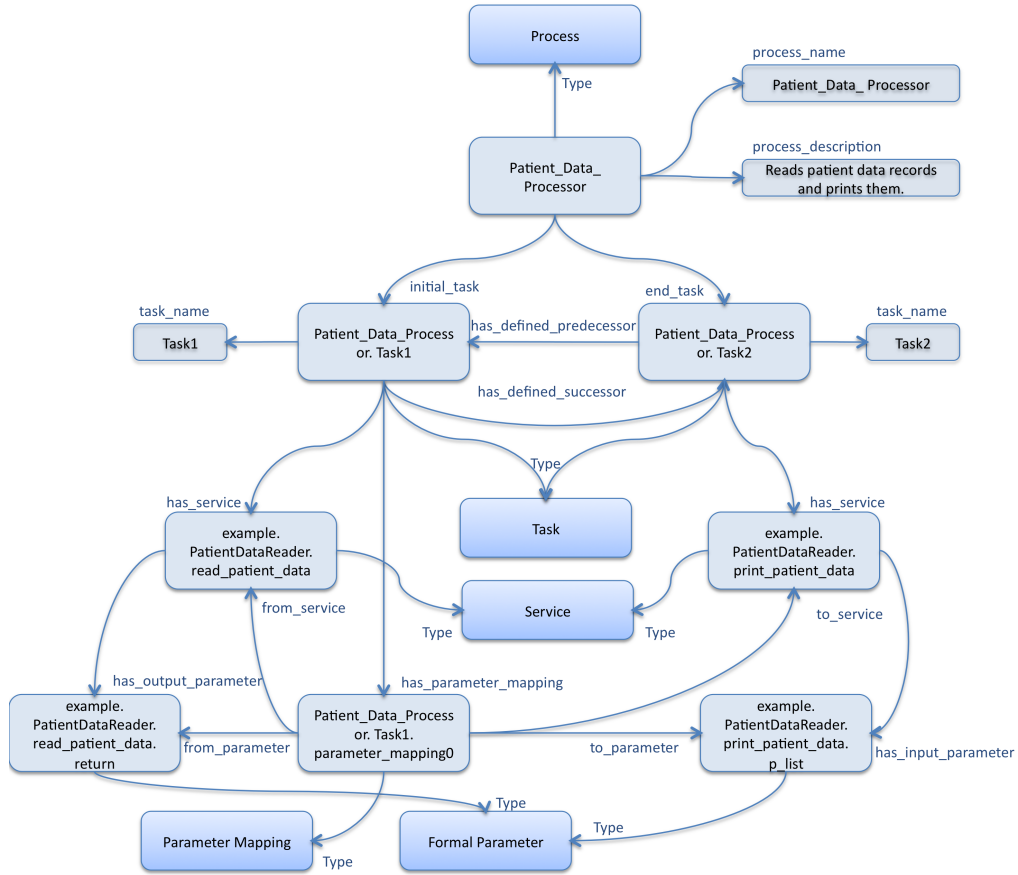


Figure 43: Provenance Model - Register Process

of the provenance data framework. A new Process instance is generated and all defined execution steps are realized by Service.Calls. Provenance data of process execution comprises several types of data. On the one hand, process-related data is represented by recording the execution order of Services and logging all involved Service parameter values. On the other hand, all Resource access operations of Service.Calls are documented. For instance, if a Service.Calls reads a certain record of PatientData, a reference from the Service.Call to the record is logged. Further, all Resources, that have been generated by a Service.Call, are linked to the Service.Call. That is, the entire

data flow during process execution becomes traceable. The 'production plan' for generated Resources (intermediate results) may be derived by identifying the Service_Call that references the Resource as an output parameter and extracting all preceding Service_Calls and related parameter values of the same Process_Instance. By filtering all Service_Calls that use a Resource as an input parameter, the set of Process_Instances that depends on the same Resource may be detected.

Listing 21: Enact Process

```
ArrayList<ParameterValue> p_vs = new ArrayList<ParameterValue>();
Properties properties = new Properties();
properties.put("column_name", "age");
properties.put("numeric_from", "49");
properties.put("numeric_to", "51");
Filter_Criteria criteria = Filter_Criteria.create_criteria(
    properties, Filter_Criteria.NUMBER_CRITERIA);
ArrayList<Filter_Criteria> criteria_list =
    new ArrayList<Filter_Criteria> ();
criteria_list.add(criteria);
ParameterValue pv1 =
    new ParameterValue("criteria_list", criteria_list);
ParameterValue pv2 =
    new ParameterValue("sort_result", new Boolean(false));
p_vs.add(pv1);
p_vs.add(pv2);
ServiceParameterMap service_pvs = new ServiceParameterMap(
    "example.PatientDataReader.read_patient_data", p_vs);
HashMap<String, ServiceParameterMap> map_service_pvs =
    new HashMap<String, ServiceParameterMap>();
map_service_pvs.put("example.PatientDataReader.read_patient_data",
    service_pvs);
model_handler.enact_process("Patient_data_processor", map_service_pvs, true);
```

The provenance framework comes with a built-in process execution engine allowing to enact processes consisting solely of local Java services. Typically,

the provenance framework should be deployed as a provenance recording system that obtains process and service execution data from a workflow execution engine. The passed execution data is then transformed into entries of the provenance model. Code Listing 21 shows how the example process may be enacted by passing actual input parameter values for all services present in the process definition. Method `enact_process` of class `ProvenanceModelHandler` allows to capture provenance data for the execution of a process. The first parameter specifies the identifier of the process definition. A map of parameter values of all involved services is passed as the second parameter. The third parameter (boolean `execute`) defines, whether the process instance should be executed or not. In our example, the process is composed of local Java services and can therefore be executed. If `execute` is set to false, the service parameter map must include all output parameter values generated by an already completed service execution. Starting at the beginning of the code listing, the input parameter values of service `read_patient_data` are specified in `ParameterValue` objects. Parameter `criteria_list` contains a list of `Filter_Criteria`, while parameter `sort_result` contains the boolean value false. Since the input parameter `p_list` of service `print_patient_data` is filled with the output of service `read_patient_data`, no input parameter for the second service has to be defined. All parameter values of a service are encapsulated in a `ServiceParameterMap` object which in turn is added into a `HashMap` of `ServiceParameterMaps` (`map_service_pvs`). This `HashMap` is passed to method `enact_process`.

After executing method `enact_process`, new provenance data is inserted into model as shown in Figure 44. A new `Process_Instance` individual is created which is based on the process definition of process `Patient_Data_Processor`. The URI of the `Process_Instance` is composed of the process name followed by a timestamp in milliseconds. Following the process definition, service `read_patient_data` is executed for `Task1` and service `print_patient_data` for `Task2`. The execution of services is accomplished using method `enact_service`

which takes a service object, the service parameter map and an execute flag as inputs. If the execute flag is set, the corresponding local Java service is executed using the Java Reflection library. Service_Call individuals are generated for each executed Service and identified by the service name concatenated with '.call.' and a timestamp. For all available input and output parameter values new Actual_Parameter individuals are generated and connected with has_input and has_output properties. Timestamps are used to generate unique identifiers for Actual_Parameters. All Actual_Parameter individuals are linked to appropriate Formal_Parameters. Parameter values are represented by Literals, Structured_Objects and Object_Collections. In our example, the input parameter values for Service read_patient_data are an Object_Collection for the list of Filter_Criteria and a boolean Literal for the sort_result flag. The output parameter of read_patient_data is an Object_Collection of PatientData objects. The returned Patient_Data collection is used as an input parameter for Service print_patient_data. For lack of space, details of resources collection.1345122394430, Literal.1345122394430 and collection.1345122394427 are omitted from Figure 44. In Figure 45, the registration of object collection collection.1345122394427 is illustrated. Generally, Structured_Objects are registered the first time they are accessed by a Service_Call. Since they are uniquely identified by their id attribute, no duplicates of Structured_Objects are present in the provenance model. Literals and Object_Collections are generated for each Service_Call separately. The URI of a Literal is composed of 'Literal.' and a timestamp. Similarly, URIs for Object_Collections are created by concatenating 'Collection.' with a timestamp. The depicted Object_Collection consists of two Structured_Objects (PatientData records p3 and p4). Two new Structured_Object individuals are created based on the type of example.PatientData. For all attribute values, Attribute_Value individuals are generated and linked by has_attribute_value properties to Structured_Objects p3 and p4. The literal Attribute_Values are connected by has_value data properties.

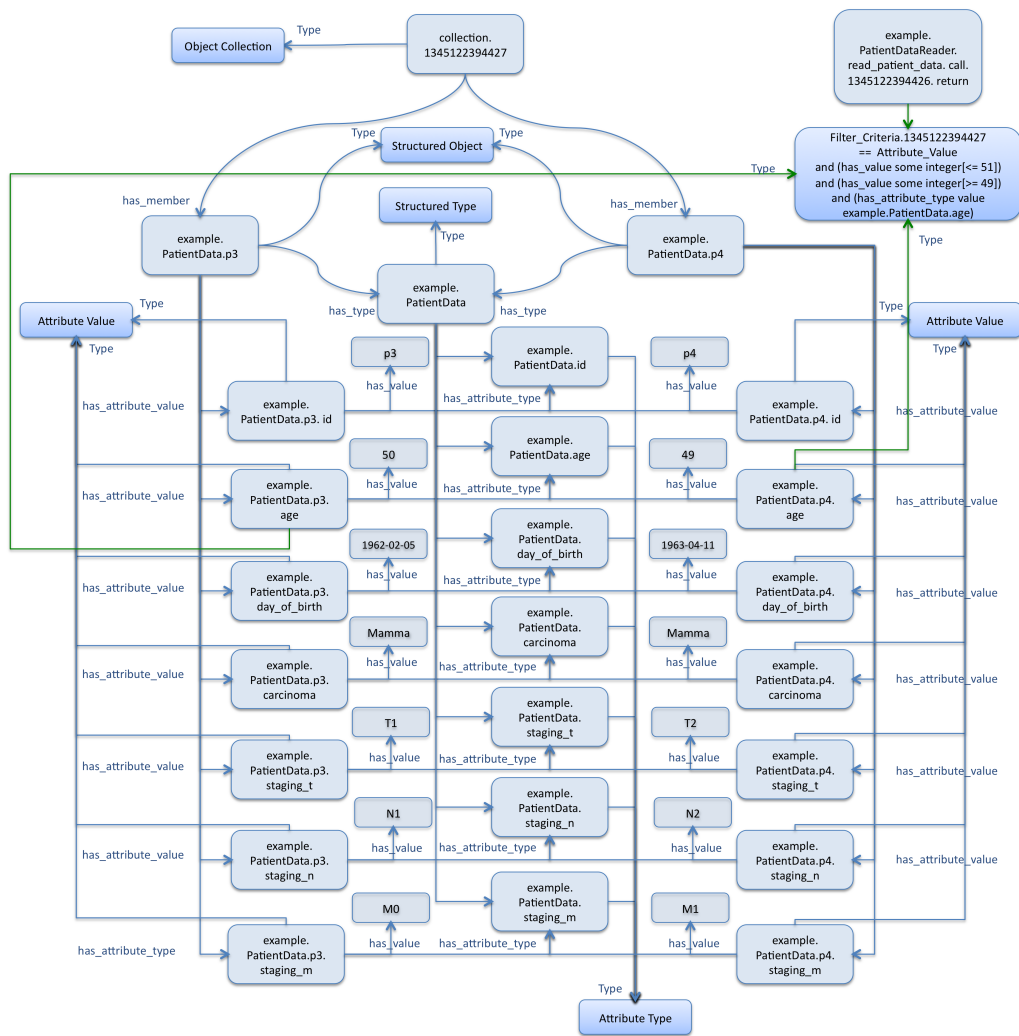


Figure 45: Provenance Model - Register Resource

One of the key requirements of the provenance model was to handle changes in input data of processes and to detect process results that are based on invalid input data. Therefore, OWL equivalence classes are used for defining validity constraints on input data. If the validity constraints are violated by data changes, an OWL reasoner is able to detect the inconsistencies and identifies all responsible individuals. If resources were selected based on filter criteria, a new `Filter_Criteria` class is created and all involved `Attribute_Value` individuals are defined as members of the class. The new `Filter_Criteria` class is defined as an equivalence class of `Attribute_Value` (or `Structured_Object`) and the selection criteria is modelled by using the Manchester OWL Syntax [43].

`Object_Collection.collection.1345122394430` is the result of a filtering `PatientData` records in which the patient age is greater than 49 and smaller than 51. A new class `Filter_Criteria.1345122394427` is created which individuals are of Type `Attribute_Value` and are assigned to `Attribute_Type.example.PatientData.age`. Further, the `Attribute_Values` refer to integer literals in the interval [49, 51].

5.6.5 Detection and Handling of Data Changes

Changes in data resources, that are used as input data in scientific workflows, may have considerable impacts on the quality and validity of generated results. Invalid output data of scientific workflows may cause biased analysis results and impede subsequent research activities. Therefore, management of data changes was a key aspect in the design of the provenance model. Figure 46 illustrates the management process of data changes which consists of four basic steps: Definition of validity constraints, propagation of data changes, validation of the provenance model and compensation techniques.

Definition of validity constraints: If a task of a workflow selects `Structured_Objects` based on selection criteria that is defined on `Attribute_Values`, a validity constraint captures the selection criteria and is linked to

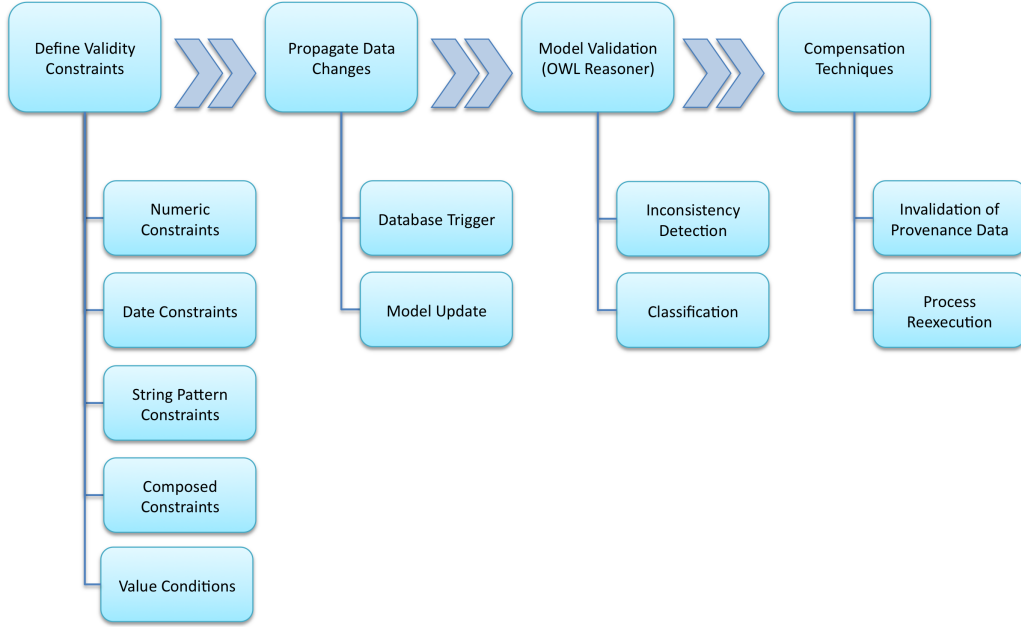


Figure 46: Management of Data Changes in Provenance Data

all selected `Structured_Objects`. The defined validity constraint ensures, that changes in attribute values violating the selection criteria can be easily identified. Validity constraints are modelled as OWL classes that are subclasses of `Filter_Criteria`. All `Attribute_Values` of `Structured_Objects` that match the filter criteria, are declared as instances of the new defined `Filter_Criteria` subclass. Since OWL allows that instances are assigned to several classes, multiple `Filter_Criteria` classes may be defined as types of the same `Attribute_Value`. `Filter_Criteria` classes are defined as equivalence classes of `Attribute_Values`. The equivalence classes are restricted by class expressions that are formulated in the Manchester OWL syntax. These class expressions model the selection criteria with properties `has_attribute_type` and `has_value`. Currently, the provenance framework is capable of defining `Filter_Criteria` for attribute value types integer, String and Date. In the following, a `Filter_Criteria` defined on the attribute age of `PatientData` is presented.

$$\begin{aligned}
& \textit{Filter_Criteria_Age_Interval} \equiv \\
& \quad (\textit{Attribute_Value} \\
& \quad \textit{and} \ (\textit{has_attribute_type} \ \textit{value} \ \textit{example.PatientData.age}) \\
& \quad \textit{and} \ (\textit{has_value} \ \textit{some} \ \textit{integer}[\geq 49]) \\
& \quad \textit{and} \ (\textit{has_value} \ \textit{some} \ \textit{integer}[\leq 51]))
\end{aligned}$$

The class expression defines the new `Filter_Criteria` class `Filter_Criteria_Age_Interval` as an equivalence class of `Attribute_Value`. The equivalence class is restricted to `Attribute_Type` `example.PatientData.age` and the range of age values is limited to the integer interval [49, 51]. A similar `Filter_Criteria` may defined for Date intervals. The next example illustrates the definition of class `Filter_Criteria_Date_Interval` which is an equivalence class for attribute `day_of_birth` of `PatientData`.

$$\begin{aligned}
& \textit{Filter_Criteria_Date_Interval} \equiv \\
& \quad (\textit{Attribute_Value} \\
& \quad \textit{and} \ (\textit{has_attribute_type} \ \textit{value} \ \textit{example.PatientData.day_of_birth}) \\
& \quad \textit{and} \ (\textit{has_value} \ \textit{some} \ \textit{dateTime}[\geq \text{"1964-05-31T00:00:00"} \wedge \textit{dateTime}]) \\
& \quad \textit{and} \ (\textit{has_value} \ \textit{some} \ \textit{dateTime}[\leq \text{"1964-06-02T00:00:00"} \wedge \textit{dateTime}]))
\end{aligned}$$

`Filter_Criteria_Date_Interval` is restricted to attribute values of `day_of_birth` that are greater than 1964-05-31 and smaller than 1964-06-02. For `Filter_Criteria` on String values there are two options. Either an exact String

value is matched or a regular expression defines the matching criteria. `Filter_Criteria_Staging_T` matches all attribute values of type `staging_t` having a String value of "T1". Alternatively, the pattern operand of the Manchester OWL syntax could be used for defining regular expressions.

$$\begin{aligned} \textit{Filter_Criteria_Staging_T} \equiv & \\ & (\textit{Attribute_Value} \\ \textit{and (has_attribute_type value example.PatientData.staging_t)} & \\ & \textit{and (has_value value "T1"{}^{\wedge}\textit{string}})) \end{aligned}$$

If attribute values of different attribute types are combined in a filter criteria, a `Filter_Criteria_Composition` class can be constructed. A `Filter_Criteria_Composition` is an equivalence class of a `Structured_Objects` and is restricted by `Filter_Criteria` on `Attribute_Values` which may be connected by logical AND and OR operators. In the following, `Filter_Criteria_Composition` is defined on the base of `Filter_Criteria_Age_Interval` and `Filter_Criteria_Staging_T`. `Filter_Criteria_Composition` is valid for all instances of type `PatientData` having attribute `age` in the integer interval $[49, 51]$ and an attribute `staging_t` value of "T1".

$$\begin{aligned} \textit{Filter_Criteria_Composition} \equiv & \\ & (\textit{Structured_Object} \\ \textit{and (has_attribute_value some Filter_Criteria_Age_Interval)} & \\ \textit{and (has_attribute_value some Filter_Criteria_Staging_T)}) \end{aligned}$$

In order to detect wrong data associations, `Value_Conditions` are used.

An association is a foreign key relationship from one data record to another. If the foreign key relationship is relevant for the execution of a service, a new `Value_Condition` individual is created and defined as the same individual as the `Attribute_Value` containing the foreign key. Further, the `Value_Condition` is linked to all related `Service_Calls` in order to enable compensation actions. In the following example, the `Attribute_Value` `example.PatientData.p3.sample_id` is a reference to a tissue sample "s21". `value_condition_1` is specified as the same individual as `example.PatientData.p3.sample_id` and a validity constraint is added to its type definition. `value_condition_1` is a valid `Attribute_Value`, if its `has_value` property points to String "s21". If `example.PatientData.p3.sample_id` is changed, the validity constraint is violated and the affected `Service_Call` can be detected.

$$\begin{aligned}
 &value_condition_1 \equiv \\
 &Value_Condition\ or \\
 &(Attribute_Value\ and\ has_value\ only\ \{ "s21" \}) \\
 &value_condition_1\ sameAs\ example.PatientData.p3.sample_id
 \end{aligned}$$

Propagate Data Changes: Scientific workflows use several types of input data as parameter values. Simple literals may be passed, file resources may be provided by network shares and database records may be extracted from database management systems. The semantic provenance model supports data change management for structured resources that result from database systems or XML database repositories. In order to track changes in database records, a simple logging mechanism can be introduced. The basic idea is to let database triggers log all data changes at the row level and periodically generate a change log that updates the provenance model. Figure 47

depicts three database tables that record data changes and coordinate data synchronization. Every database table that represents a `Structured_Type` in the provenance model requires an `ON-INSERT`, an `ON-UPDATE` and an `ON-DELETE` trigger. The database triggers log data changes in tables `Changed_Entry` and `Changed_Attribute_Value`. Each modification of a record is logged in a new entry in `Changed_Entry`. A `Changed_Entry` has a unique id `entry_log_id` and a reference to the primary key of the changed record `entry_id`. Attribute `change_type` documents whether the change results from an insert, update or delete operation. The name of the database table is captured in `table_name` and a timestamp is stored in `change_time`. For every changed attribute value of the record, a new entry is written into `Changed_Attribute_Value` and linked to the corresponding `Changed_Entry` by foreign key `entry_log_id`. Name and type of the affected attribute are stored in `attribute_name` and `attribute_type`. Changed values are recorded in attributes `old_value` and `new_value`. Values of attribute `old_value` may be used to verify the correctness of changes in the provenance model. Synchronization of data changes occurs by selecting a set of `Changed_Entries` and related `Changed_Attribute_Values` that were inserted after a certain synchronization point and create a change log. Every synchronization run is represented by an entry in `Data_Sync_Run`. Attribute `from_time` corresponds to the lower bound of the time interval the `Changed_Entries` were logged, and attribute `to_time` captures the upper bound. The freshest synchronization run is marked by setting attribute `last_run` to true. If a new synchronization run is to be executed, the set of relevant `Changed_Entries` is determined by reading the value of `to_time` of the freshest synchronization run, and filter all `Changed_Entries` with subsequent `change_time` values. The change log has to be processed and all modifications have to be applied to the provenance model. Currently, the Java Provenance Framework lacks a data change processing component, that is capable of reading change logs and of modifying `Structured_Objects` and `Attribute_Values`.

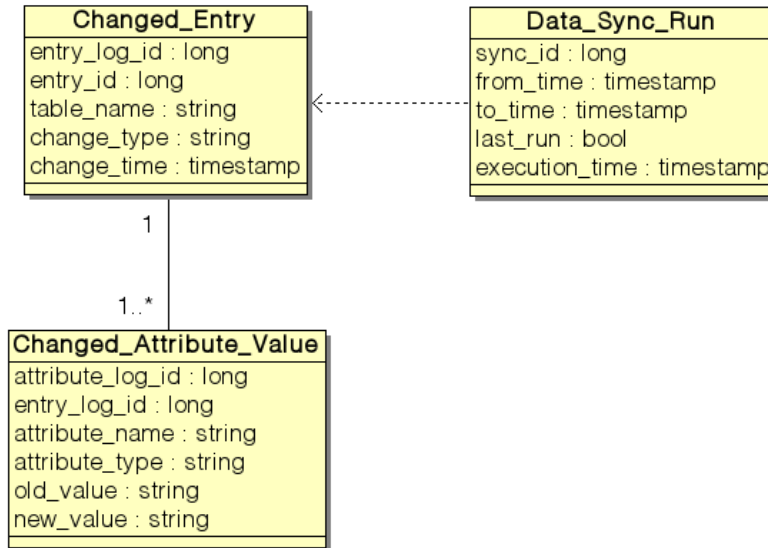


Figure 47: Provenance Model - Capture Data Changes

Model Validation: The provenance model has to be validated, each time data changes are incorporated in the provenance model. Model validation is composed of two phases: detection of inconsistencies and classification of new or changed data. OWL reasoners are capable of validating ontology models and may infer additional relationships by subsumption. We deploy the OWL reasoner HermiT (Version 1.3.6)²⁰, which has been proven as efficient for complex ontologies with large instance data sets [89]. Inconsistencies in the provenance model may occur if changes in attribute values violate defined Filter_Criteria classes. In Figure 48, a detected inconsistency, that resulted from a changed Attribute_Value, is shown. PatientData p3 was selected by a Service_Call of read_patient_data based on two Filter_Criteria: Attribute staging_t must have a String value of "T1" and attribute age must be in the integer interval [49, 51]. After changing staging_t to value "T2", Attribute_Value example.PatientData.p3.staging_t can no longer be of type

²⁰HermiT <http://hermit-reasoner.com>

Filter_Criteria_Staging_T, as the equivalence class definition is violated. Further, PatientData p3 can not be of Type Filter_Criteria_Composition, since it has no Attribute_Value that is of type Filter_Criteria_Staging_T. Both inconsistencies are marked with red arrows. As a consequence, the Process_Instance that used Filter_Criteria_Composition as a selection criteria, is based on invalid data and may have generated invalid output data. The affected Process_Instance can be identified by following the Actual_Parameter that is connected with the Filter_Criteria and then extracting the relevant Service_Call and Process_Instance. Detection of inconsistencies may be accomplished by method consistency_check of class ProvenanceModelHandler. The method returns a list of Filter_Criteria that are invalid due to data changes.

Aside from inconsistencies, data changes can extend data sets and allow to include additional Resources in process executions. For example, a set of patient records is used for annotation of tissue samples. After a gene expression analysis of the sample was conducted, supplementary patient records and biological material become available. If the Filter_Criteria defined for the gene expression analysis matches the new records, the analysis could be repeated with a larger set of gene expression profiles which may deliver more significant results. The identification of Resources that match existing Filter_Criteria is accomplished by using the inference capabilities of an OWL reasoner. As Filter_Criteria are defined as classes, they not only model validity constraints, but can also be used as new types for changed Attribute_Values and Structured_Objects. Figure 49 gives an example of how an OWL reasoner assigns a newly inserted Structured_Object to an existing Filter_Criteria by inferring a new type relationship. PatientData p7 has an age value of 50 and staging_t value of "T1". Thus, the age value fulfils the conditions of Filter_Criteria_Age_Interval and the staging_T value the conditions of Filter_Criteria_Staging_T. Consequently, all constraints of Filter_Criteria_Composition are fulfilled for Structured_Object p7. An OWL

reasoner, classifying the OWL instances of the provenance model, is able to infer the three new type relationships, depicted by green arrows in Figure 49. The newly deferred type relationships indicate, that a reexecution of the service will deliver an extended result set. All data generated by the service execution is still valid. Though, the information that additional data is available for a certain scientific process is valuable and should be presented to scientific users.

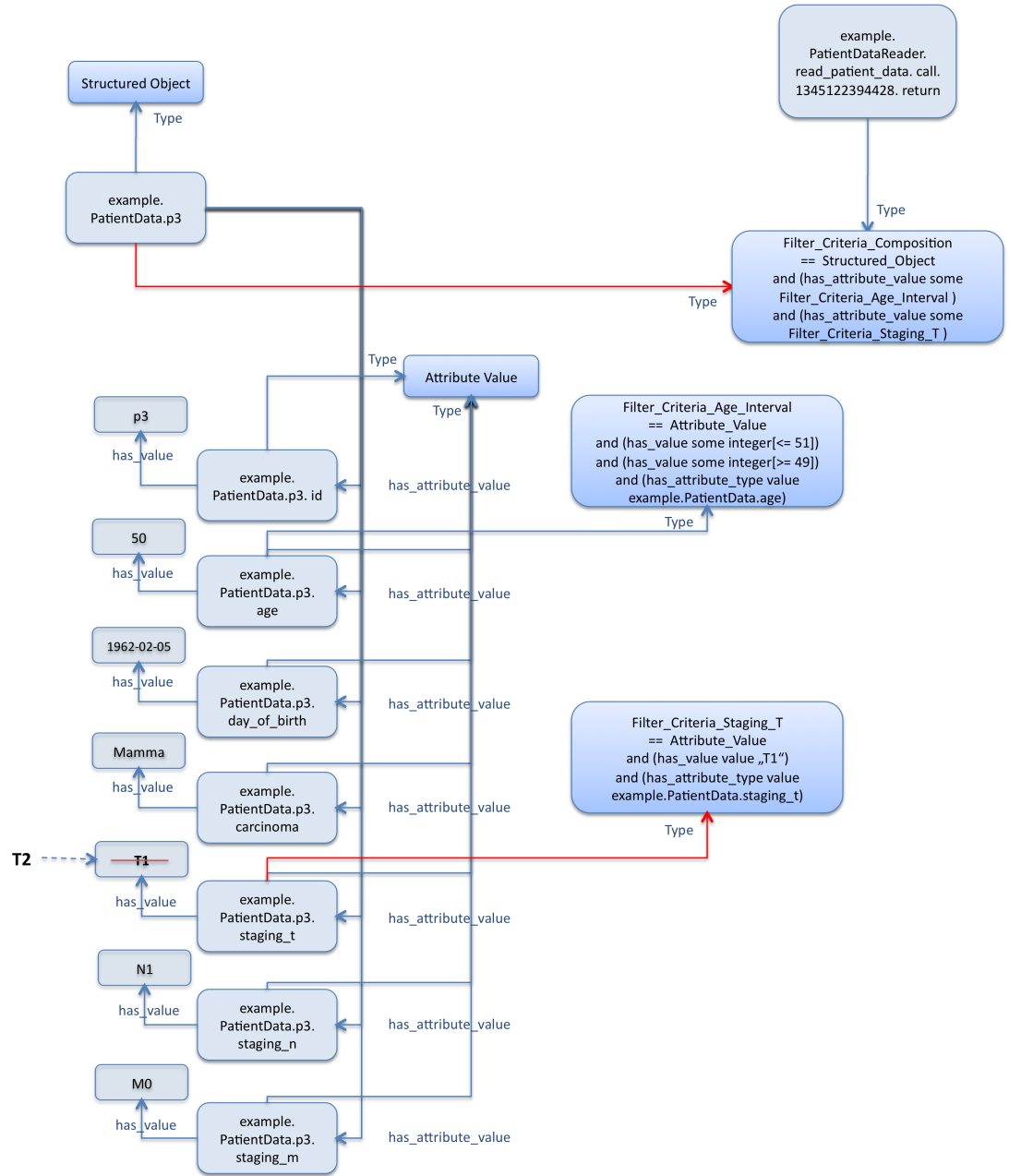


Figure 48: Provenance Model - Inconsistency Detection

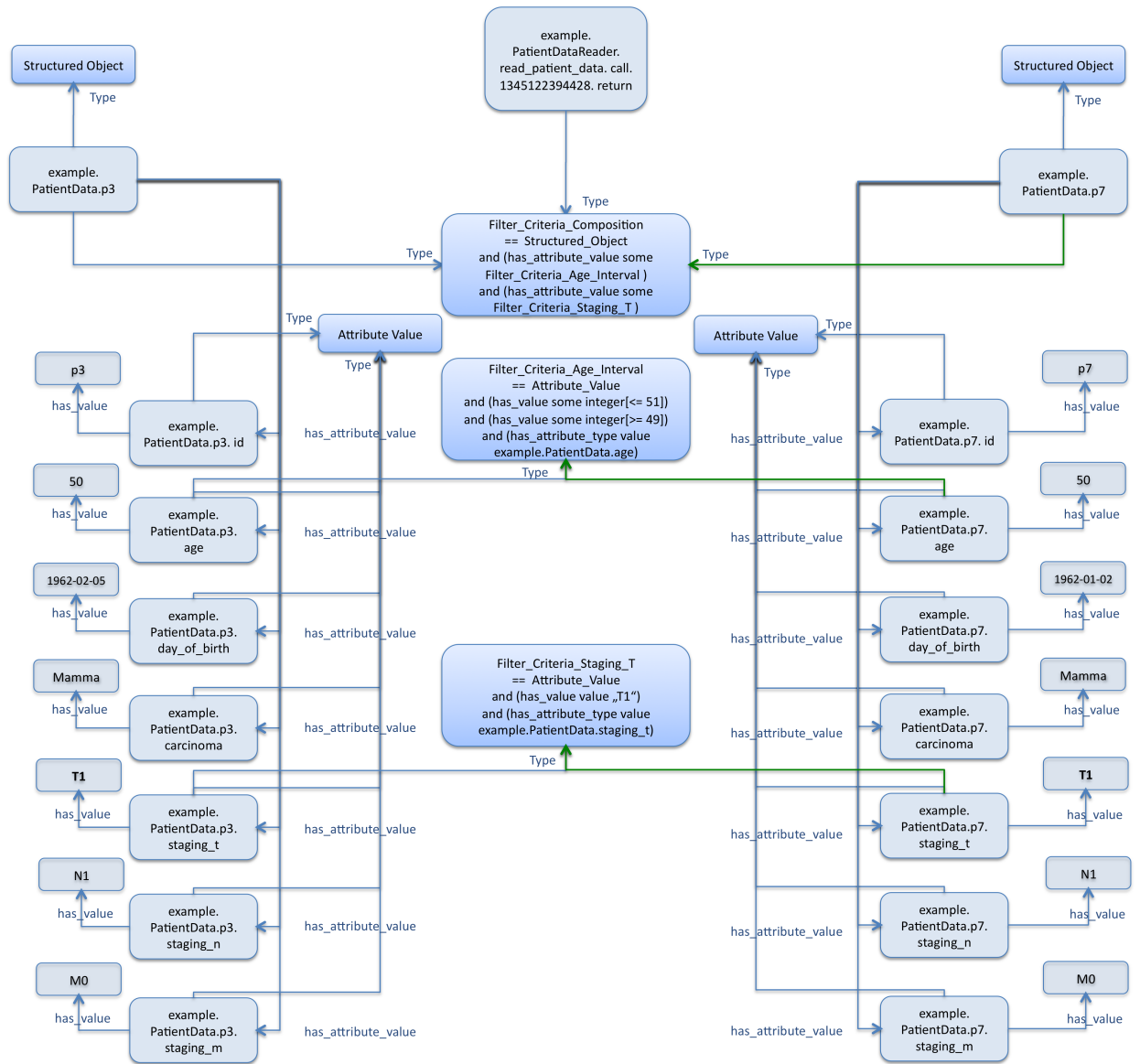


Figure 49: Provenance Model - Classification

Model validation and classification by an OWL reasoner are appropriate techniques for identifying all relevant data changes presented in Section 5.2.2. Data values, that have turned out invalid or outdated in source data repositories, may be corrected in the provenance model. All affected Service_Calls and Process_Instances are identifiable by examining the validity constraints. For incomplete data, additional Attribute_Value entries are inserted in the provenance model. If the new Attribute_Values extend data sets delivered by Service_Calls, classification is capable of deducing the new data set memberships. Supplementary data records may be assigned to existing data sets in the same manner. Changes in foreign key values lead to violations in Value_Conditions and are also detectable by model validation.

Compensation Techniques: The results of model validation are handled by two different compensation techniques: invalidation and process re-execution. Resources that violate validity constraints are flagged as invalid as well as affected Service_Calls and Process_Instances. Hence, process executions that are based on invalid input data become identifiable. Consequently, if output data of invalid Process_Instances has been consumed by other Process_Instances, they have to be invalidated in likewise manner. The process of identifying further Process_Instances that are affected by invalidation of Resources is called invalidation propagation. In the following, the invalidation actions are described by SPARQL Update statements. In the first step, the immediately affected Service_Calls, Process_Instances and Resources are invalidated, as shown in Listing 22.

In our example, Attribute_Value example.PatientData.p3.staging_t is violating Filter_Criteria_Staging_T and example.PatientData.p3 is violating Filter_Criteria_Composition. We start the SPARQL Update with a deletion of these two inconsistent type relationships. Further, Filter_Criteria_Composition is decoupled from its related Actual_Parameter. The actual_parameter_id data property has to be deleted because it is used in the definition of the equivalence class of Filter_Criteria_Composition. Since the Service_Call (?ser-

vice_call_10), that is related to Filter_Criteria_Composition, generated invalid data, it is not necessary to include it in future validations. The data property invalid of ?service_call_10 is set to true. Object_Collection ?object_collection contains invalid data and is therefore invalidated as well. All successor Service_Calls of ?service_call_10 are identified and invalidated together with all generated output data of the Service_Calls. The predecessor Service_Calls do not have to be invalidated, since their results are still valid. If a Process_Instance is reexecuted, only the invalidated Service_Calls need to be conducted. Finally, the related Process_Instance ?process_instance_1 is invalidated.

Listing 22: Invalidation of Inconsistent Process Components

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>

DELETE
{
  m:example.PatientData.p3 rdf:type m:Filter_Criteria_Composition .
  m:example.PatientData.p3.staging_t rdf:type m:Filter_Criteria_Staging_T .
  ?actual_parameter rdf:type m:Filter_Criteria_Composition .
  ?actual_parameter m:actual_parameter_id ?parameter_id .
  ?object_collection m:invalid false .
  ?service_call_10 m:invalid false .
  ?service_call_11 m:invalid false .
  ?o_resource_1 m:invalid false .
  ?process_instance_1 m:invalid false .
}
INSERT
{
  ?object_collection m:invalid true .
  ?service_call_10 m:invalid true .
  ?service_call_11 m:invalid true .
}
```

```

    ?o_resource_1 m:invalid true .
    ?process_instance_1 m:invalid true .
}
WHERE
{
    ?actual_parameter rdf:type m:Actual_Parameter ;
        rdf:type m:Filter_Criteria_Composition ;
        m:actual_parameter_id ?parameter_id .
    ?service_call_10 m:has_output ?actual_parameter ;
        m:has_output/m:has_resource ?object_collection ;
        m:has_successor+ ?service_call_11 .
    OPTIONAL { ?service_call_11 m:has_output/m:has_resource ?o_resource_1 } .
    ?process_instance_1 m:starts_with/m:has_successor* ?service_call_10 .
}

```

In the next step, ProcessInstances that may be affected by the invalidations are searched. In Listing 23, a SPARQL query searches for valid ProcessInstances, which consumed invalidated Resources. For each valid ProcessInstance ?process_instance, all related ServiceCalls ?service_call2 are examined. If a ServiceCall ?service_call2 used an invalidated Resource ?i_resource as input parameter, the triple ?process_instance, ?service_call2 and ?i_resource is added to the result set. If the result set is empty, the invalidation can be finished. Otherwise, the identified ProcessInstances, ServiceCalls and Resources must be invalidated.

Listing 23: Invalidation Propagation - Query

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>

SELECT ?process_instance ?service_call_2 ?i_resource
WHERE
{

```

```

?process_instance m:invalid false ;
    m:starts_with ?service_call_1 .
?service_call_1 m:has_successor* ?service_call_2 .
?service_call_2 m:has_input/m:has_resource ?i_resource .
?i_resource m:invalid true .
}

```

Listing 24 illustrates how invalidations are propagated in additional ProcessInstances. The affected ProcessInstances are identified similarly to the previous SPARQL query. All ServiceCalls that are based on invalidated Resources are invalidate as well. Further, all generated output data, all successor ServiceCalls and their generated output data are invalidated. Finally, the invalid flag is set for the corresponding ProcessInstance. Propagation of invalidations is an iterative process - whenever a ProcessInstance is invalidated, a subsequent search for other potentially affected ProcessInstances is required. Therefore, the steps invalidation propagation query and invalidation propagation update are repeatedly executed, as long as the invalidation propagation query delivers results.

Listing 24: Invalidation Propagation - Update

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparql: <http://pellet.owldl.com/ns/sdle#>
DELETE
{
    ?service_call_20 m:invalid false .
    ?service_call_21 m:invalid false .
    ?o_resource_1 m:invalid false .
    ?o_resource_2 m:invalid false .
    ?process_instance_1 m:invalid false .
}
INSERT
{

```

```

?service_call_20 m:invalid true .
?service_call_21 m:invalid true .
?o_resource_1 m:invalid true .
?o_resource_2 m:invalid true .
?process_instance_1 m:invalid true .
}
WHERE
{
  ?process_instance_1 m:invalid false ;
    m:starts_with ?service_call_1 .
  ?service_call_1 m:has_successor* ?service_call_2 .
  ?service_call_20 m:has_input/m:has_resource ?i_resource .
  ?i_resource m:invalid true .
  OPTIONAL
  {
    ?service_call_21 m:has_output/m:has_resource ?o_resource_1 .
    ?service_call_20 m:has_successor+ ?service_call_21 .
    ?service_call_21 m:has_output/m:has_resource ?o_resource_2 .
  }
}

```

The results of classification of the provenance model are used to enhance the information that is present for executed processes. If new resources match defined Filter_Criteria of Service_Calls, the Service_Calls and the related Process_Instances are marked with the flag `additional_data_available`. The information does not imply, that processes have to be reexecuted. The scientific user may choose to reexecute processes or skip the information. Based on the classifying example, in which the additional PatientData p7 matches the Filter_Criteria_Composition, Listing 25 illustrates how the corresponding Service_Call and Process_Instance is marked with the `additional_data_available` flag. Starting from Filter_Criteria_Composition, the related Service_Call `?service_call_10` and the related Process_Instance `?process_instance_1` may be identified and enriched with the new flag.

Listing 25: Mark Service Call and Process Instance as Reexecutable

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m: <http://www.sem_model.org#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>

DELETE
{
    ?service_call_10 m:additional_data_available false .
    ?process_instance_1 m:additional_data_available false .
}
INSERT
{
    ?service_call_10 m:additional_data_available true .
    ?process_instance_1 m:additional_data_available true .
}
WHERE
{
    ?actual_parameter rdf:type m:Actual_Parameter ;
        rdf:type m:Filter_Criteria_Composition .
    ?service_call_10 m:has_output ?actual_parameter .
    ?process_instance_1 m:starts_with/m:has_successor* ?service_call_10 .
}
```

5.7 Established Scientific Workflow Management Systems

A variety of scientific workflow managements have been developed in recent years. In the following, an overview of the most established scientific workflow management systems is given. The system architectures are presented as well as the tools and interfaces for designing and executing scientific workflows. Additionally, techniques for capturing provenance data and query interfaces

for extracting provenance data are analyzed. With regard to the requirements defined in Section 5.2, the features and capabilities of the systems are investigated in order to find out to which extend the requirements can be covered. Finally, a summarized comparison of all analyzed systems is given.

5.7.1 BioWBI

The Bioinformatics workflow builder interface (BioWBI) is a workflow management system specialized for application in life sciences and has been developed by the IBM Innovation Center. The system provides a platform for executing bioinformatics analysis workflows and share the results among researchers. The architecture of BioWBI is illustrated in Figure 50.

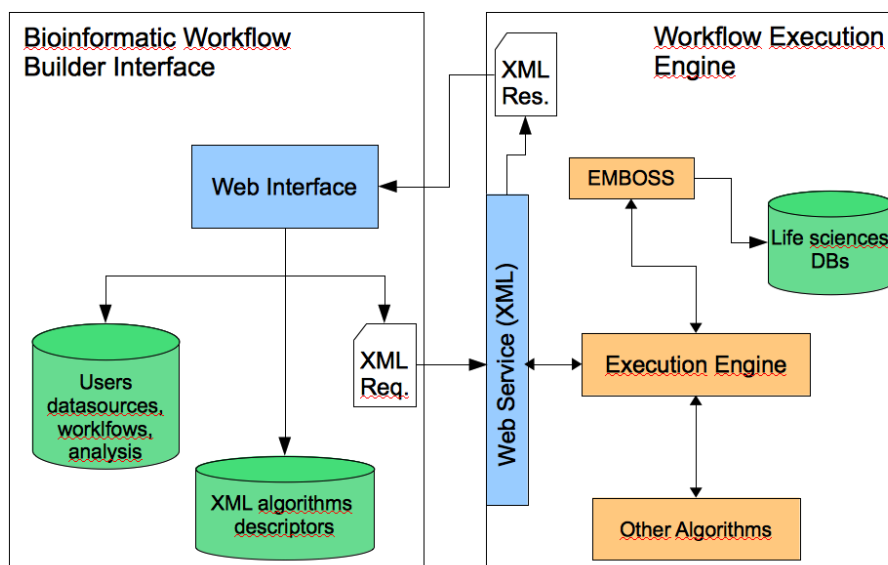


Figure 50: BioWBI Architecture [51]

BioWBI is composed of two web applications which provide workflow definition and workflow execution functionality. The application “Bioinformatics Workflow builder interface”, from which the system derives its name, is a Java web application that is used for the management of input data sources and for creating workflows. Researchers access the BioWBI component by a web interface, which allows to design workflows by selecting algorithms from a repository. Further, data sources may be added and results of completed workflow runs may be displayed. Workflows are executed by translating the workflow definition in corresponding web service calls which are sent to the workflow execution engine (WEE), which calls the requested analysis method and returns the result back to the BioWBI. In order to provide analysis methods for molecular biology, the European Molecular Biology Open Software Suite (EMBOSS) ²¹ has been integrated in the workflow execution engine. EMBOSS is an open source analysis package offering various analysis methods such as sequence alignment, protein analysis or nucleotide sequence pattern analysis. The system focuses on strict decoupling of data management, workflow definition and workflow execution [51]. As tasks are executed by creating web service requests, additional external web services may be integrated with less effort. The system supports typical job-oriented workflows and documents results of execution runs. Though, logging and querying data provenance is not offered by BioWBI.

5.7.2 Discovery Net

Discovery Net is a scientific workflow management system that is capable of executing workflows in a grid-based environment. The system may execute workflows that consist of distributed grid services and dynamically transfer data resources between these processing services. Both data access and computational algorithms are encapsulated into services with predefined input and output parameters. By specifying the order of service execution and the

²¹EMBOSS <http://emboss.sourceforge.net>

involved data passing rules, workflows may be executed as pipelines. Discovery Net was used for assigning computer-intensive analyses of the molecular biology domain to a high-performance computing infrastructure. For instance, nucleotide- und protein-level annotation of the genome were performed.

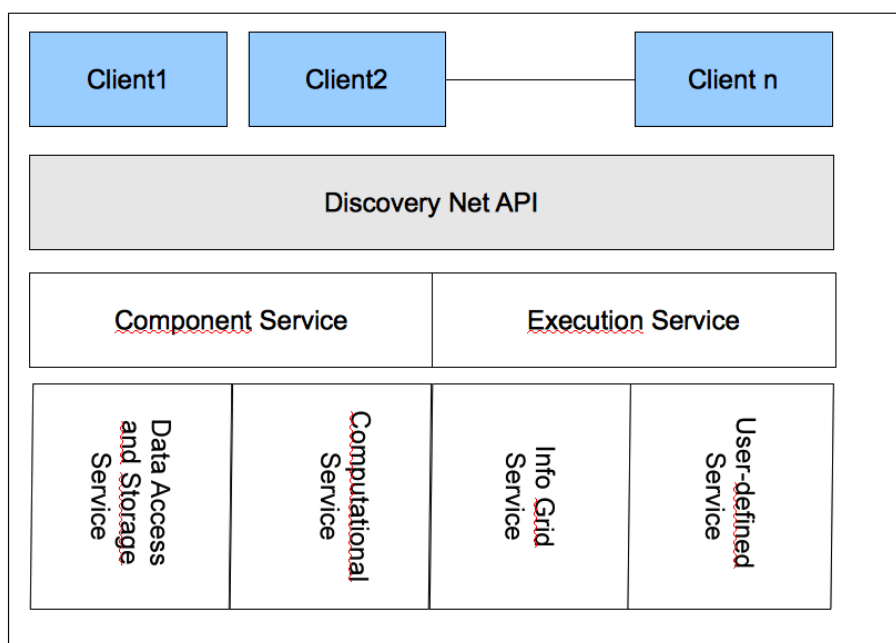


Figure 51: Discovery Net Architecture [85]

In Figure 51, the core components of Discovery Net are presented. Multiple clients may connect concurrently to a Discovery Net node, define workflows and access data resources and analysis results. Discovery Net nodes may be arranged in a cluster and provide access to a multitude of services. Clients use the Discovery Net API to communicate with all types of services. The task of the component service is to integrate various services into

Discovery Net. It supports different kinds of protocols such as HTTP and SOAP. Workflows are defined in the proprietary language DPML (Discovery Process Markup Language), which allows to specify both the data and the control flow model in an XML-based notation. The DPML specifications are analyzed by the execution service which maps the defined operations to available services. The execution service is responsible for coordinating the execution of distributed services. It initiates data transfers of all necessary input data and collects the output results. The role of the data access service is to provide a general interface for retrieving data that is available at the local node. Intermediate results are cached in Discovery Net nodes to speed up workflow executions. Further, the data service is responsible for maintaining workflow definitions. The computational service is used to integrate services that are available on the same Discovery Net node. The info grid service is an interface to external databases. Finally, customized services may be integrated by the user defined service [85].

5.7.3 Kepler

The Kepler system [54] is a well-known scientific workflow management systems in life sciences. It is based on the Ptolemy II-Framework²², which is an open-source software for coordinating the execution of software components. Kepler reuses the actor/director design principle of Ptolemy II to define and execute workflows. Actors are execution tasks that build up the workflow to be run. Actors possess input and output ports which are used to realize passing of input and output data. Actors may be arranged in a graphical user interface and data flow edges connect the corresponding input and output ports. Data flow edges are denoted as channels or relations. Directors allow to configure the type of workflow execution. For instance, if a workflow should be run in a predefined sequence, a SDF (synchronous data flow) director executes all actors sequentially in a single thread. An input port is

²²Ptolemy <http://ptolemy.eecs.berkeley.edu/ptolemyII/>

capable of receiving input data in a FIFO queue up to a predefined number. Though, parallel execution of a workflow may be realized by assigning a PN (process network) director, which initializes a separate thread for each actor. In workflows with PN directors, input and output ports are unlimited FIFO queues due to the dynamic and parallel execution of actors [1]. This kind of execution is data-driven, as each actor starts its execution as soon as all required input ports are filled. Workflows may contain branches of data flow allowing to distribute data to multiple actors. Additionally, workflows may contain nested workflows in form of composite actors.

Kepler provides a large standard library of actors including basic file operations (accessing and writing local and remote files), data manipulation and conversion methods, statistical operations, unix commands and web service execution. Workflows may be defined, saved as templates and executed in the same user interface. The results (e.g. text files, images) generated by actors are displayed in separate windows. In Figure 52, an example workflow in the Kepler system is displayed. In the illustrated workflow, all Gene Ontology information for a certain oligonucleotide 'H200006598' is extracted from distributed data repositories. Gene Ontologies for biological processes, molecular functions and cellular components are stored in separated PostgreSQL databases. Each database may be accessed by a database connector actor which is parametrized with the corresponding database url, database name, user and password values.

The output port of the database connector is connected with the input port of a database query actor. Further, a query for every data repository is written and connected with the input port of the database query actor. The results of all database query actors are sent to a single display actor which collects all results and displays them in a separate window. A SDF director is assigned to the workflow, specifying the sequential execution of all actors. When executing the workflow, the result sets of all three data repositories are fetched, joined and displayed as text in a pop-up window.

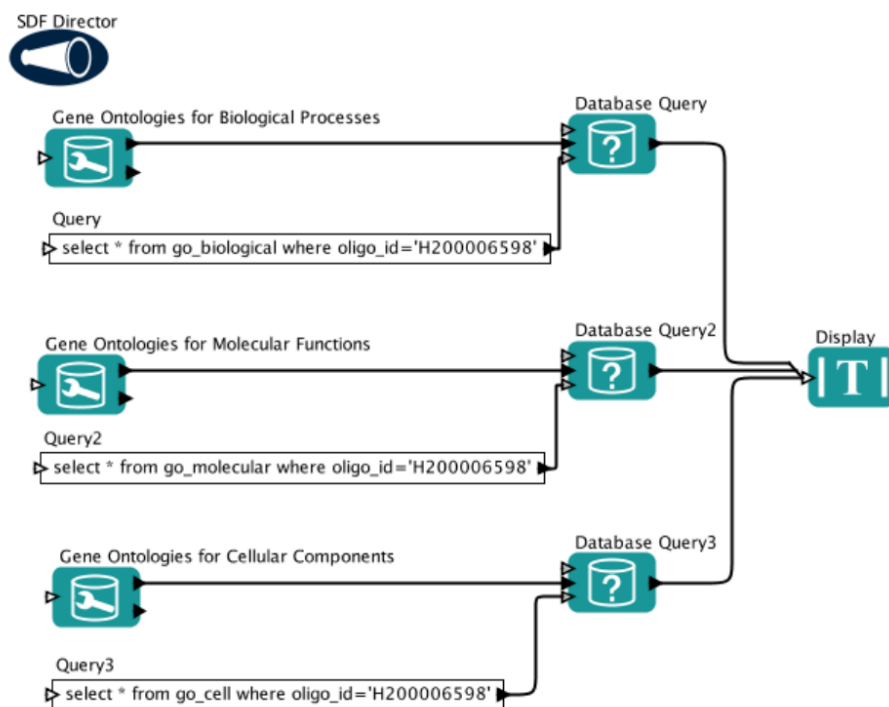


Figure 52: Kepler Example Workflow

Kepler has a multi-layered system architecture (Figure 53) encapsulating the Ptolemy II framework as a core component. Ptolemy II allows to model compositions of actors which are executable in various simulation scenarios that are specified by directors. The user interface integrates Vergil which is a visual editor for building models pictorially [14]. Actors are stored as Kepler archives (KAR files) and may be imported in other Kepler Systems. The Kepler Object Manager is used for accessing various kinds of data (kar archives, metadata, annotations) which may be located on the local host or on remote resources.

Further, Kepler supports semantic annotation of actors, types and work-

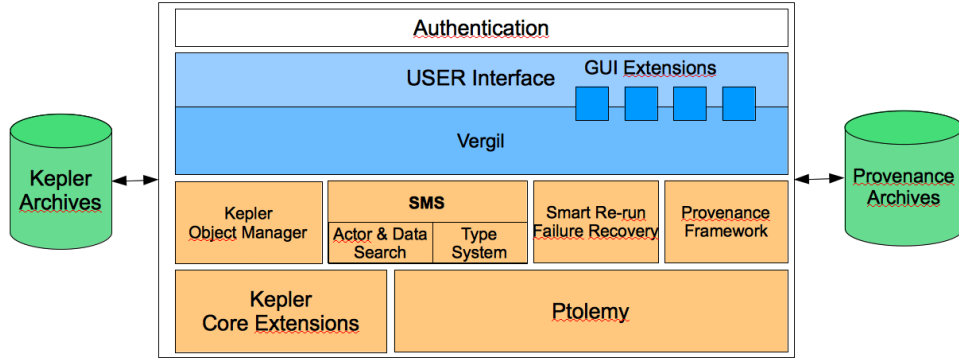


Figure 53: Kepler Architecture

flows enhancing the categorization and search capabilities of the system. The Semantic Mediation System (SMS) facilitates data integration from distributed resources with the help of ontologies. Recording of provenance has been recently implemented in a separate module which may be installed optionally. Provenance recording is deactivated by default and has to be turned on by the user. With the help of tracking data provenance, the workflow execution is to be enriched with additional features. If the execution of an actor fails because of unavailability of a service or a wrong configured library, the entire workflow execution has to be repeated. Though, if intermediate results are stored, all successful completed tasks do not have to be reexecuted. In this case, it is possible to perform a *failure recovery* of the workflow by reexecuting only execution branches of failed actors. Additionally, the execution of a similar workflow may be optimized on the basis of provenance data. If a workflow is re-run with slightly modified input parameters, only actors that depend on the modified data have to be reexecuted. All actors that are not effected by the changed parameters are not repeated, since the generated output data is still available [1]. Kepler's provenance module interacts with a relational provenance model which is stored in a separate database sys-

tem. The provenance model captures data provenance information during workflow execution and may be accessed either by the supplied provenance query API or immediately by SQL queries. Generally, the provenance model is a mixture of workflow definition and workflow execution data allowing to retrace data flows between actors and to retrieve the corresponding static information of workflows like the set of involved actors. In Figure 54, a coarse UML class diagram of the provenance model is illustrated [76].

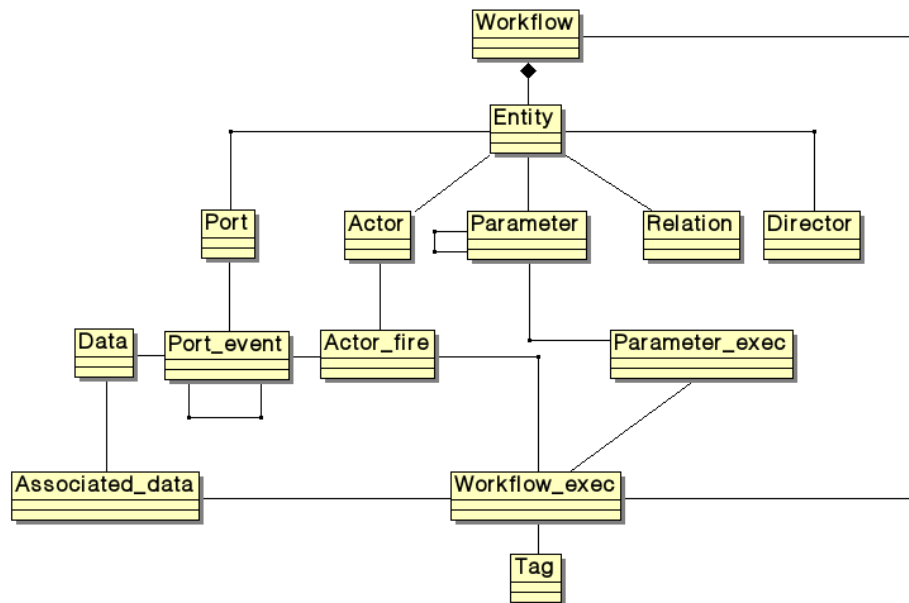


Figure 54: Kepler Provenance Model

A workflow is composed of a set of entities which may be directors, actors, ports, relations and parameters. These classes represent the specification of the workflow. Executions of workflows are recorded in *Workflow_exec* entries. Though, parameter values are part of the workflow definition as well as of the workflow execution. Since Kepler supports execution of loops, actors may be executed multiple times with different parameters. The initial set

of parameter values at the beginning of workflow execution is captured in *Parameter_exec* entries. Further, if parameter values change during execution, they are stored in separate parameter entities. A recursive association maps parameter values to previous values. During workflow execution, all executed actors are recorded in the *Actor_fire* table. Each actor receives a set of input data tokens and creates a set of output data tokens. All data tokens are stored in the *Port_event* table, which links data values of table *Data* with *Actor_fire* and *Port* entries. Kepler's provenance model allows to reconstruct the entire data flow of a workflow execution. For instance, the set of data tokens that were involved in generating a certain data token X could be determined by iteratively querying tables *Actor_fire*, *Port_event* and *Data*. Further, by logging both input data and input parameters of actor executions, the provenance model supports smart reruns of workflows. If a workflow is reexecuted with modified parameters, the set of actors may be identified that is affected by the modifications and must be reexecuted. Since the provenance model is implemented in a relational database, data provenance may be extracted by SQL queries. Although Kepler's provenance model is well-suited for most of the data provenance requirements in a scientific workflow system, it has some limitations. On the one hand, data provenance is recorded at a coarse-grained level. Data files are modelled as tokens, without considering the internal structure of the file and obvious dependencies of data items within the file. On the other hand, Kepler does not offer semantic annotation of data tokens. While workflows have a mandatory field for assigning a semantic category, data tokens can not be annotated.

5.7.4 Kepler/pPod

Kepler/pPod ²³ is an extended version of the Kepler system which was designed for executing phylogenetic data analysis - pPod is an abbreviation for processing PhyloData. pPod focuses on the design and execution of collection-oriented scientific workflows and leverages an elaborated data provenance tracking system for capturing data lineage for single data items. pPod introduced a new director type for collection-oriented modelling and design (COMAD) which may be assigned to workflows, similar to Kepler's PN and SDF directors [13]. The provenance model of pPod captures data flows of single data tokens and collections of data tokens during workflow execution. A tree-based structure is used for representing all data tokens of a workflow run. The tree consists of internal nodes, which are collections and leaf nodes, that are empty collections or single data tokens. Initially, the tree contains solely the input tokens of the workflow. During workflow execution, for all generated data tokens additional internal and leaf nodes are appended to the tree. At the end of execution, a single tree contains all data tokens that were involved in the workflow. In order to track the changes of each execution step (actor), provenance annotations are used. For instance, if actor a_1 reads data token d_1 and produces data token d_2 , the annotation $\{ins(a_1, d_2), dep(d_2, d_1)\}$ is created. The annotation defines that data token d_2 was inserted by actor a_1 and d_2 depends on d_1 . Thereby, it is possible to deduce data dependency graphs for all tokens.

Figure 55 gives an example of capturing provenance data in a simple workflow. The illustrated workflow consists of three actors a_1 , a_2 and a_3 which receive a set of input data tokens and generate new output data tokens. The tree of provenance data is denoted as X_i , whereas i depicts the current version of the tree. Actor a_1 is assigned input tokens $d_1, \dots, d_6$. d_1 and d_2 are collections of tokens while d_3 , d_4 , d_5 and d_6 are single tokens. Actor a_1 produces the new collection node d_7 which consists of single tokens d_8 ,

²³Kepler/pPod <http://daks.ucdavis.edu/kepler-ppod/>

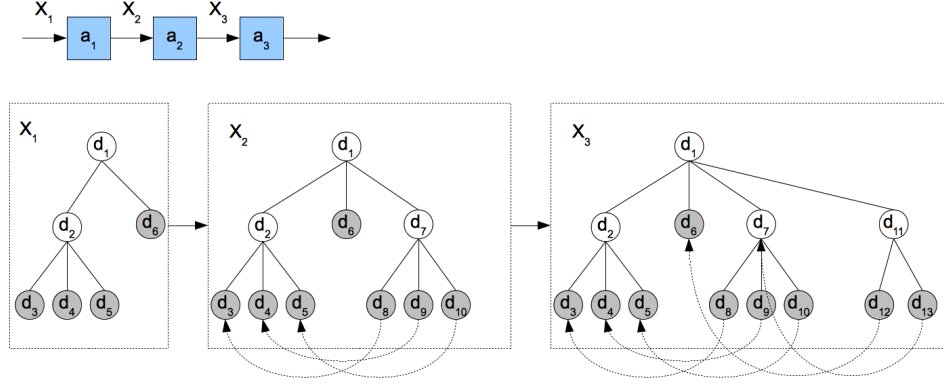


Figure 55: pPod Kepler - Data Provenance Tree

d_9 and d_{10} . Each of the new single tokens was generated on the basis of one of the input tokens d_3 , d_4 and d_5 . Thus, three dependency relations are created: $d_8 \mapsto d_3$, $d_9 \mapsto d_4$ and $d_{10} \mapsto d_5$. The resulting data provenance tree is given in version X_2 . All additional data tokens are inserted as new collection and leaf nodes and dotted arrow lines indicate the dependency relations between data tokens. In the next step, actor a_2 is executed, inserting the new collection token d_{11} which consists of single tokens d_{12} and d_{13} . Token d_{12} is based on d_6 and token d_{13} depends on d_7 . The resulting version of the provenance tree is shown in X_3 . Actor a_3 does not create any new tokens, therefore no new tree version is created.

In order to capture the changes of actors, the following provenance annotations are created:

$$\Delta a_1 = \{ins(d_7, a_1), dep(d_8, d_3), dep(d_9, d_4), dep(d_{10}, d_5)\}$$

$$\Delta a_2 = \{ins(d_{11}, a_2), dep(d_{12}, d_6), dep(d_{13}, d_7)\}$$

The tree of provenance data may be used to recover a certain state of workflow execution. Since actors are executed sequentially, there is an implicit ordering of annotations. Starting at the final state of execution, the

sequence of annotation may be used as an undo log for transforming the tree of data tokens into a certain execution state. For instance, in the above example, the execution state of the workflow before actor a_2 has been executed can be recovered by undoing changes of Δa_2 . The internal node d_{11} has to be removed and the involved dependency relations have to be deleted [2].

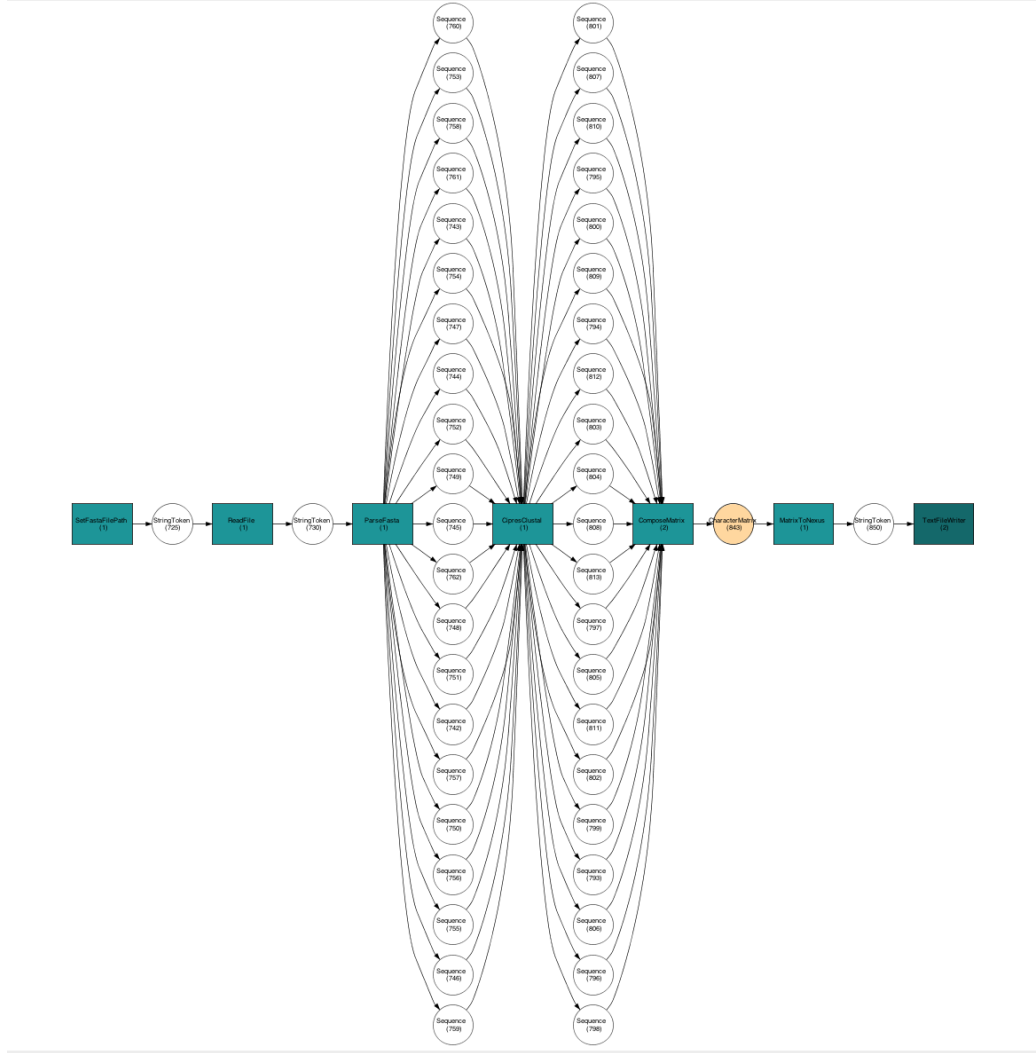


Figure 56: pPOD Kepler - Provenance Browser

Kepler/pPOD has a built-in provenance browser for visually exploring data lineage of workflow executions. The provenance browser offers three different types of view: the dependency history view, the collection history and the invocation graph. The dependency history perspective presents actors, data tokens and data dependencies in an integrated view. If a single data token is selected, all data tokens it depends on are highlighted. Execution states of a workflow may be chosen by navigating in the sequence of actor executions. Figure 56 illustrates an example dependency history graph.

The collection-history graph focuses on the hierarchical structuring of data tokens during execution. It visualizes the data provenance tree and allows to navigate between different versions of the tree. Finally, the invocation graph is a graphical representation of actors of the workflow which shows related input and output tokens in a detail panel. The pPOD provenance model is capable of assessing data lineage for collection-oriented workflows. It has been seamlessly integrated in the Kepler system and allows powerful data exploration by the provenance browser. There is a lack of query capabilities for extracting provenance information based on user-defined filter criteria. Further, exploration of data provenance is limited to single workflow executions. Thus, provenance queries involving multiple workflow executions may not be answered. pPOD was released as a preview software in 2008 and has not been updated in recent years.

5.7.5 Pegasus

Pegasus [24] was designed as a scientific workflow management system for executing complex workflows on distributed resources. Various scientific domains (e.g. astronomy, bioinformatics, chemistry) require the support of computer-intensive tasks in distributed environments and the efficient sharing of data and applications. In this context, grid computing has been proved as a powerful technology for establishing a network of loosely coupled computers which may conduct large-scale calculations and share data for scien-

tific applications. Pegasus derives its name from *Planning for Execution in Grids*. That is, Pegasus allows to specify a scientific application in form of a workflow, which may be executed on a grid infrastructure. The specification of a workflow is denoted as an abstract workflow, which is independent of the underlying grid infrastructure. Tasks are considered as transformations which are assigned input parameters and input files and generate output files. An abstract workflow is specified as a directed acyclic graph, in which processing steps (transformations) are denoted as nodes and data flows as edges. The graphs are stored in a DAX (DAG XML) notation.

New abstract workflows may be created in different ways. On the one hand, workflow specifications may be created programmatically by Python or Java programs. On the other hand, workflows may be specified by the web-based Pegasus computational portal or designed by the standalone Pegasus GUI applications. Further, other SWfMS like Wings or Triana, that are capable of exporting DAX specifications may be used. The core components of the Pegasus workflow management system are the Pegasus Mapper, Engine and Scheduler. The Pegasus Mapper translates an abstract workflow specification into an executable workflow. Therefore, the mapping component consults information services of the grid in order to locate the required resources. The Globus Replica Location Services delivers the physical file locations of requested resources. The set of available transformations is managed by the Transformation Catalog, which maps an transformation identifier to its physical location. Further, information about the current state of grid infrastructure components like load or free disk space are supplied by the Globus Monitoring and Discovery Service. Depending on current workload and free resources, Pegasus mapper may define an optimized executable workflow. In order to transport data files to destination systems in the grid, gridftp server are used. Before the workflow execution is initiated, Pegasus checks, whether all necessary input files for the workflow are available and are accessible by data transport mechanisms [23]. If all preconditions for the

workflow execution are met, Pegasus Mapper translates the abstract workflow into an executable workflow which is denoted as a *concrete* workflow. For each transformation of the workflow and for each required data movement a submit file is created that contains all necessary parameters for the execution of a job on the grid infrastructure. The set of generated submit files is handed over to the execution engine DAGMan which is a meta-scheduler for the high-throughput computing environment Condor-G [33]. DAGMan is in charge of submitting jobs to the Condor-G framework in a predefined order. If a job execution fails, DAGMan may reschedule the job several times or alternatively generate a rescue DAG that may be reexecuted at a later time [22]. The scheduling component (Condor Schedd) is responsible for monitoring the execution of jobs on the grid framework. The system architecture of Pegasus is illustrated in Figure 57.

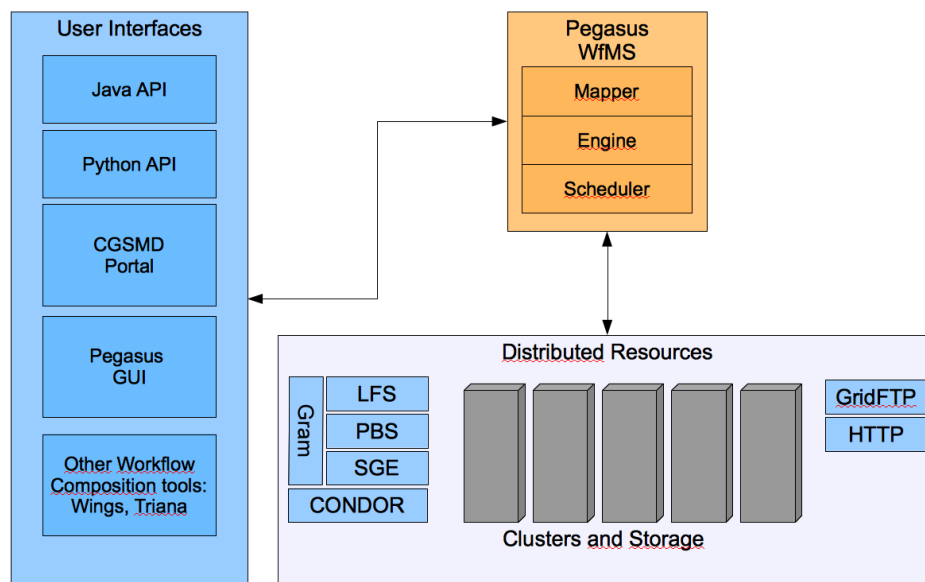


Figure 57: Pegasus System Architecture

Pegasus captures data provenance information during workflow execution. If a data file is generated by a transformation, the involved input files and parameters are recorded. Depending on free storage capabilities, data files may remain physically on the grid infrastructure or are deleted. If a transformation is to be executed and the corresponding output data has already been generated and is still available, the execution may be skipped and the output data may be transported to dependent subsequent transformations. When translating abstract workflows into concrete workflows, Pegasus Mapper controls, whether required data has already been generated by previous workflow runs. If such data files are still presents, transformations are replaced by simple data accesses in concrete workflows. The Pegasus provenance tracking mechanism allows efficient optimizations of workflow executions. Provenance data is stored in the Pegasus Provenance Tracking Catalog, which is a MySQL database. However, the provenance model represents data at file level and does not consider issues of collection-oriented workflows. There is no type information for input/output data and semantic annotations for data items are not supported.

5.7.6 Wings for Pegasus

Wings is an extension framework for Pegasus which supports sophisticated management and validation of workflow templates. Workflow definitions are represented as semantic objects in OWL-DL which allows to specify constraints on transformation steps and input data. These constraints may be validated by a semantic reasoner. For instance, a file processing service may require that all input files are UTF-8 encoded. Users are presented a repository of available workflow templates and input data. A new workflow definition may be created by selecting a template and assigning input data from the repository to transformation steps. If no constraints are violated, Wings translates the workflow definition into a DAX notation which is sent to Pegasus for resource allocation and workflow execution [35]. In Figure

58, the details of the Wings architecture are depicted. Technicians may add new transformations as application components and define requirements for both input data and execution. New workflow templates are designed by experienced scientists and are stored as workflow libraries.

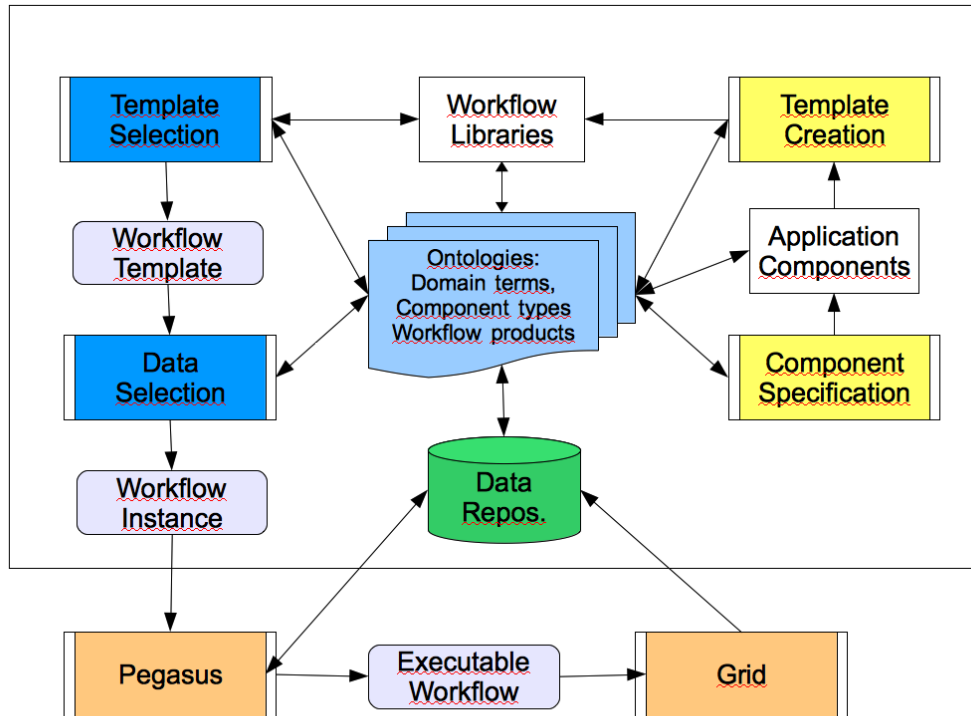


Figure 58: Wings Architecture [35]

After a template is selected in the data selection step, all input parameters must be entered and all input data has to be assigned. A data repository provides access to all data resources available on the grid. Each data resource is characterized by a set of OWL properties (e.g. file type or file size). A data resource can be selected as an input data for a transformation if it has the required properties and does not violate any predefined constraints. When all inputs of a workflow template have been bound to specific data

sets, the template becomes a workflow instance which is translated into a DAX notation for Pegasus.

The semantic model of Wings is used for representing workflow definition data, domain-specific constraints and workflow execution data. An excerpt of the model is illustrated in Figure 59. A workflow template consists of a set of nodes which are workflow tasks. Each node has either an associated application component or a collection of components that should be executed. Data flows between tasks are modelled by links which assign data to component nodes. Depending on the type of a link, data flows are categorized into input data, input/output data and output data. Data items are represented at the file level. That is, either a single file or a set of files may be assigned by a single link to a task node. File sets are specified by DataCollections, which are either collections of files (FileCollections) or collections of data collections (ColOfDataCollec). Thus, it is possible to define nested file collections up to an arbitrary depth ²⁴.

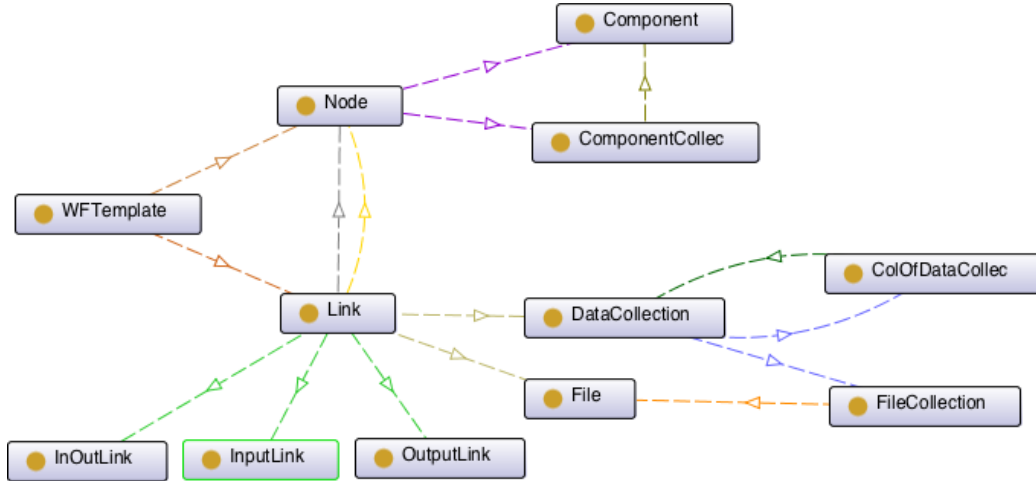


Figure 59: Wings Provenance Model

The classes depicted in Figure 59 constitute the domain independent part

²⁴The Wings semantic model for workflows is available under <http://www.isi.edu/ikcap/Wingse/workflowOntology.owl>.

of the model. In order to represent domain-specific components, corresponding subclasses of the OWL class `Component` are created. For instance, an analysis of variance components may be modelled as a separate subclass of `Component`. Instances of domain-specific components are actual programmes with typed parameters and input/output data. Similarly, domain-specific input data is represented by creating subclasses of general `File` or `DataCollection` classes. The semantic model is capable of answering the set of provenance queries specified in the First Provenance Challenge. Provenance queries are formulated as SPARQL queries, whereas various filter criteria on workflows, data items and components may be specified. However, queries concerning workflow execution are answered by accessing the Pegasus Provenance Tracking Catalog which is located in a relational database. Thus, provenance queries involving both semantic data and workflow execution data could only be answered by creating separate SPARQL and SQL queries and merging the result sets. For instance, if a set of gene expression files, assessed in a liver cancer project, that has been normalized by a R library should be identified, the file set is identified by the semantic repository and is then filtered by comparing it with workflow execution data from the provenance tracking catalogue [47].

5.7.7 Taverna

Taverna²⁵ has been developed in context of the myGrid²⁶ project which aims at providing middleware for executing data-intensive experiments in various research domains such as systems biology, chemistry and astronomy. Taverna was designed as a workbench for defining and executing scientific workflows, which are composed of services encapsulating domain-specific tools and applications. Since more and more scientific software is provided with publicly available service interfaces and research databases offer service-based

²⁵Taverna <http://www.taverna.org.uk>

²⁶myGrid <http://www.mygrid.org.uk/>

access, Taverna puts much emphasis on integrating these web resources. For instance, a wide range of bioinformatics tools have been supplied as web services by the BioMOBY project [102]. BioMOBY web services have been integrated into Taverna, allowing to arrange them in workflows and execute them on remote BioMOBY servers. Additionally, web service based access to the sequence analysis package EMBOSS (European Molecular Biology Open Source Software Suite) [82] has been established by using the Soaplap framework.

Taverna has a three-tiered system architecture in order to differentiate various abstraction layers. The Taverna workbench presents end users an application-oriented view on workflows which hides complex technical details. A service panel displays the set of available services (local and remote) whereas BioMoby and EMBOSS services are automatically integrated. Each service is considered as a processor which has a predefined set of input and output data. Workflows are defined in the proprietary Scuf language (Simple Conceptual Unified Flow Language) and executed by the open source workflow orchestration tool Freefluo. Scuf allows to specify workflows in a data flow centric perspective. Workflow tasks are modelled as processors, which are connected by data links, defining the input and output data for processors. Further, coordination-links are used to specify running order dependencies among processors. If a coordination-link exists from processor A to processor B, processor B postpones its execution, until processor A has been completed. The composition of workflows is supported by a visual workflow diagram editor, which allows to define initial workflow parameters (input ports), data flow links between processors and the final result set (output ports). Additionally, a hierarchical view of all workflow components is provided by the workflow explorer, which provides validation of services and defines particular runtime properties of services. That is, looped execution of single services may be specified and parallel invocation of multiple service instances is supported. The workflow enactor user interface is used for ex-

ecuting workflows, monitoring execution progress and displaying the results [69].

Workflows may be published and shared with other scientists on the my-Experiment platform [36], which is an online research environment for exchanging bioinformatics workflows. Currently, over 1,700 workflows are available in the repository. An example workflow, published by Antoon Goderis [37], is depicted in Figure 60.

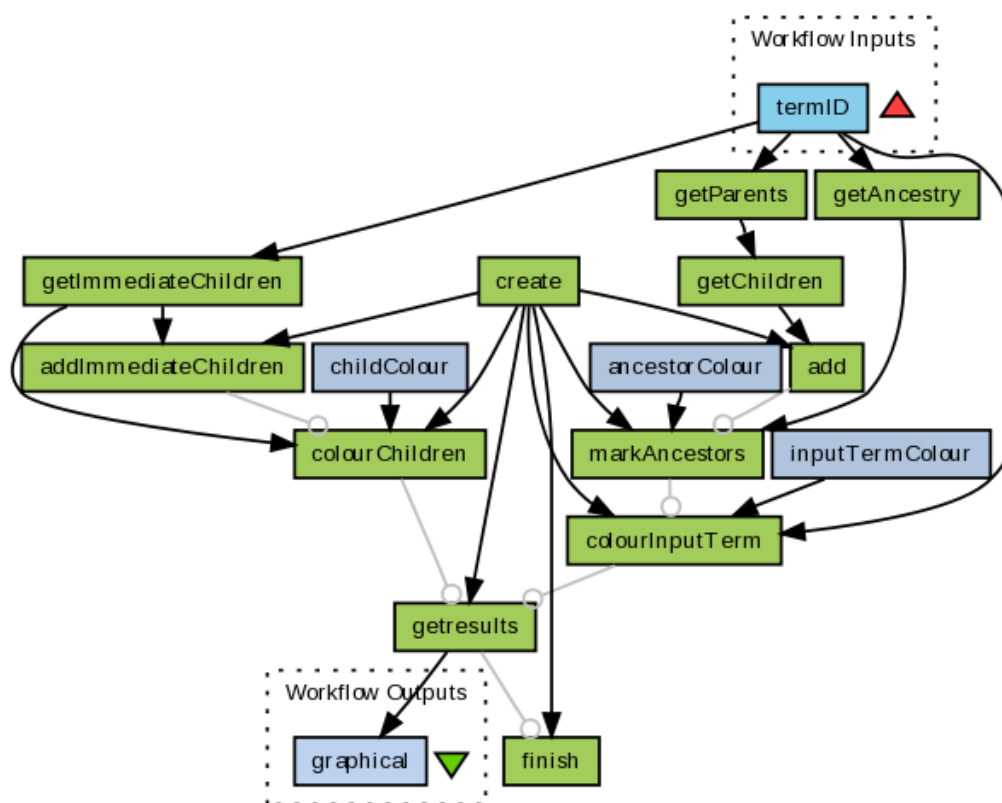


Figure 60: Taverna Example Workflow [37]

The illustrated workflow is used to visualize a subset of Gene Ontology terms graphically. The workflow receives a set of Gene Ontology terms as input ports, determines the set of related Gene Ontology terms, which may

be ancestor, child or sibling nodes and represents the entire set of input and deferred nodes in a graph. The workflow diagram consists of input and output ports (blue nodes), tasks which are implemented by SOAP services (green nodes) and local operations for emitting constant String values as service inputs (gray-blue nodes). Data flows are represented by arrow-connectors while control links are connectors with cycle end points.

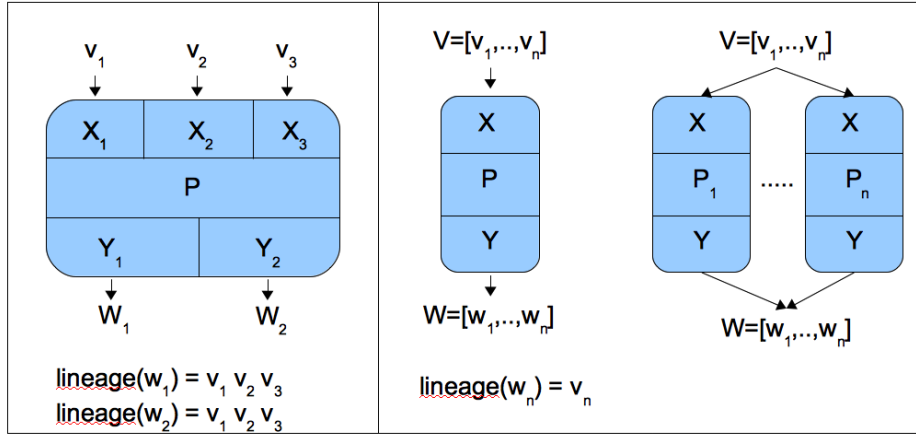


Figure 61: Taverna - Data Provenance Handling [62]

Taverna comes with a built-in component for capturing provenance data. When executing workflows, provenance data is recorded in a relational database - currently MySQL and Derby databases are supported. The ontology-based provenance model Janus [62] was designed which is capable of representing data dependency relations that are enriched by domain-aware semantic annotations. An exemplary fragment of Janus is depicted in Figure 61. Processor P realizes a search service for genetic pathways. Based on a set of genes, it retrieves from the KEGG database²⁷ all metabolic pathways, in which the input genes are involved. The service has domain-specific input types

²⁷KEGG database www.genome.jp/kegg/pathway.html

(Gene and KEGG) and output types (Pathway and KEGG), which are templates for data items. At runtime, typed data items are assigned to input and output ports of a service by *has_port_value* properties. P has three input ports (X_1, X_2, X_3) and two output ports (Y_1, Y_2) which are connected with data items.

The provenance model of Taverna facilitates fine-grained provenance tracking. That is, if collections of data items are processed by a service, data dependencies are captured for each data item separately. A service P that receives a data item V_1 and creates an output value W_1 , generates a new data dependency (a data lineage) between these data items. The data lineage of W_1 is defined as *data_lineage* (W_1) = V_1 . If P receives a collection of input data values $V = (V_1, \dots, V_N)$, an iteration over P is initiated, whereas a new instance of P is created for each input value. Each instance of P produces an output value W_i for every input value V_i . In this case, N new data lineage dependencies are created such that *data_lineage*(W_i) = V_i . The fine-grained provenance model allows detailed provenance queries. For instance, in the pathway search example a list of gene identifiers may be passed to the search service. The search service is executed one time for each gene identifier and separate result sets containing pathways and data dependency relations are created. Hence, it is possible to determine for every set of pathways on which gene it depends on. Though, the presented provenance model has some limitations. Collection-oriented capturing of provenance data does only work for processors expecting atomic values. A processor performing searches can be assigned a list of atomic search keys and it may be iterated several times producing an independent result set for each search key. However, processors that operate on lists of values can not be split in multiple processor iterations. For instance, a processor receives a list of patient data and computes the average survival of two patient groups and generates two results. In the Taverna provenance model, both results would be modelled as being dependent on the entire patient data, although each result depends only on

a subset of the patient list.

There is no graphical user interface for querying and displaying provenance data. However, provenance information may be extracted programmatically using the Java Provenance API and transformed in various output formats. On the one hand, the result of a provenance query may be exported as a RDF triple set based on the Janus model. On the other hand, the result set may be transformed into an OPM (Open Provenance Model) graph. Provenance data is assessed by capturing events released by the Taverna runtime and stored in a relational data base. An XML-based query language was developed allowing to formulate provenance queries. Result sets may be restricted by the following XML elements: target, focus and scope. Target allows to specify the set of output data values for which provenance information is to be extracted. Scope defines which workflow run should be considered and focus allows to filter the result data by specifying the set of processors whose generated data should be returned [61].

5.7.8 VisTrails

VisTrails²⁸ was developed at the University of Utah as a scientific workflow management system that allows scientists to define and execute workflows (dataflows) and assist data exploration of workflow results. The main focus and strength of VisTrails is on effectively supporting workflow evolution by providing graphical user interfaces for exploring versions of workflows and the generated data outputs. Therefore, VisTrails tracks all workflow changes in history logs and facilitates easy switching between workflow versions. Whenever the workflow definition is altered - parameters are changed or processing steps are added or removed - a new workflow version is created. The results of different workflow versions may be presented and compared in a spreadsheet-like view. Further, workflows may be executed repeatedly based on input sets of parameters [15].

²⁸VisTrails <http://www.vistrails.org>

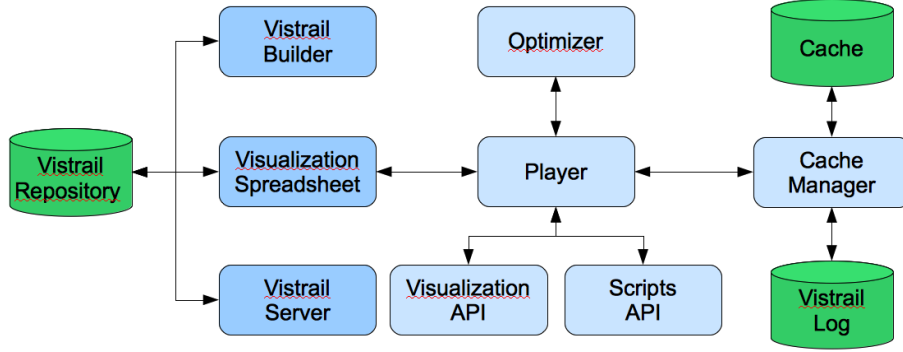


Figure 62: VisTrails System Architecture [15]

The system architecture of VisTrails is sketched in Figure 62. Workflows are defined and modified in the VisTrails Builder component which is a user interface for creating workflow graphs. Workflow tasks are called modules having input and output ports of a certain type. A large set of predefined modules has been integrated in a module repository. These modules implement standard operations like file handling or String manipulation as well as elaborated visualization techniques like polygon drawing or three-dimensional graphics. Further, user-defined Python scripts may be attached to modules and connected with other modules. An example workflow that was designed in the VisTrails Builder is presented in Figure 63. The illustrated workflow reads the content of a file and passes the file path and the content to the input ports of module ConcatenateString which simply joins all input Strings to an output String. Two other input ports of the module are filled with String constants. The result of the concatenation is sent to the standard output console. Additionally, a Python script is executed by a module and the return value is printed on the output console. Workflow definitions are saved in the VisTrails Repository and may be shared with other users. Alternatively, stored VisTrails workflows may be accessed and imported from a VisTrail server. The visualization spreadsheet is used for

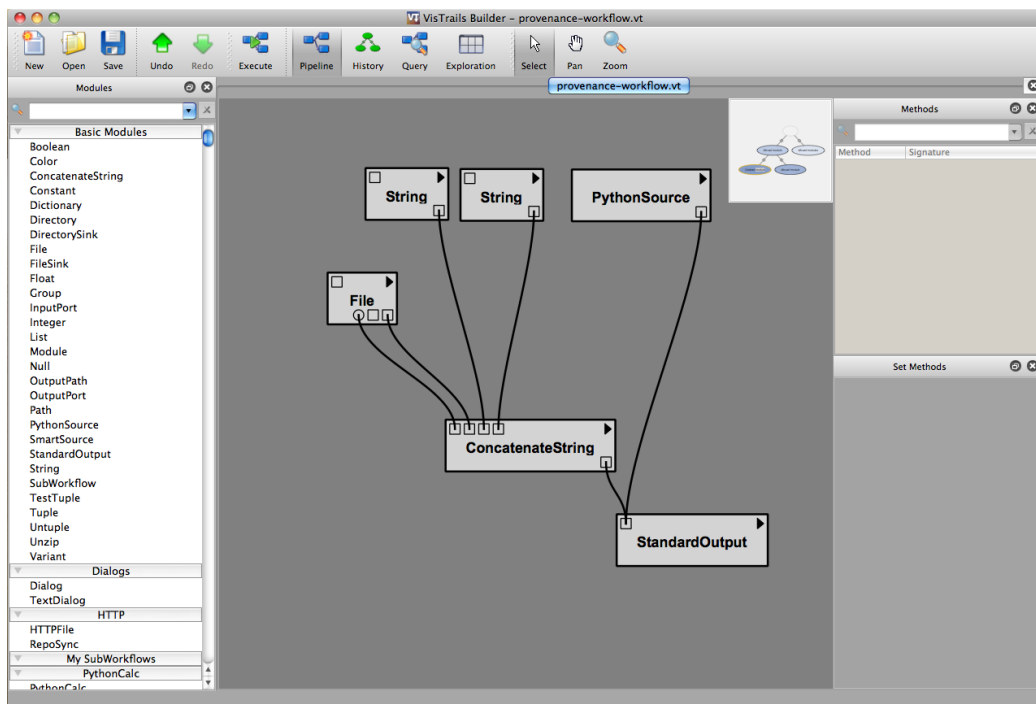


Figure 63: VisTrails Builder

collecting results of workflow executions. Different kinds of data may be represented by each cell. For instance, cells may contain graphics, render HTML pages or data collections. The enactment of workflows is performed by the Player Component which translates the workflow definition into a sequence of processing steps. These steps are executed by calls to the Scripts and Visualization APIs.

The execution of workflows is supervised by the Cache Manager, which keeps tracks on the current bindings of data to module ports and caches result data. If a module is to be reexecuted with identical input parameters, the corresponding results are retrieved from the cache. Evolution of workflows is supported by creating workflow versions for all changes. Each time a module is added, updated or removed or a data binding has changed, a new workflow version is generated and stored. Workflow versions are maintained as nodes

in a tree-based structure, whereas predecessor and successor relationships are modelled by edges. VisTrails uses an incremental representation of workflow versions. That is, a workflow version is based on a predecessor version to which a set of changes is applied.

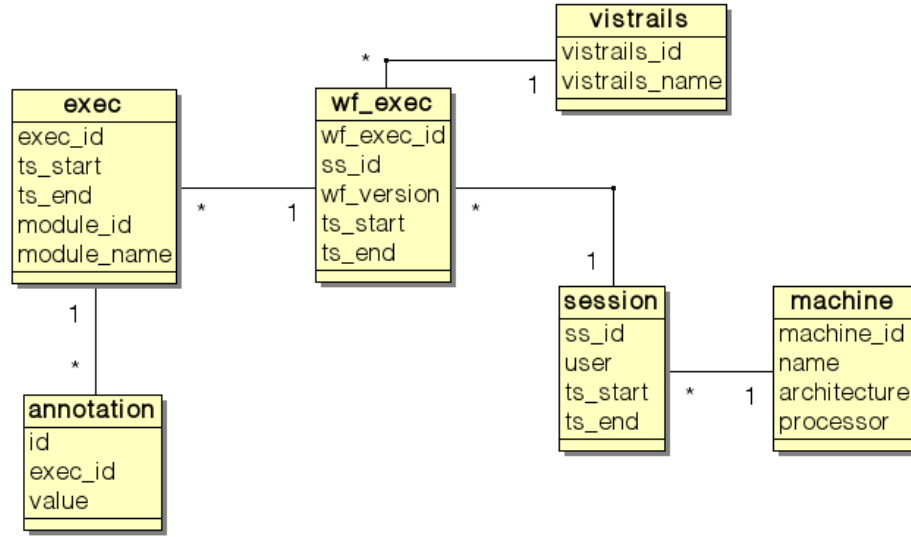


Figure 64: VisTrails Provenance Model

VisTrails took part in the First Provenance Challenge and succeeded in answering provenance queries Q1,...,Q9. Since VisTrails originally focused on managing provenance data about the evolution of workflows, the system had to be extended in order to capture provenance data from a more data-oriented perspective. A MySQL database was developed for logging execution data of workflow instances. The database schema of the provenance model is illustrated in Figure 64. Workflow definitions are stored in the VisTrails table, whereas each definition may have multiple workflow instances which are captured in the wf_exec table. A workflow execution consists of a set of module executions which are represented by entries in the exec table. Finally, all data bindings are modelled by annotation entries. Simple key-value pairs map data values to input and output ports. The provenance model does not

support typing of data values. That is, parameters may be searched only by qualified names, e.g. exact file names.

An algebraic query language has been designed for extracting data out of workflow definitions, workflow versions and workflow executions. The syntax of the language is similar to SQL, though it is enriched with functions for set-based operations. For instance, the function *upstream*(*x*) takes a module identifier *X* as an input and searches for all modules that have been executed before module *X* in a workflow. Provenance query Q1 can be answered by the following VisTrails query:

```
wf* : upstream(x) union x where x.module = FileSink
and x.parameter('name') = 'atlas - x.gif' and executed(x)
```

This query returns the set of modules that have been executed in order to generate the output file atlas-x.gif. As module FileSink created the output file, the closure of all modules, that precede FileSink in the workflow, is detected by the *upstream* function. The advanced provenance-capturing system has not been integrated into the current release of VisTrails [3].

5.7.9 Comparison of Scientific Workflow Management Systems

A short overview of the features and capabilities of the compared SWfMS is presented in Table 4. All of the systems provide components for designing and executing scientific workflows. While most systems have built-in editing tools, some systems are capable of importing workflow specifications defined in proprietary language. Pegasus and Wings may import workflows defined in DAX format and Discovery Net allows to represent workflows in DPML. A workflow execution engine is provided by each of the systems. Integration and execution of web services are essential features that have been implemented by all compared SWfMS. Thus, workflow tasks are executable as web services on local and remote resources. Capturing of data provenance was implemented in all systems except BioWBI and Discovery Net. Kepler, Pegasus and VisTrails realized tracking of data provenance at

file level. Kepler/pPod, Wings for Pegasus and Taverna implemented fine-grained tracking of data provenance that is capable of dealing with collections of data objects. Querying and extracting provenance data is supported by Kepler/pPOD, Wings, Taverna and VisTrails. While Kepler/pPOD offers a visual querying interface, the other systems provide access to provenance data by SPARQL queries and proprietary query languages. Compared with the requirements stated in Section 5.2, the capabilities of the systems do not cover all of the requested specifications. Web service based execution of the gene expression analysis workflow can be realized on all of the systems.

The collection-oriented tracking of data provenance is offered only by Kepler/pPOD, Wings and Taverna. Kepler/pPOD provides querying of data provenance only for single workflow executions. Thus, data objects used in multiple workflow executions are not detectable. Further, the provenance model of Kepler/pPOD does not take into account user-defined semantic annotations. Wings has a powerful semantic provenance model that may be enriched with domain-specific annotations. Provenance data may be only extracted by querying two separate data repositories (relational and semantic database) and by merging the results. Taverna emulates the behaviour of collection-oriented provenance tracking by iteratively executing a service on a list of input data and storing the dependencies between input and output objects. Though, data objects that depend on a set of input objects are not considered in this model. None of the compared systems is capable of dealing with changes in input data. Propagation of changes and validation of workflow results has not been realized up to now. GATiB-CSCW supports the execution of workflow tasks that are encapsulated in web services. Workflows may be defined by a web-based workflow editor. Workflow execution is accomplished by an external WS-BPEL engine (e.g. Apache ODE). Though, no built-in analysis libraries are provided; all required functions have to be realized by external web services. The provenance model of GATiB-CSCW records data transformation at a fine-grained level which allows to deduce

Name	Description	WF Design	WS Support	Libraries	Prov. Capturing	Prov. Query Interface	Data Changes
BioWBI	Application in life sciences	Web-based	Yes	EMBOSS	No	No	No
Discovery Net	Grid infrastructure	Visual Editor, DPML language	Yes + grid services	No	No	No	No
Kepler	Based on Ptolemy II Framework	Visual Editor	Yes + grid services	R, Matlab, database interfaces	file-level	No	No
Kepler/pPOD	Extension of Kepler, phylogenetic data analysis	Visual Editor	Yes	Same as Kepler	collection-oriented	Provenance browser	No
Pegasus	Specialized for execution on distributed resources (GRID)	Visual Editor, DAX (DAG XML)	Yes + grid services	Grid interaction services	file-level	No	No
Wings for Pegasus	Extension of Pegasus, supports semantic annotation	Visual Editor, DAX	Yes + grid services	Grid interaction services	semantic model, collection-oriented	Mixture of SPARQL and SQL	No
Taverna	myGrid, data-intensive experiments	Visual Editor, Scufi	Yes + grid services	BioMoby, EMBOSS	Semantic model, collection-oriented	Query API	No
VisTrails	Focused on workflow evolution	Visual Editor	Yes	Python library, visualization library	file-level	Algebraic query language	No
GATiB-CSCW	Mixture of collaborative and scientific workflow system	Web-based process editor	Yes	No	semantic model, collection-oriented	SPARQL	Yes

Table 4: Comparison of Scientific Workflow Management Systems

data dependency and data usage graphs for single objects. Further, the provenance model captures data selection criteria and represents them as semantic constraints. These constraints are used to validate workflow results in cases of changed input data. Since the provenance model records both collaborative and process-oriented data, novel types of provenance queries may be formulated, which is a unique feature compared to other scientific workflow management systems.

6 Anonymization of Sensitive Data

When releasing patient-specific data (e.g. in medical research cooperations) privacy protection has to be guaranteed for ethical and legal reasons. Even when immediately identifying attributes like name, address or day of birth are eliminated, other attributes (quasi-identifying attributes) may be used to link the released data with external data to re-identify individuals. In recent research, much effort has been put on privacy preserving and anonymization methods. In this context, k -anonymity [72] was introduced allowing to protect sensitive data by generating a sufficient number of k data twins. These data twins prevent that sensitive data is linkable to individuals.

6.1 k -Anonymity

The concept of k -anonymity requires that each distinct combination of quasi-identifying attributes occurs at least k times in a shared table [72]. Hence, a sufficient number of k data twins is used to mantle the individuality of persons. For instance, Table 5 contains diagnoses and personal data of patients. This table is to be released without disclosing the identity of patients. Thus, the sensitive diagnosis data must not be linkable to any patient. Obviously, patient name is an immediately identifying attribute and has to be removed, as illustrated in Table 6. Further, sex and age are quasi-identifying attributes that may be used to identify the patients. Since the last two en-

Name	Sex	Age	Diagnosis
Sam	Male	44	Pneumonia
Ferdinand	Male	42	Prostata carcinoma
Andrea	Female	32	Appendicitis
Margret	Female	32	Mamma carcinoma

Table 5: Patient Diagnoses

Sex	Age	Diagnosis
Male	44	Pneumonia
Male	42	Prostata carcinoma
Female	32	Appendicitis
Female	32	Mamma carcinoma

Table 6: Patient Diagnoses without Identifiers

tries have equal values for sex and age, these records are already protected from reidentification. However, the first two records are still identifiable. For instance, if somebody knows that the medical record of Ferdinand is part of the released data, and he knows his age, his diagnosis prostata carcinoma could be revealed.

When replacing the exact age by age intervals, the first two records have equal values for sex and age, and become indistinguishable. Table 7 is ($k=2$) 2-anonymous, as each record has at least one ($= k-1$) data twin.

k -anonymity may be accomplished by transforming attribute values to more general values, increasing the number of data twins. Nominal and

Sex	Age	Diagnosis
Male	40-45	Pneumonia
Male	40-45	Prostata carcinoma
Female	30-35	Appendicitis
Female	30-35	Mamma carcinoma

Table 7: Patient Diagnoses Anonymized

categorical attributes may be transformed by taxonomy trees or user-defined generalization hierarchies. Numerical attributes may be mapped to intervals. For instance, age 45 may be transformed to age interval 40-50. For a given data set, several k-anonymous anonymizations may be created depending on how attributes are generalized. Transformations of attribute values are always accompanied by an information loss, since exact values are replaced by generalized values. The information loss may be used as a quality criteria for an anonymization. That is, an optimal anonymization may be defined as the k-anonymous anonymization with the minimal information loss.

Although the idea of k-anonymization has been published several years ago, there is a lack of tools actually supporting the anonymization process. Therefore, we implemented the anonymization algorithm presented in [94] as a Java web application that may be deployed on a web application server and accessed by a web browser. Open_Anonymizer is a highly customizable anonymization tool providing the best anonymization for a certain context. That is, the anonymization process is strongly influenced by data quality requirements of users. We allow users to specify the importance of attributes as well as transformation limits for attributes. These parameters are considered in the anonymization process, which delivers a solution that is guaranteed to fulfil the user requirements and has a minimal information loss. Open_Anonymizer provides a wizard-based, intuitive user interface which guides the user through the anonymization process. Instead of anonymizing the entire data set of a data repository, a simple query interface allows to extract relevant subsets of data to be anonymized. For instance, in a biomedical context, diagnoses of a certain carcinoma type may be selected, anonymized and released without considering the rest of the diagnoses.

Since we developed Open_Anonymizer in context of the Austrian biobank initiative GATiB (Genome Austria Tissue Bank), we focused on anonymiza-

tion of patient diagnoses. However, `Open_Anonymizer` may be configured to work with arbitrary data repositories from different domains. `Open_Anonymizer` has a relational database interface allowing flexible access of database tables. Currently, two variants of best-first search algorithms are used to determine the optimal anonymization. Though, the design of `Open_Anonymizer` allows to flexibly integrate additional search strategies, e.g. heuristic search algorithms, with little effort. The search algorithms are strictly decoupled from data management and transformation components, which allows easy integration of additional algorithms.

6.2 Related Work

Several methods for achieving k-anonymity have been developed. Generally, we can distinguish between approaches using global recoding [101, 49, 34] of attribute values, those using constrained local recoding [50] and those using full local recoding [103]. Global recoding of attribute values may be defined as a set of functions that transform attribute values to generalized or altered values. In a single-dimensional global recoding all values of a certain attribute domain are mapped to transformed values, while multi-dimensional global recoding (also considered as constrained local recoding) recode combinations of attribute domains. Full local recoding is used to recode non-distinctive attribute values, that is, two tuples with equivalent attribute values may be recoded differently. `Open_Anonymizer` uses full-domain generalization which is a kind of single-dimensional global recoding. Each attribute is generalized independently from all the other attributes. Further, full-domain generalization guarantees that every attribute value belongs to the same generalized domain [49]. Although local recoding may produce solutions with lower information loss, we hold the opinion that global recoding generates solutions are more suitable for data analysis. For instance, if the overall survival of patients suffering from carcinoma is specified in various levels of granularity (93 days, 100-500 days, 150-200 days ..), no reasonable analysis may be made.

Therefore, Open_Anonymizer provides equal granularity for each attribute.

Open_Anonymizer has a wizard-based user interface controlling the anonymization process. Before the wizard may be started, a data repository containing data to be anonymized has to be registered, and the generalization hierarchies must be defined. Note: this tool may be run in a demo mode allowing to test the functionality and performance. In this case, no configuration is necessary as we provide a synthetic data repository that is filled with random data. Please see Section 6.8 for more details.

6.3 Generalization Hierarchies

Generalizing attribute values requires predefined generalization hierarchies. For instance, diseases may be coded according to the ICD-10 classification. The ICD-10 is an international standard diagnostic classification and was released by the World Health Organization. It allows a hierarchical classification of diseases, and has more than 16,000 entries [71]. Since the ICD-10 is an elaborated and widely-used classification system, it may also be used as a generalization hierarchy for data anonymization. In Figure 65 a subtree of the ICD-10 classification tree is displayed.

The classification hierarchy is based on 6 levels, starting with exact diagnoses at the bottom entries, which are categorized to more and more general terms upwards the hierarchy. For instance, all related entries (C22.0,..., C22.9) may be generalized to the common parent "Malignant neoplasm of liver and intrahepatic bile ducts ". A generalization hierarchy is composed of levels $L_0, ..L_t$, where L_0 is the most bottom level of unchanged values and L_t the most general level. These levels are used for both the specification of information losses and the creation of a search space for our anonymization algorithm.

Let $A = \{\alpha_1, .., \alpha_n\}$ be the set of attributes available in a data repository. Then each attribute α_i has an associated set of domain values which may be

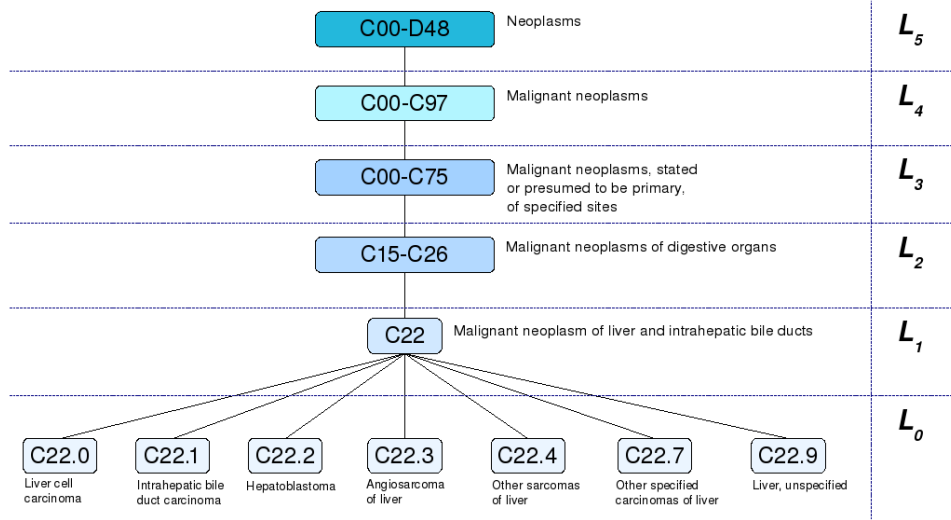


Figure 65: ICD10 Generalization Hierarchy

denoted as $D_0^{\alpha_i} = \{dv_1^{\alpha_i}, \dots, dv_m^{\alpha_i}\}$. The values of $D_0^{\alpha_i}$ may be considered as the set of unchanged values for attribute α_i . In terms of a database table, $D_0^{\alpha_i}$ is the set of distinct attribute values including *null* if the attribute has no IS NOT NULL constraint. Since a generalization hierarchy is composed of domain values of different levels, various domain value sets may exist for the entire hierarchy of an attribute. So let the set of domain values of a hierarchy be $DV_{\alpha_i} = D_0^{\alpha_i}, \dots, D_t^{\alpha_i}$, where t is the number of levels of the hierarchy. Domain values may be generalized by applying transformation rules. A transformation rule is a child-to-parent transformation $(dv_m^{\alpha_i}, dv_n^{\alpha_i})$, where $dv_m^{\alpha_i} \in D_k^{\alpha_i}$ and $dv_n^{\alpha_i} \in D_{k+1}^{\alpha_i}$. For instance, tumor staging T value 1a may be generalized to value 1 by the following relation $(1a, 1)$. The set of transformation rules, TR_{α_i} , interconnects the domain values of all levels and form implicitly the generalization hierarchy.

A generalization hierarchy for an attribute α_i may be specified as $H_{\alpha_i} = (DV_{\alpha_i}, TR_{\alpha_i})$, containing domain value sets of all levels and the corresponding transformation rules. At the top of each hierarchy, there is a general *ALL* value. The *ALL* value is the most general value of the hierarchy, meaning that all domain values are generalized to the same value. Since the values cannot be distinguished any more, the *ALL* value has the maximal information loss and corresponds to a suppression of the attribute. We use this hierarchy notation in the configuration of Open-Anonymizer for performance reasons. The generalization hierarchies of all attributes are loaded in the main memory which leads to a considerable speed-up of the anonymization process.

Depending on the type of the attribute, the generalization hierarchy may either be an interval-based tree for numerical attributes or a taxonomy tree for categorical attributes. While taxonomy trees have to be modelled explicitly, interval-based trees may be calculated. For example, age 43 may be generalized to intervals $[40 - 45]$, $[40 - 50]$ or $[40 - 60]$ by comparing the value with the interval borders.

The configuration file for the generalization hierarchies of attributes is `/web/conf/attribute_hierarchies.txt`. One line of the configuration file corresponds to the generalization hierarchy of one attribute. Each line is structured as follows:

`{ATTR.INFO}%{LOSS.INFO}%{VALUE.INFO}%{HIERARCHY.INFO}`

1. **ATTR.INFO:** The info entry has the following syntax `{attribute id | attribute type}`. The attribute id is the same attribute identifier used in the data repository configuration 6.7. The attribute type specifies whether the attribute is "Categorical" or "Numerical". Categorical attributes have a predefined set of values that may be generalized by taxonomy trees. Numerical attributes have values out of a certain range of numbers. For instance, patient age may be specified

as an interval $[0, 100]$.

2. **LOSS_INFO:** This entry defines the degree of information loss along the generalization hierarchy. We measure the information loss as a value between 0.0 and 1.0. Domain values of level 0 (unchanged values) have an information loss of 0 (= 0%), while the most general *ALL* value has an associated information loss of 1.0 (=100 %). The intermediate generalization levels ($D_1^{\alpha_i}, \dots, D_{t-1}^{\alpha_i}$) have information losses greater than 0 and less than 1.0. The information losses are defined relatively to level 0. For instance, an information loss of 0.6 corresponds to a 60 % information loss compared to the unchanged values of level 0. These values are customizable allowing to specify appropriate information losses. That is, a domain expert may define the information losses by evaluating the different levels of the hierarchy. The relative information losses do not have to equal for all generalization levels. Our anonymization algorithm takes into account the information loss metrics when searching for a k-anonymous generalization having an overall minimal information loss. The syntax of the entry is as follows: $\{t|IL_1, \dots, IL_t\}$. t specifies the number of generalization levels and $IL_1, \dots, IL_t$ are the information losses for all generalization levels. IL_t is always 1.0.
3. **VALUE_INFO:** For categorical attributes, this entry contains the domain values of the generalization levels. It has the following syntax $\{D_0^{\alpha_i}, \dots, D_{t-1}^{\alpha_i}\}$. Note: the domain value set $D_t^{\alpha_i}$ needs not to be specified, as it simply contains the *ALL* value. For numerical attributes, the numerical range of attribute values is defined in the following way $\{lower_bound, upper_bound\}$.
4. **HIERARCHY_INFO:** For categorical attributes, this entry contains the set of transformation rules TR_{α_i} . The transformation rules are defined in the following syntax: $\{tr_1|tr_2|\dots|tr_{n-1}|tr_n\}$. The transformation

rules are ordered by generalization levels. That is, first all transformation rules from $D_0^{\alpha_i}$ to $D_1^{\alpha_i}$ are listed, then all from $D_1^{\alpha_i}$ to $D_2^{\alpha_i}$ and so on. Note, the transformation rules from $D_{t-1}^{\alpha_i}$ to $D_t^{\alpha_i}$ do not have to be defined. While categorical attributes require an explicit specification of the generalization hierarchy, for numerical attributes only the range of the interval has to be defined. For instance, the specification of an age attributes may be $\{5, 10, 20, 40\}$.

6.4 Search Space

Generally, a k-anonymous solution is created by searching for a generalization that transforms a data set into an other data set that is k-anonymous. The original data set is composed of unchanged attribute values $D_0^{\alpha_i}$. Thus, all attributes of the data set are at generalization level L_0 . Each attribute α_i may be generalized up to its maximal generalization level $Lmax^i$. If all values of an attribute $D_0^{\alpha_i}$ are replaced by values of a higher level, $D_j^{\alpha_i}$, $0 < j \leq Lmax^i$, we call it a transformation for attribute α_i from L_0 to L_j . A generalization is a set of transformations, whereas for each attribute at most one transformation is defined.

For a given data set, various generalizations may produce k-anonymous data sets. Though, with respect to a minimal (weighted) information loss, a generalization transforming the data set as little as possible is to be found. The number of potential generalizations is determined by the number of generalization levels of the attributes, and may be calculated by the following formula: $\pi_{i=1}^n Lmax^i$. Figure 66 illustrates a search space lattice containing all possible generalizations for two attributes α_i and α_j , where $Lmax^i = 4$ and $Lmax^j = 3$, yielding 12 different states in the search space. The size of the lattice grows extensively, when adding more attributes. For instance, the 8 attribute hierarchies in our demo mode allow to create a lattice of 54,432 search nodes. A simple breadth-first algorithm would consecutively

create generalizations for all search nodes traversing bottom-up the lattice. By contrast, our anonymization algorithm ranks all search nodes based on their weighted information loss. Nodes complying with the user requirements are explored at first, while other nodes are tested, if no solution could be found. Since all search nodes of the lattice are visited, our anonymization algorithm is guaranteed to find an anonymization, if one exists. Generalization limits are used to prune the search lattice, as all nodes violating the limits would produce unacceptable solutions. Therefore, the algorithm delivers an anonymization that is as close to the user requirements as possible. However, generalization limits may cut areas of the search space that contain valid anonymizations, which may impede to find a solution at all.

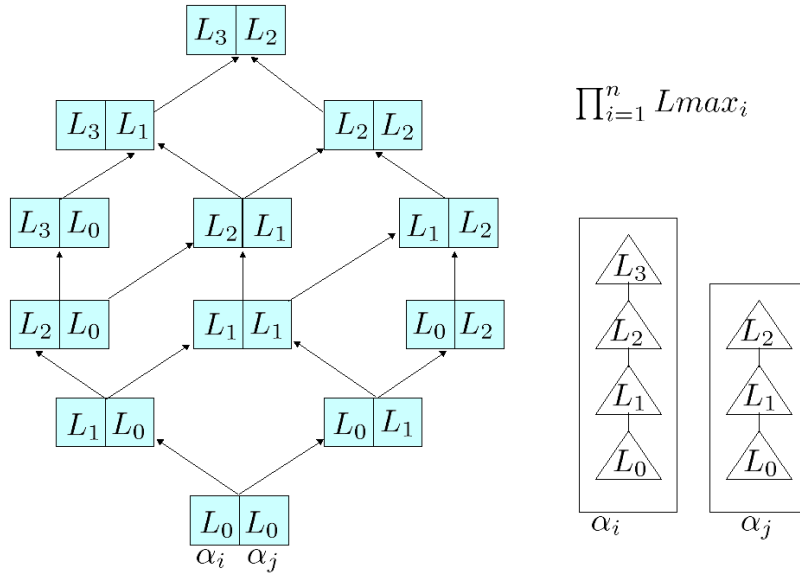


Figure 66: Anonymization Search Space

6.5 Anonymization Algorithm

In the following, the details of the anonymization algorithm are described. The following input parameters are required: T_{Anon} is the set of equivalence classes that violates the k-anonymity constraint. The k-Anonymity parameters is specifying the number of requested data twins. However, that parameter has to be greater than the minimal value claimed by a general security policy. A priority vector for all α_i is specified as $priorV[n]$. The priority values are in the range $[0, \dots, 1]$, whereas the most import attribute has the highest priority value and all differences between any two consecutive priorities values are equal. The generalization limits are stored in vector $limV[n]$ and a level vector $levelV[n]$ stores the current generalization levels for all attributes - initially, all level values are set to 0. $levelV[n]$ is an input/output parameter recording the generalization levels of the current search node. The algorithm returns a k-anonymous data set T_{Result} or null, if no anonymization solution could be found.

Algorithm *Anonymization Algorithm*

Input: T_{Anon} , $priorV[n]$, $limV[n]$, $levelV[n]$, k

Output: T_{Result}

1. $Search_Space = \{node(L^1, \dots, L^n)\} \bullet$
2. $0 \leq L^i \leq Lmax^i \wedge L^i < limV[i]$
3. $ranked_search_nodes = rank_nodes(Search_Space)$
4. **while** T_{Anon} not fulfils $k - anonymity$
5. **do if** $ranked_search_nodes.end_of_sequence()$
6. **then return** null
7. $node(L^1, \dots, L^n) = ranked_search_nodes.next()$
8. **for all** $\alpha_i \in QI$
9. **do** $T_{Anon} = generalize(T_{Anon}, \alpha_i, L^i)$
10. $level[\alpha_i] = L^i$
11. $T_{Anon} = merge(T_{Anon})$
12. $T_{Result} = T_{Anon}$
13. **return** T_{Result}

Initially, the search space is determined by creating all possible combinations of generalization levels for all attributes. The maximal generalization level of an attribute α_i is determined by its $Lmax^i$ value and by the user-defined limit $limV[i]$. For all search nodes, the weighted information loss is calculated. Let $info_loss(\alpha_i, L_j)$ be the cumulative information loss for attribute α_i at level L_j . Then the weighted information loss for search node $node(L^1, \dots, L^n)$ may be calculated by the formula: $\sum_1^n info_loss(\alpha_i, L^i) * prioV[i]$. In the next step of the algorithm, method *rank_nodes* creates a list of search nodes (*ranked_search_nodes*), which are sorted by their weighted information losses in ascending order. Iteratively, generalizations are generated for all search nodes until an anonymization that transforms T_{Anon} into a k-anonymous data set is found. A generalization is a set of transformations, whereas each transformation generalizes all values of a certain attribute up to a specified generalization level. Every attribute-based transformation is performed by method *generalize*. After all attribute value transformation have been executed, *merge*(T_{Anon}) creates a new equivalence class set by grouping distinct attribute values of T_{Anon} to equivalence classes. If each equivalence class has at least an associated cardinality of k, a solution has been found.

6.6 Performance Issues and Optimizations

We implemented two variants of the search algorithm: **best-first** and **best-first-advanced**. The best-first variant is most suited for small or medium-sized data sets (up to 10,000 records). It creates replicas of the data set in order to speed-up the transformation process. Thus, for each search node a replica is created and held in the main memory. The advantage of this implementation is the availability of intermediate data allowing to transform data sets quicker. For instance, if the search node $\langle L_2, L_2 \rangle$ has already been explored, search node $\langle L_2, L_3 \rangle$ may be derived by simply transforming the second attribute α_j from L_2 to L_3 , without changing the

first attribute α_i . The big disadvantage of this variant is the extensive main memory consumption for large data sets. The best-first-advanced implementation holds only one replica of the data set in the main memory. Whenever a new search node is reached, all transformations are applied to the replica of the data set and the test for k-anonymity is done. If the k-anonymity constraint can not be fulfilled, the replica is discarded, and a new search node is selected. For instance, search node $\langle L_2, L_3 \rangle$ is derived by transforming the original data of node $\langle L_0, L_0 \rangle$ accordingly. Although search node $\langle L_2, L_2 \rangle$ has already been visited, both attributes are transformed again based on the original data set. The disadvantage of this implementation is the repeated execution of transformations for similar nodes. Though, by efficiently storing of the generalization hierarchies, the transformation time is equal for all generalization levels. That is, a transformation from level L_0 to L_1 is as fast as a transformation from L_0 to L_5 . The best-first-advanced implementation guarantees a nearly constant memory consumption during the execution of the algorithm. For example, 270 MB main memory is needed for the anonymization of 20,000 records traversing 1,455 search nodes.

6.7 Data Repository

We assume a data repository as a table (or view) of a relational database system. Currently, our data interface allows to access arbitrary tables of a PostgreSQL database server. Though, access to other database systems (Oracle, MySQL,..) may be easily added with little effort. A data repository may be registered by editing the file `/web/conf/data_repository.xml`.

```
<xml>
<synthetic activated="true" number_of_records="500000"/>
<dbtype>          postgresql          </dbtype>
<dbhost>          localhost            </dbhost>
<port>            5432                 </port>
<database>        patientRepository   </database>
<user>            ***your_user***      </user>
```

```

<password>      ***your_password***      </password>
<repository table_name="diseases_replica">
  <attribute id="Topology" label="Topology" sql_name="code_value"/>
  <attribute id="Staging_T" label="Staging_T" sql_name="t"/>
  <attribute id="Staging_N" label="Staging_N" sql_name="n"/>
  <attribute id="Staging_M" label="Staging_M" sql_name="m"/>
  <attribute id="Grading" label="Grading" sql_name="g"/>
  <attribute id="R" label="R" sql_name="r"/>
  <attribute id="V" label="V" sql_name="v"/>
  <attribute id="Age" label="Age" sql_name="age"/>
  <attribute id="disease_id" sql_name="disease_id" primary_key="true"/>
</repository>
</xml>

```

The first line of the configuration file contains the *synthetic* element which controls, whether Open-Anonymizer is run in demo mode (see section 6.8 for more details). If a real data repository is used the attribute *activated* should be set to *false*. Specify the IP address (or localhost for local installations) of the database server inside the *dbhost* element. *port* holds the database port and *database* the name of the database. Further, enter a valid database user and his password into the corresponding elements. The settings for the data repository may be configured inside the repository element. Enter the name of the table or view into the *table_name* attribute.

Finally, the attributes of the data repository have to be specified. For each attribute element, the following attributes may be defined:

1. **id**: is a unique identifier, used to distinguish the attributes. The attribute id corresponds to the one used in the configuration of generalization hierarchies in Section 6.3.
2. **label**: specifies the label of the attribute in the user interface.
3. **sql_name**: is the sql name of the attribute in the database table.

4. **primary_key**: specifies, whether the attribute is a primary key in the database table.

The database table may contain additional attributes, not defined in the configuration file. Solely the attributes used for the anonymization do have to be defined.

6.8 Anonymization Wizard

Open_Anonymizer may be tested in a demo mode without requiring a real data repository. The demo mode is activated by editing the configuration file and setting the attribute *activated* of element *synthetic* to true. In this case, the data repository configuration is ignored and random data is created for test purposes.

```
<synthetic activated="true" number_of_records="500000"/>
```

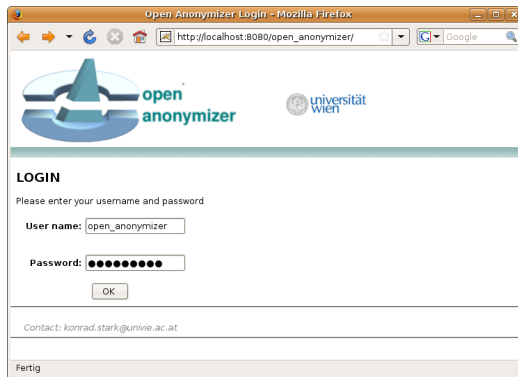


Figure 67: Open Anonymizer - Login Mask

Attribute *number_of_records* determines the size of the synthetic data repository. Test data is generated by selecting random values of domain $D_0^{\alpha_i}$ for each attribute α_i specified in the configuration file *attribute_hierarchies.txt*. The generated attribute values are equally distributed, as we do not include

frequency parameters in the generation of random values. After the application is deployed, the synthetic data is created and remains in the main memory.

In the following we describe the steps of the anonymization wizard. We assume data anonymization as a sensitive process which requires authorized users. The login mask displayed in Figure 67 asks for a valid user name and password. Please enter user name "open_anonymizer" and password "anonymize" in our demo installation. Additional users may be added by changing the class file `gui.LoginHandler.java`. Though, the authentication should be accomplished by an external database or LDAP server. We plan to provide an adequate authentication interface in the next release.

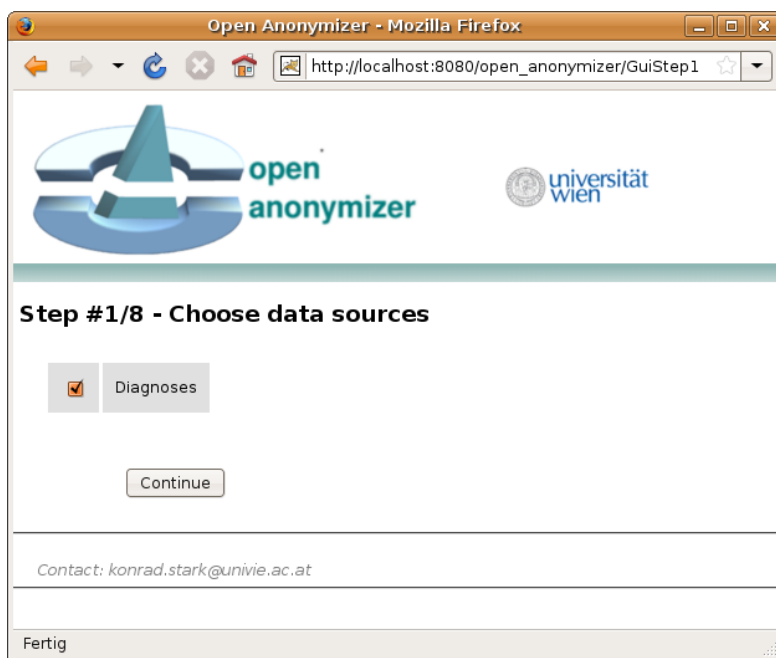


Figure 68: Open Anonymizer - Data Source Selection

Data source selection: A data source is a data repository containing sensitive data. As illustrated in Figure 68, this step allows to select the data repository that should be used for the anonymization. Currently, only

one data repository may be handled, since we have not implemented distributed anonymization up to now. The distributed anonymization will allow to anonymize multiple data repositories concurrently. For instance, if diagnoses and treatments of patients are stored in two separate repositories, we generate k-anonymous data sets for both repositories, join the result sets and release them.

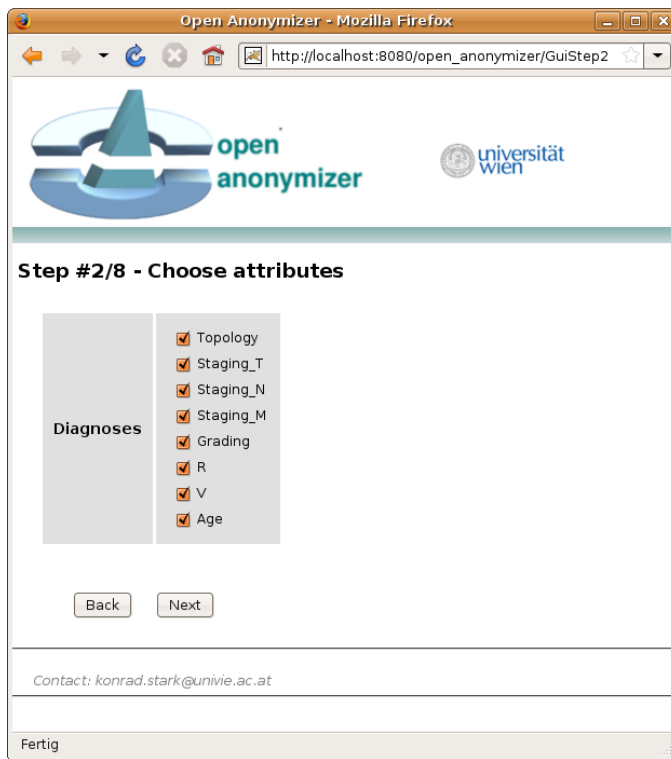


Figure 69: Open Anonymizer - Attribute Selection

Attribute selection: As only a subset of all available attributes may be required, the relevant attributes may be selected (see Figure 69). The fewer attributes are selected, the more likely is the presence of data twins in the data repository. Further, the likelihood of finding a k -anonymous solution is higher and the overall search space is smaller. The demo installation allows to select up to eight attributes. For a quick demonstration of the tool, we encourage to select five or six arbitrary attributes which allows to find an anonymization within a few seconds.

Figure 70: Open Anonymizer - Data Selection

Selection criteria: Open Anonymizer allows to select a subset of records from the data repository. A simple query interface is provided (see Figure 70) in order to specify query selection criteria. For instance, if a medical study is based on small tumors with metastases, staging $T=1$ and $M=1$ may be specified. A subset of records may be safely extracted and anonymized without breaching the k -anonymity constraint [95]. If a selection criteria is defined for an attribute, it is excluded from the anonymization process. In our example, all of the selected diagnoses would have equal values for staging

T and staging M. Hence, by generalizing these two attributes no additional data twins could be created. Thus, defining a selection criteria for an attribute is the same as setting the generalization limit to "unchanged". The sets of selectable values in the combo boxes are the corresponding $D_0^{\alpha_i}$ sets. For simplicity, only the conjunction of filter criteria is supported. Though, optional disjunction of filter criteria as well as selection of multiple attribute values will be supported in the next release. The number of records that match the selection criteria is displayed together with the query response time.

Attribut	Limit
Topology	[A00-B99] [C00-D48] [D50-D89] [E00-E90] [F00-F99] [G00-G99] [H00-H59] [H60-H95] [I00-I99] [...] [v]
Staging_T	[*] [v]
Staging_N	[unchanged] [v]
Staging_M	[*] [v]
Grading	[*] [v]
R	[*] [v]
V	[*] [v]
Age	[*] [v]

Back Next Reset

Contact: konrad.stark@univie.ac.at

Fertig

Figure 71: Open Anonymizer - Specification of Generalization Limits

Generalization limits: Figure 71 displays the user interface for specifying the generalization limits of attributes. For each attribute, a generalization limit between $D_0^{\alpha_i}$ (= unchanged values) and $D_t^{\alpha_i}$ (= most general ALL value, attribute is suppressed) may be chosen. The combo boxes contain example values for the domain values of the hierarchy. For attributes with small domain value sets, all domain values are displayed. For attributes having a

large set of domain values (e.g ICD-10 coded Topology), only a summarized presentation is displayed.

Open Anonymizer - Mozilla Firefox

http://localhost:8080/open_anonymizer/GuiStep5

open anonymizer universität wien

Step #5/8: Specify priorities of attributes

Drag and drop attributes to assign priorities.
The first row has the highest priority
the last row has the lowest priority.

Diagnoses
Age
Diagnoses Staging_N
Diagnoses Staging_M
Diagnoses Grading
Diagnoses R
Diagnoses Staging_T
Diagnoses V
Diagnoses Topology

Back Next Reset

Contact: konrad.stark@univie.ac.at

Fertig

Figure 72: Open Anonymizer - Priorities of Attributes

Priorities of attributes: The relative importance of attributes for a certain context may be specified by ranking them in a priority list. As illustrated in Figure 72, the chosen attributes appear in a sortable list. The order of attributes reflects their importance, with more important attribute at the top of the list and less important attributes at the bottom. Attributes may be ordered as required by drag and drop. The priorities of attributes are considered in the anonymization algorithm. They are used to weight the

information loss of a generalization from $D_i^{\alpha_i}$ to $D_{i+1}^{\alpha_i}$. Thus, attributes with lower priorities tend to be more generalized than attributes with higher information loss. More details about weighting in our anonymization algorithm can be found in [94] and [95].

Open Anonymizer - Mozilla Firefox

http://localhost:8080/open_anonymizer/GuiStep

open anonymizer universität wien

Step #6/8 - Algorithm settings

k parameter: 3

Search type: Best first advanced

Threshold: 0.5 %

Output file: data.csv

Back Next

Contact: konrad.stark@univie.ac.at

Fertig

Figure 73: Open Anonymizer - Algorithm Settings

Algorithm settings: The anonymization algorithm may be configured in a separate mask (Figure 73). As mentioned in Section 6.4, two variants of the anonymization algorithm have been implemented. The memory-saving best-first-advanced implementation is the default setting. Simple best-first may be selected, but should only be used with small data sets ($< 10,000$ records) and few attributes (≤ 6). Further, the minimal number of data twins may be specified by the k parameter. By default, k is set to 3. The threshold parameters allows to exclude records that violate the k-anonymity constraint. We use this parameter to eliminate outliers from the result set. That is, if a k-anonymous generalization is found for nearly all records of the data set,

and just a few records are still violating the constraint, it may be reasonable to remove them instead of trying to find an other solution. In the worst case, a few records could cause an extensive information loss, or they may impede that a valid anonymization is found at all. The threshold parameter is the terminating condition for the anonymization algorithm, defining the sufficient number of k-anonymous records. A threshold of 0.5 % means, that 99,5 % of the records may be generalized to a k-anonymous solution. Currently thresholds of 0.0, 0.1, 0.25, 0.5 and 1.0 % may be specified. If a k-anonymous solution is found, the result set is available as a csv file. The name of the csv file may be defined in the field "Output file".

Step #7/8 - Check parameters

Data source	Attribute	Limit	Priority
Diagnoses	Age	5	0.11
Diagnoses	Staging_N	0	0.22
Diagnoses	Staging_M	2	0.33
Diagnoses	Grading	2	0.44
Diagnoses	R	2	0.55
Diagnoses	Staging_T	3	0.66
Diagnoses	V	2	0.77
Diagnoses	Topology	5	0.88

k parameter **Search type** **Threshold** **Output file**
 3 Best first advanced 0.5 % data.csv

Contact: konrad.stark@univie.ac.at

Fertig

Figure 74: Open Anonymizer - Parameter Summary

Check paramters: After defining all parameters for the data set and the algorithm, a summary page displays all settings (see Figure 74). As

the user interface is wizard-based, the user may navigate back step by step and change the parameters. In order to confirm the correct parameters, the "Start Algorithm" button triggers the execution of the algorithm.

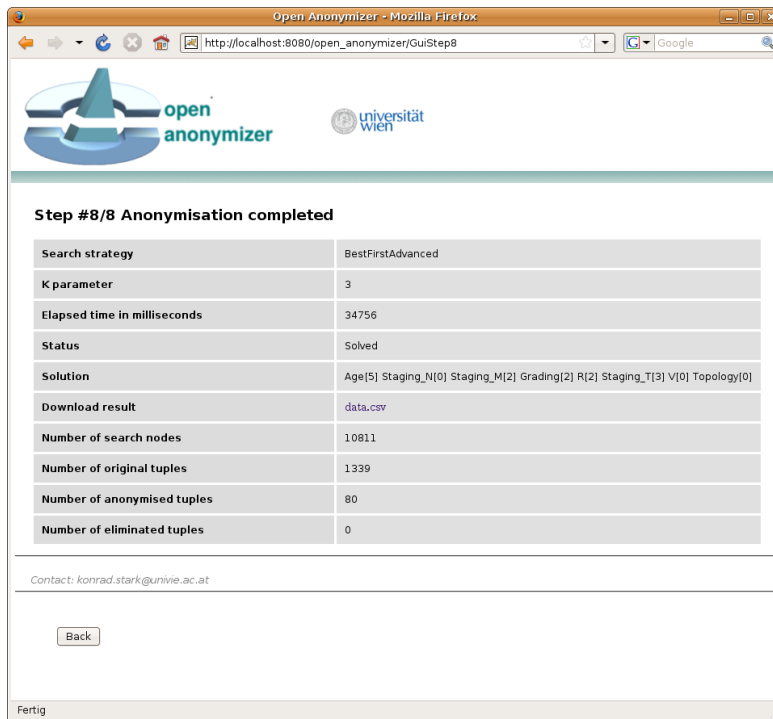


Figure 75: Open Anonymizer - Result of Anonymization

View result: The final mask (see Figure 75) displays the result of the anonymization. If no suitable solution could be found, the status label displays this information. In this case, the anonymization may be restarted with modified parameters by navigating back in the wizard. There may be several reasons for an unsuccessful anonymization. Most likely, the generalization limits have been set too conservatively, pruning all parts of the search space that contain valid anonymizations. If no limits are set, the algorithm also will find solutions with suppressed attributes. For instance, if no solution with 7 attributes can be found, the algorithm searches for solutions with 6 attributes, starting with attributes with low priorities. Moreover, too strin-

gent parameters for k-anonymity or threshold may be responsible. If a valid anonymization is found, a detailed description of the solution is displayed. The runtime of the algorithm is shown as well as the k-anonymity parameter and the chosen search strategy. Further, the generalization levels of the solution are displayed for all attributes: *attribute_name* [*level*]. The number of search nodes is the number of nodes that have been visited before the solution was found.

The number of original records is the number of records selected from the data repository. The number of eliminated records specifies the number of records that have been discarded due to the threshold constraint. The anonymized data consists of data twin groups, where each data twin group has equal values for all attributes and has at least k members. The result set may be downloaded by clicking on the linked csv file. The file may simply be opened by Microsoft Excel or OpenOffice Calc. The entries of the file are ordered by data twin groups. The first line of the file contains the attribute names plus one control column counting the number of data twins in each group. Details about the installation and configuration of Open_Anonymizer are given in the Appendix.

7 Conclusion

This thesis covers various aspects and challenges which are related to the design and realization of an IT-infrastructure for medical research activities. Biomedical research is a complex process in an interdisciplinary environment, bringing together experts and knowledge of manifold domains. Although the research activities of medical, biological and chemical scientists are quite different, they are strongly coupled with each other and operate on common data. By realizing the medical cooperative system GATiB-CSCW, researchers are capable of accessing, structuring, annotating and sharing data from different distributed sources in a flexible environment. The novel aspects of the system are the representation of data in virtual knowledge spaces, the self-organizing possibilities in research groups and the modern service-oriented architecture which allows a seamless integration of external services and Web 2.0 tools such as instant messaging or wikis.

Scientific Workflows are deployed in a multitude of scientific areas for managing the evolving requirements concerning automation of complex research processes, traceability and reproducibility of research results. Scientific workflow management systems strive to support the planning, executing and documenting of so-called virtual experiments. In context of the GATiB initiative, a scientific workflow management system was required for conducting gene expression analyses. The results of gene expression analyses are used for understanding courses of disease and for developing disease treatments such as tumor markers or tumor suppressor genes. An IT infrastructure was developed for accessing gene expression profiles and medical data from distributed sources and for step-wise executing customized gene expression analyses. Statistical libraries were encapsulated into web services and analysis results made persistent in an experiment database. Furthermore, a web-based process editor for defining workflows and a web-based process execution component for enacting workflows were developed.

Data provenance protocols the origin of data. It may be considered as

a production plan of which data sources are accessed and transformed by which component in order to generate a certain data object. In context of scientific workflows, data provenance is used to understand and verify workflow results. Established scientific workflow management systems frequently lack the recording and querying of provenance data. Some systems provide capturing of provenance data only at file-level granularity, others do not offer expressive query functionality. The semantic provenance model presented in this thesis allows to track data provenance data at the object level. Hence, it is possible to deduce fine-grained data dependency graphs based on single or collections of data objects. The provenance model is the first one that is capable of managing data changes in input data of scientific workflows. By assigning validity constraints to input objects, semantic reasoners are able to infer which data changes do have an impact on executed workflows. This allows to identify invalid workflow results and to reexecute workflows with valid input data. Currently, provenance data may be queried by formulating SPARQL-queries. In future work, a graph-based user interface could be created allowing non-technical users to interactively navigate through data dependency graphs and to query provenance data based on predefined filter criteria and patterns.

Since medical data is a valuable source in medical research projects, protection of sensitive patient-related data is a key issue. The anonymization algorithm, presented in this thesis, guarantees k -anonymity in a released medical record set. The anonymization is strongly influenced by data quality requirements. User-defined prioritization of attributes of the released record set are taken into account as well as information loss quantifies that are specified in the generalization hierarchies of the attributes. The algorithm generates a k -anonymous data set with minimal information loss with regard to user-specific parameters. The algorithm was realized as a Java web application and is the first open source implementation of a k -anonymization technique.

8 List of Figures

1	Biobank Categories	16
2	Research Purposes of Different Kinds of Biobanks [83]	19
3	GATiB - Biological Samples Stratified by Donor Organs	25
4	Nucleotides	28
5	Gene Expression Analysis with Microarrays	41
6	Wasabi - Virtual Knowledge Space	50
7	Wasabi - System Architecture	52
8	Integration of GATiB-CSCW in the MUG IT Infrastructure	53
9	Life Cycle of a Scientific Workflow [55]	57
10	Gene Expression Analysis	62
11	Gene Ontology Example	67
12	Gene Expression Analysis Workflow - Details	68
13	Data Changes in Medical Data I	74
14	Data Changes in Medical Data II	75
15	Data Changes in Genetic Data	76
16	Types of Data Changes	78
17	Open Provenance Model - Interoperability [65]	89
18	Open Provenance Model - Nodes and Dependencies [63]	90
19	BPMN Model of Gene Expression Analysis Workflow	92
20	GATiB Scientific Workflow System Architecture	97
21	Gene Expression Workflow - Patient Grouping User Interface	100
22	Gene Expression Workflow - Gene Group Analysis User Interface	102
23	Gene Expression Workflow - Gene Group Analysis Results User Interface	103
24	Gene Expression Workflow - Gene Group Analysis Plots	104
25	Gene Expression Workflow - Gene Group Analysis Plots	104
26	Microarray Experiment Database Schema	108
27	Adapted System Architecture of GAMECS-CSCW	112
28	Process Editor - User Interface	114

29	Process Editor - Example Process	116
30	Process Database Schema	117
31	Process Execution Component - Parameter Passing	129
32	Process Execution - User Interface	135
33	OPM - Semantic Provenance Model Match - Entities	136
34	OPM - Semantic Provenance Model Match - Relationships	137
35	Data Provenance Model Overview	139
36	Different Versions of a Resource in Various Contexts	157
37	Gene Ontologies as Service Call Outputs or Annotations	159
38	Provenance Model - Java API	161
39	Example Process as BPMN	163
40	Example Patient Record Set	164
41	Provenance Model - Register Structured Data Type	167
42	Provenance Model - Register Service	170
43	Provenance Model - Register Process	172
44	Provenance Model - Enact Process	176
45	Provenance Model - Register Resource	177
46	Management of Data Changes in Provenance Data	179
47	Provenance Model - Capture Data Changes	184
48	Provenance Model - Inconsistency Detection	187
49	Provenance Model - Classification	188
50	BioWBI Architecture [51]	195
51	Discovery Net Architecture [85]	197
52	Kepler Example Workflow	200
53	Kepler Architecture	201
54	Kepler Provenance Model	202
55	pPod Kepler - Data Provenance Tree	205
56	pPOD Kepler - Provenance Browser	206
57	Pegasus System Architecture	209
58	Wings Architecture [35]	211

59	Wings Provenance Model	212
60	Taverna Example Workflow [37]	215
61	Taverna - Data Provenance Handling [62]	216
62	VisTrails System Architecture [15]	219
63	VisTrails Builder	220
64	VisTrails Provenance Model	221
65	ICD10 Generalization Hierarchy	230
66	Anonymization Search Space	234
67	Open Anonymizer - Login Mask	239
68	Open Anonymizer - Data Source Selection	240
69	Open Anonymizer - Attribute Selection	241
70	Open Anonymizer - Data Selection	242
71	Open Anonymizer - Specification of Generalization Limits	243
72	Open Anonymizer - Priorities of Attributes	244
73	Open Anonymizer - Algorithm Settings	245
74	Open Anonymizer - Parameter Summary	246
75	Open Anonymizer - Result of Anonymization	247

9 List of Tables

1	Overview of Biobank Initiatives	26
2	Differences between Business and Scientific Workflows	59
3	Mapping of First Provenance Challenge Queries to GATiB Provenance Queries	87
4	Comparison of Scientific Workflow Management Systems . . .	224
5	Patient Diagnoses	226
6	Patient Diagnoses without Identifiers	226
7	Patient Diagnoses Anonymized	226

10 References

- [1] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. *Provenance and Annotation of Data*, pages 118–132, 2006.
- [2] Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludaescher. Efficient Provenance Storage over Nested Data Collections. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 958–969. ACM, 2009.
- [3] Erik Anderson, Steven Callahan, Juliana Freire, David Koop, Emanuele Santos, Carlos Scheidegger, Claudio Silva, Nathan Smith, and Huy Vo. First Provenance Challenge: VisTrails. Online, accessed 2013-01-31, <http://twiki.ipaw.info/bin/view/Challenge/VisTrails>.
- [4] S. R. Armendolia, F. Estrella, and R. McClathey et al. Managing Pan-European Mammography Images and Data Using a Service Oriented Architecture. In *IDEAS-Workshop on Medical Information Systems*, pages 99–108, 2004.
- [5] M. Assel, B. Krammer, and A. Loehden. Management and Access of Biomedical Data in a Grid Environment. In *Proceedings of Cracow Grid Workshop 2006*, pages 263–270, 2007.
- [6] Martin Asslaber, Peter Abuja, and Konrad Stark. The Genome Austria Tissue Bank GATIB. *Pathobiology 2007*, 74:251–258, 2007.
- [7] Martin Asslaber and Kurt Zatloukal. Biobanks: transnational, European and global networks. *Brief Funct Genomic Proteomic*, 6:193–201, 2007.

- [8] Ann Finney Batiza. Bioinformatics, Genomics, and Proteomics, Getting the Big Picture. *Chelsea House Publications*, 2006.
- [9] David A. Bennett and Michael D. Waters. Applying Biomarker Research. *Environmental Health Perspectives*, 108:907–910, 2000.
- [10] Bioresource-Med. Genome Austria Tissue Bank . Online, accessed 2013-01-31, <http://www.bioresource-med.com>.
- [11] Applied Biosystems. Introduction to Gene Expression. Online, accessed 2013-01-31, http://www3.appliedbiosystems.com/cms/groups/mcb_support/documents/generaldocuments/cms_083618.pdf?ICID=mp-taqman-gex-0910.
- [12] Y. Bouillon, F. Wendling, and F. Bartolomei. Computer-Supported Collaborative Work (CSCW) in Biomedical Signal Visualization and Processing. *Trans. on Information Technology in Biomedicine*, 3(1), 1999.
- [13] Shawn Bowers, T M McPhillips, Sean Riddle, M Anand, and B. Kessler/pPOD: Scientific workflow and provenance support for assembling the tree of life. (1):70–77, 2008.
- [14] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuen-dorffer, Yang Zhao, and Haiyang Zheng. Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). (UCB/EECS-2007-7), Jan 2007.
- [15] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Claudio T. Silva, and Huy T. Vo. Managing the Evolution of Dataflows with Vistrails. In *In IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow)*, 2006.
- [16] Carina Christ. Process Editor for the Composition of Web Services. Bachelor Thesis, University of Vienna, 2009.

- [17] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genet.*, 25:25–29, 2000.
- [18] The World Wide Web Consortium. OWL Web Ontology Language Guide. Online, accessed 2013-01-31, <http://www.w3.org/TR/owl-guide/>.
- [19] Oracle Cooperation. Java Business Integration (JBI). Online, accessed 2013-01-31, <http://www.jcp.org/en/jsr/detail?id=208/>.
- [20] Francis Crick. Central Dogma of Molecular Biology. *Nature*, 227(5258), 1970.
- [21] Michael W. Davidson. Molecular Expressions Cell Biology. Online, accessed 2013-01-31, <http://micro.magnet.fsu.edu/cells/ribosomes/ribosomes.html>.
- [22] Ewa Deelman, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. *Lecture Notes in Computer Science*, 3165:11–20, 2004.
- [23] Ewa Deelman, Gaurang Mehta, Karan Vahi, Gideon Juve, Mats Rynge, Prasanth Thomas, and Jens Voeckler. Pegasus WMS: Leveraging Condor for Workflow Management. Online, accessed 2013-01-31, <http://www.cs.wisc.edu/condor/CondorWeek2010/condor-presentations/deelman-pegasus.pdf>.
- [24] Ewa Deelman, Gurmeet Singh, Mei hui Su, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

- [25] Johann Eder, Claus Dabringer, Michaela Schicho, and Konrad Stark. Information Systems for Federated Biobanks. In Abdelkader Hameurlain, Josef Kng, and Roland Wagner, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems I*, volume 5740 of *Lecture Notes in Computer Science*, pages 156–190. Springer Berlin Heidelberg, 2009.
- [26] G. Elgar and T. Vavouri. Tuning in to the signals: noncoding sequence conservation in vertebrate genomes. *Trends Genet.*, 24:344–352, 2008.
- [27] A. Arbona et al. A Service-Oriented Grid Infrastructure for Biomedical Data and Compute Services. *IEEE Trans. on Nanobioscience*, 6(2), 2007.
- [28] The National Center for Biotechnology Information. A Basic Introduction to the Science Underlying NCBI Resources. Online, accessed 2013-01-31, http://www.ncbi.nlm.nih.gov/About/primer/genetics_genome.html.
- [29] Organisation for Economics Cooperation and Development. Best Practice Guidelines for Biological Resource Centres. Online, accessed 2013-01-31, <http://www.oecd.org/dataoecd/7/13/38777417.pdf>.
- [30] Organization for the Advancement of Structured Information Standards. Web Services Business Process Execution Language Version 2.0. Online, accessed 2013-01-31, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>.
- [31] Apache Software Foundation. Apache Jena - A Java framework for building Semantic Web applications. Online, accessed 2013-01-31, <http://jena.apache.org/index.html>.

- [32] The Apache Software Foundation. Apache ODE. Online, accessed 2013-01-31, <http://ode.apache.org/>.
- [33] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5:237–246, July 2002.
- [34] Benjamin C. M. Fung, Ke Wang, and Philip S. Yu. Top-Down Specialization for Information and Privacy Preservation. In *ICDE*, pages 205–216, 2005.
- [35] Yolanda Gil, Jihie Kim, Varun Ratnakar, and Ewa Deelman. Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows. In *Proceedings of the 2006 Workshop on OWL: Experiences and Directions (OWL-ED)*, Athens, GA, 2006.
- [36] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, gkq429(Web Server issue):W682, May 2010.
- [37] Antoon Goderis. Example myExperiment Workflow : Gene Ontology Diagram. Online, accessed 2013-01-31, <http://www.myexperiment.org/workflows/317.html>.
- [38] Jelle J. Goeman, Sara A. van de Geer, Floor de Kort, and Hans C. van Houwelingen. A global test for groups of genes: testing association with a clinical outcome. *Bioinformatics*, 20(1):93–99, 2004.
- [39] Jeff Gulcher and Kari Stefansson. Population Genomics: Laying the Groundwork for Genetic Disease Modeling and Targeting. In *Clinical Chemistry and Laboratory Medicine*, volume 36, pages 523–527. De Gruyter, August 1998.

- [40] Daniel L. Hartl and Elizabeth W. Jones. Essential Genetics: A Genomic Perspective. *Jones and Bartlett Publishers, 4th edition*, 2005.
- [41] Agnar Helgason, Bryndis Yngvadottir, Birgir Hrafnkelsson, Jeffrey Gulcher, and Kari Stefansson. An Icelandic example of the impact of population structure on association studies. In *Nature Genetics*, volume 37, pages 90–95. Nature Publishing Group, December 2004.
- [42] David Hollingsworth. Workflow Management Coalition: The Workflow Reference Model. Online, accessed 2013-01-31, <http://www.wfmc.org/standards/docs/tc003v11.pdf>.
- [43] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H. Wang. The Manchester OWL syntax. In Ian Horrocks, Bijan Parsia, and Peter P. Schneider, editors, *OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11 2006*. <http://CEUR-WS.org/Vol-216/>, 2006.
- [44] Manuela Hummel, Reinhard Meister, and Ulrich Mansmann. Global-ANCOVA: Exploration and Assessment of Gene Group Effects. *Bioinformatics*, 24(1):78–85, 2007.
- [45] R.J. Kelleher, P.M. Flanagan, and R.D. Kornberg. A novel mediator between activator proteins and the RNA polymerase II transcription apparatus. *Cell* 61, pages 1209–1215, 1990.
- [46] Muin Khoury. The case for a global human genome epidemiology initiative. *Nature Genetics*, 36, 2004.
- [47] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun Ratnakar. Provenance trails in the Wings-Pegasus system. *Concurr. Comput. : Pract. Exper.*, 20:587–597, April 2008.
- [48] Jocelyn E. Krebs, Elliott S. Goldstein, and Stephen T. Kilpatrick. Lewin’s Genes X. *Jones and Bartlett Publishers*, 10, 2009.

- [49] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient Full-Domain K-Anonymity. In *SIGMOD Conference*, 2005.
- [50] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*. IEEE Computer Society, 2006.
- [51] P. Leo, C. Marinelli, G. Pappad, G. Scioscia, and L. Zanchetta L. BioWBI: an Integrated Tool for building and executing Bioinformatic Analysis Workflows. *Bioinformatics Italian Society Meeting (BITS 2004)*, 2004.
- [52] Ricki Lewis. Human Genetics: Concepts and Applications. *McGraw-Hill Science/Engineering/Math; 5 edition*, 2002.
- [53] W. Li, S. Krishnan, K. Mueller, and K. Ichikawa et al. Building Cyberinfrastructure for Bioinformatics Using Service Oriented Architecture. *Sixth IEEE Int. Symposium on Cluster Computing and the Grid Workshops*, 2006.
- [54] Bertram Ludaescher, 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. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.
- [55] Bertram Ludaescher, Mathias Weske, Timothy Mcphillips, and Shawn Bowers. Scientific Workflows: Business as Usual? *Proceedings of the 7th International Conference on Business Process Management*, pages 31–47, 2009.
- [56] L. Makris, I. Kamilatos, E. V. Kopsacheilis, and M. G. Strintzis. Teleworks: A CSCW Application for Remote Medical Diagnosis Sup-

- port and Teleconsultation. *Trans. on Information Technology in Biomedicine*, 2(2), 1998.
- [57] Erwin Mann. SECO - Eine smarte Ausführungskomponente für komponierte Web Services. Bachelor Thesis, University of Vienna, 2009.
 - [58] Jon F. Merz. On the Intersection of Privacy, Consent, Commerce and Genetics Research. In *B. M. Knoppers, ed., Population and Genetics: Legal and Socio-Ethical Perspectives*, 2004.
 - [59] A. Metspalu, F. Koehler, G. Laschinski, D. Ganten, and I. Roots. The Estonian Genome Project in the context of European genome research. In *Deutsche medizinische Wochenschrift (1946)*, volume 129 Suppl 1, April 2004.
 - [60] Sara Migliorini, Mauro Gambini, Marcello La Rosa, and Arthur H.M. ter Hofstede. Pattern-Based Evaluation of Scientific Workflow Management Systems. *QUT ePrints*, February 2011.
 - [61] Paolo Missier, Norman W. Paton, and Belhajja Khalid. Fine-grained and efficient lineage querying of collection-based workflow provenance. *Proceedings of the 13th International Conference on Extending Database Technology*, pages 299–310, 2010.
 - [62] Paolo Missier, Satya S. Sahoo, Jun Zhao, Amit Sheth, and Carole Goble. Janus: from Workflows to Semantic Provenance and Linked Open Data. *Procs. IPAW 2010*, 2010.
 - [63] Luc Moreau, Ben Clifford, Juliana Freire, Yolanda Gil, Paul Groth, Joe Futrelle, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The Open Provenance Model - Core Specification (v1.1). *Future Generation Computer Systems*, December 2009.

- [64] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The Open Provenance Model: An Overview. *IPAW*, pages 323–326, 2008.
- [65] Luc Moreau, Paul Groth, and Jun Zhao. Open Provenance Model Tutorial Session 6: Interoperability. Online, accessed 2013-01-31, <http://openprovenance.org/tutorial/slides/6-interoperability.pptx>.
- [66] Council of Europe. Recommendation Rec(2006)4 of the Committee of Ministers to member states on research on biological materials of human origin. Online, accessed 2013-01-31, <https://wcd.coe.int/ViewDoc.jsp?id=977859&Site=%20COE>.
- [67] Bethesda MD: National Institutes of Health U.S. Department of Health and Human Services. Stem Cell Basics. Online, accessed 2013-01-31, <http://stemcells.nih.gov/info/basics/Pages/Default.aspx>, 2011.
- [68] U.S. National Library of Medicine. Handbook, Help Me Understand Genetics. Online, accessed 2013-01-31, <http://ghr.nlm.nih.gov/handbook>.
- [69] Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1067–1100, August 2006.
- [70] William Ollier, Tim Sprosen, and Tim Peakman. UK Biobank: from concept to reality. In *Pharmacogenomics*, volume 6, pages 639–646, September 2005.

- [71] The World Health Organization. International Statistical Classification of Diseases and Related Health Problems, 10 th Revision. Online, accessed 2013-01-31, <http://www.who.int/classifications/apps/icd/icd10online2006>.
- [72] L. Sweeney P. Samarati. Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement Through Generalization and Suppression . *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1998.
- [73] Leena Peltonen. GenomEUtwin: A Strategy to Identify Genetic Influences on Health and Disease. In *Twin Research*, pages 354–360. Australian Academic Press, 2003.
- [74] Jonathan Pevsner. *Bioinformatics and functional genomics*, volume Second Edition. John Wiley and Sons, 2009.
- [75] Roland Pieler, Fatima Sanchez-Cabo, Hubert Hackl, Gerhard G. Thallinger, and Zlatko Trajanoski. ArrayNorm: comprehensive normalization and analysis of microarray data. *Bioinformatics*, 20(12):1971–1973, 2004.
- [76] Kepler Project. Getting Started with the Kepler Provenance Module. Online, accessed 2013-01-31, <https://code.kepler-project.org/code/kepler/trunk/modules/provenance/docs/provenance.pdf>.
- [77] U.S. DOE Human Genome Project. How Many Genes Are in the Human Genome. Online, accessed 2013-01-31, http://www.ornl.gov/sci/techresources/Human_Genome/faq/genenumber.shtml.
- [78] Kim D. Pruitt, Tatiana Tatusova, William Klimke, and Donna R. Maglott. NCBI Reference Sequences: current status, policy and new ini-

- tiatives. *Nucleic acids research*, 37(Database issue):D32–D36, January 2009.
- [79] C. Goble R. Stevens, J. Zhao. Using provenance to manage knowledge of in Silico experiments. *Brief Bioinform.*, 8(3):183–94, 2007.
 - [80] S. Ram and J. Liu. A Semiotics Framework for Analyzing Data Provenance Research. *Journal of computing Science and Engineering*, 2(3):221–248, 2008.
 - [81] Thomson Reuters. Biomarkers: An indispensable addition to the drug development toolkit. Online, accessed 2013-01-31, <http://clsdf.com/wp-content/uploads/2009/12/Thomson-Reuters-Examining-Biomarkers-20092.pdf>.
 - [82] Peter Rice, Ian Longden, and Alan Bleasby. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6):276–277, June 2000.
 - [83] P. Riegman, M. Morente, F. Betsou, P. Deblasio, and P. Geary. Biobanking for better healthcare. In *Molecular Oncology*, volume 2, pages 213–222, October 2008.
 - [84] Christine Rothmayr. Biobanks: Governance in Comparative Perspective. In *Governance*, volume 22, pages 759–761. Blackwell Publishing, 2009.
 - [85] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Guo. The Discovery Net System for High Throughput Bioinformatics. *Bioinformatics*, 19(1):225–231, July 2003.
 - [86] K. Sartipi, M. Yarmand, and D. Down. Mined-knowledge and Decision Support Services in Electronic Health. *Int. Workshop on Systems Development in SOA Environments*, 2007.

- [87] Jonas Schulte, Thorsten Hampel, Thomas Bopp, and Robert Hinn. Wasabi Framework – An Open Service Infrastructure for Collaborative Work. *SKG '07: Proceedings of the Third International Conference on Semantics, Knowledge and Grid*, pages 242–247, October 2007.
- [88] Jonas Schulte, Thorsten Hampel, Konrad Stark, Johann Eder, and Erich Schikuta. Towards the Next Generation of Service-Oriented Flexible Collaborative Systems - A Basic Framework Applied to Medical Research. *ICEIS (1)*, pages 232–239, 2008.
- [89] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [90] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance Techniques, Technical Report IUB-CS-TR618. *Science*, 47405(3):1–25, 2005.
- [91] Gordon K. Smyth, Yee H. Yang, and Terry Speed. Statistical Issues in cDNA Microarray Data Analysis Functional Genomics. *Methods in molecular biology (Clifton, N.J.)*, 224:111–136, March 2003.
- [92] L. H. Sobin. *TNM Classification of Malignant Tumours*. John Wiley and Sons, 2009.
- [93] Pothur Srinivas, Kramer Barnett, and Srivastava Sudhir. Trends in biomarker research for cancer detection. *The Lancet Oncology*, 2:698–704, 2001.
- [94] Konrad Stark, Johann Eder, and Kurt Zatloukal. Priority-based k-anonymity accomplished by weighted generalisation structures. In

- DaWaK 2006: Proc. of the 8th Int. Conference on Data Warehousing and Knowledge Discovery*, LNCS, Volume 4081. Springer Verlag, 2006.
- [95] Konrad Stark, Johann Eder, and Kurt Zatloukal. Achieving k-anonymity in DataMarts used for gene expressions exploitation. In *Journal of Integrative Bioinformatics*, 4(1):57, 2007.
 - [96] Konrad Stark, Christian Koncilia, Jonas Schulte, Erich Schikuta, and Johann Eder. Incorporating Data Provenance in a Medical CSCW System. In *Database and Expert Systems Applications - Proceedings of the 21st International Conference, DEXA 2010*, volume 6261 of *Lecture Notes in Computer Science*, pages 315–322, Bilbao, Spain, August 2010. Springer.
 - [97] Konrad Stark, Jonas Schulte, Thorsten Hampel, Erich Schikuta, Kurt Zatloukal, and Johann Eder. GATiB-CSCW, Medical Research Supported by a Service-Oriented Collaborative System. *CAiSE '08: Proceedings of the 20th international conference on Advanced Information Systems Engineering*, pages 148–162, 2008.
 - [98] Robert Tamarin. *Principles of Genetics*. W C B/McGraw-Hill, seventh edition, 1999.
 - [99] O. Quintana Trias. GenomEUtwin: A European Community Framework 5 Functional Genomics Special Initiative. In *Twin Research*, page 353. Australian Academic Press, 2003.
 - [100] S. Wallace, K. Bedard, A. Kent, and B. M. Knoppers. Governance Mechanisms and Population Biobanks: Building a Framework for Trust. In *GenEdit*, volume 6, 2008.

- [101] Ke Wang, Philip S. Yu, and Sourav Chakraborty. Bottom-Up Generalization: A Data Mining Solution to Privacy Protection. In *ICDM*, pages 249–256, 2004.
- [102] Mark Wilkinson, Heiko Schoof, Rebecca Ernst, and Dirk Haase. BioMOBY successfully integrates distributed heterogeneous bioinformatics Web Services. The PlaNet exemplar case. *Plant physiology*, 138(1):5–17, May 2005.
- [103] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. Utility-based anonymization using local recoding. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 785–790. ACM Press, 2006.

A Abstract

Medical research is characterized by a highly interdisciplinary environment, expert knowledge of various domains, dependency on biological material, distributed data repositories and computational resources. Biobanks have been established in recent years as biological resource centres, providing access to collections of specimens and associated biomedical data. In order to support complex medical research activities efficiently, biobanks require an adequate IT-infrastructure for enabling collaboration in research projects and executing research processes in a distributed environment. This thesis is concerned with the design and implementation of an IT system fulfilling manifold requirements for the medical research at a biobank. The developed IT system was realized as a multipurpose research platform for supporting collaborative activities and for enacting and monitoring scientific workflows, which are used for executing virtual experiments in order to proof or reject hypotheses. In order to guarantee the validity and traceability of research results, recording the provenance of data is of great importance. A fine-grained semantic data provenance model is presented in this thesis that is capable of answering research-relevant queries concerning the creation process of research results. The model allows to verify research results and is capable of identifying the impacts of changes in input data of scientific workflows. When using sensitive, patient-related data in research projects, data protection issues do have to be considered. Therefore, a novel anonymization algorithm based on the principle of k-anonymity is presented. The algorithm takes into account user-specific requirements and guarantees that an anonymized data set has a minimal information loss and is applicable to further processing in medical research projects.

B Zusammenfassung

Medizinische Forschung findet in einem stark interdisziplinären Umfeld statt und ist von Faktoren wie Expertenwissen aus verschiedensten Domänen, Abhängigkeit von biologischen Proben, verteilten Datenquellen und verteilten datenverarbeitenden Prozessoren geprägt. In den letzten Jahren haben sich Biobanken als forschungsunterstützende Zentren im medizinischen Bereich etabliert, in denen Wissenschaftlern Zugriff auf Sammlungen von biologischen Proben und den damit verbundenen medizinischen Daten ermöglicht wird. Um die Forschungsaktivitäten effizient durchführen zu können, bedarf es einer adäquaten IT Infrastruktur, die Kollaborationen in Forschungsprojekten erlaubt und Forschungsprozesse in einer verteilten Systemlandschaft umsetzen kann. Die vorliegende Doktorarbeit befasst sich mit dem Design und der Realisierung eines IT-Systems zur Unterstützung von medizinischen Forschungsaktivitäten im Kontext einer Biobank. Das entworfene IT-System ermöglicht sowohl computerunterstützte Gruppenarbeit in medizinischen Forschungsprojekten als auch die Ausführung und Überwachung sogenannter Scientific Workflows. Diese werden für die Durchführung von virtuellen Experimenten benötigt und dienen der Überprüfung von Forschungshypothesen. Für die Validierung und Nachvollziehbarkeit von Forschungsergebnissen ist eine exakte Protokollierung aller involvierten Daten und verarbeitenden Prozessoren essentiell. Die Rückverfolgbarkeit von generierten Daten, auch als Data Provenance bezeichnet, stellt einen bedeutenden Aspekt dieser Arbeit dar. Hierfür wurde ein Data Provenance Modell entwickelt, mit dessen Hilfe forschungsrelevante Abfragen bezüglich des Entstehungsprozesses von Forschungsergebnissen beantwortet werden können. Weiters kann über das Provenance Modell festgestellt werden, wenn Änderungen in Eingabedaten von Prozessen zu invaliden Forschungsergebnissen führen. Bei Verwendung von sensiblen, personenbezogenen Daten in medizinischen Forschungsprojekten müssen Datenschutzaspekte berücksichtigt werden. In dieser Arbeit wird ein innovativer Anonymisierungsalgorithmus vorgestellt, der, basierend

auf dem Konzept der k -Anonymität, eine zu veröffentlichende Menge an Datensätzen anonymisiert. Der Algorithmus bezieht Benutzeranforderungen sehr stark mit ein und gewährleistet einen minimalen Informationsverlust durch die Anonymisierung und dass die anonymisierten Daten brauchbar für die Weiterverarbeitung und Folgeanalysen sind.

C Curriculum Vitae

Konrad Stark was born and grew up in Carinthia, where he attended the commercial academy in Klagenfurt. From 1998 to 2004 he studied applied informatics at the University of Klagenfurt with focus on computer linguistics, data warehouses, data mining and workflow management systems. In his diploma thesis “High Performance Caching in a Workflow Management System”, he developed a distributed caching architecture for the workflow management system @enterprise. Parallel to his study, he worked as a software engineer in several companies and developed applications with Java Swing, web technologies, Java backends and Oracle databases. From 2004 to 2006 he worked as a research assistant at the Medical University of Graz, where he was involved in research projects of the biobank initiative Genome Austria Tissue Bank (GATiB). His main tasks were the integration of medical records from various data sources, development of a database for managing the tissue sample collection of the biobank, the statistical analysis of gene expression profiles and the anonymization of patient records. In 2006, he started to work as a research assistant at the Institute for Knowledge and Business Engineering at the University of Vienna. He continued his research activities for the GATiB initiative within the Austrian research program GEN-AU. His main research focuses were on the development of a scientific workflow management system for automating medical research processes, capturing and querying of provenance data and on designing and implementing efficient algorithms for the anonymization of sensitive patient data. From 2009 to 2013 he worked as a software engineer and team leader at the Austrian Department of Defense. His main responsibilities were the system operation, and the optimization and further development of the process and document management system ELAK.

D Installation and Configuration of Open Anonymizer

Open Anonymizer is a web application programmed in Java (source level 1.6). The software is available as a prebuilt stable release and as a source code release. Both releases are available from the open source platform sourceforge.net. The url of the project homepage is <https://sourceforge.net/projects/openanonymizer>. The source code and binaries may be downloaded using a subversion client.

The release is composed of three folders, dist, web and src. While the latter two folders contain source code for developers, the dist folder contains the binary release open_anonymizer.war. Alternatively, the war file is available at the project home page by simply clicking on the green download button.

D.1 Stable Binary Release

The source files of a stable and tested version have been compiled and packed into a WAR file, which may be deployed on an Apache Tomcat application server. We encourage to use the 6.x version of Tomcat which is available under <http://tomcat.apache.org/download-60.cgi>. Please follow the following steps in order to set up a running test environment.

1. Download the core version of the binary distributions and extract the content of the zip archive into a separate folder.
2. As the anonymization process may be memory extensive, we recommend to change the memory parameter of the Java virtual machine. Open the Tomcat start-up file in a text editor.

On windows systems open catalina.bat, and add the following line at the beginning of the file:

```
set JAVA_OPTS = %JAVA_OPTS% -Xmx512m
```

On Linux systems choose catalina.sh, and add the following line:

```
JAVA_OPTS = " -Xmx512m"
```

If the test server has enough main memory (2 GB or more), the memory limit for the virtual machine may be set to `-Xmx1024m`.

3. Open the file `tomcat-users.xml` which is located in the `/conf` folder of the tomcat directory. Add the following roles in the configuration file:

```
<role rolename="manager"/>
<role rolename="admin"/>
```

Further, add a new user and assign the two new roles to him:

```
<user username="**your_user**"
      password="**your_password**"
      roles="tomcat,manager,admin"/>
```

4. Start the application server by executing the `startup.bat` (respectively, `startup.sh` in Linux) file.
5. Open a browser and enter the following url: `http://localhost:8080/manager/html`. Enter the user name and password specified in `tomcat-users.xml`. Typically, the default port for tomcat is 8080. Though, it can be configured by the port parameter in `/conf/server.xml`:

```
<Connector port="8080" protocol="HTTP/1.1">
```

6. Scroll down to the section “WAR file to deploy” and select the downloaded `open_anonymizer.war`. If the deployment succeeds, the web application may be accessed at the following path:
`http://localhost:8080/open_anonymizer`.

D.2 Source-Code Release

The source code of Open_Anonymizer is bundled in the web and src folders that may be downloaded from the above-mentioned svn repository. We developed the web application using the NetBeans 6.1 IDE. In the following, we show how a new NetBeans project may be set up on our source code.

1. As the svn client extracted both source code and binaries into the same folder, we encourage to delete or remove the dist folder. A new dist folder will be created by NetBeans. We propose to install the NetBeans plugins "Tomcat" and "Web Applications" in order to have a working development and test environment.
2. In NetBeans start the project wizard by clicking "File" → "New Project" → "Web Application with Existing Sources". Set the location to the path of the openanonymizer folder, and set the server to Tomcat 6.0. Please change the context path to open_anonymizer.
3. All settings in the next step should be automatically detected.

The path of the web page folder is "../openanonymizer/web", and the WEB-INF content is located in "../openanonymizer/web/WEB-INF".

The libraries folder is located in "../openanonymizer/web/WEB-INF/lib", and the source package folders are contained in "../openanonymizer/src".

4. Finish the wizard and let NetBeans initialize the project. Right-click on the openanonymizer project and choose "Clean and Build" to generate a binary war file.
5. Finally, right-click on the project and select "Undeploy and Deploy". Thus, the integrated application server is started with the Open_Anonymizer

web application. The anonymization wizard may be accessed by entering the following url into a browser `http://localhost:8080/open_anonymizer`.