MASTERARBEIT

Provenance in WS-VLAM

Comparing Open Provenance Model with History Tracing XML-based Provenance Framework

vorgelegt an der
Fachhochschule Aachen, Campus Jülich

Fachbereich: Medizintechnik und Technomathematik

Studiengang: Technomathematik, M. Sc.

Diese Arbeit wurde betreut von:

Prof. Dr. rer. nat. Volker Sander
Prof. Dr. Adam S. Z. Belloum

Vorgelegt von:

Torsten Matzerath
Lyatenstr. 12
52382 Niederzier

Matrikel - Nr.: 981764

Jülich, 18. April 2011

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Ort, Datum	Unterschrift

Table of Contents

List of Figures

List of Tables

1 Introduction

In the last few decades, the use of workflows to answer diverse scientific problems in disciplines like bioinformatics, physics or medicine, strongly increased. The advance in the scientific areas as well as the introduction of the grid infrastructure to enhance the computing power caused an increasing complexity of workflows. These two reasons made the implementation of the scientific questions so complex that engineers and scientists could not create the programs themselves anymore, but computer scientists were needed.
To be more efficient and to give engineers and scientists the chance to create the workflows themselves, so-called *Workflow Management Systems* (WfMS) were developed. These systems help to create workflow tasks and deal with the workload in the distributed environments automatically. Moreover, scientists can use the WfMS to define the applications, analyse and execute the workflows, and collect the results. Another advantage is that the modules of one workflow can be reused for other workflows. Summing up, one can say that the WfMS offer an intuitive way to compose scientific workflows, especially for non-computer-scientists. [2]
However, the simple execution of workflows and the collection of results is not enough for modern workflows. As data volumes expand and the computer systems become more distributed, the interest of the origin of the data and how it was generated is growing. Moreover, information like who or what created the data, the history of the data, and the calculating system are critical to a better understanding, reusability and reproducibility of the results. [12]
This collection of data, which shows the origin of results, is called provenance. Provenance is not the original data. It is the way to trace back all events that lead to the creation of a certain data up to the initial data set. The member of the Open Provenance Model (OPM) challenge, a group of computer scientists from companies and universities around the world, defined it as:

> *"Provenance is a critical concept in scientific workflows, since it allows scientists to understand the origin of their results, to repeat their experiments, and to validate the processes that were used to derive data products."* [17]

As a consequence of the recognized importance of collecting provenance data, many provenance models and techniques were developed. All systems have advantages and disadvantages. The goal of this master thesis is to compare the provided provenance data of two models using the workflow management system WS-VLAM. On the one hand the Open

Provenance Model and on the other hand the history tracing XML-based provenance framework for workflows will be analysed.

The Open Provenance Model is a specification with the goal to define a provenance model in a precise technology-agnostic manner. By dint of the specification it should be possible to exchange provenance information between different systems and allow developers to create programs, which collect and handle provenance data. The aim of the specification is to be independent from any workflow management system and therefore no implementation exists. Hence, the developers of every WfMS have to realize their own implementation. For this thesis the implementation of the Open Provenance Model for the WfMS WS-VLAM by the University of Amsterdam is used. [13]

The history tracing XML-based provenance framework for workflows is a ready-implemented web service based program, which can be used for any WfMS with minor changes in the source code of the tracing system. The provenance data is stored in XML files in a layered structure. The re-implementations were accomplished during this thesis as well as the development of a Provenance Query Interface to figure the provenance data.

Both systems have the same goal – to collect provenance data, but in different ways. To be able to better compare them, both systems will be executed on the WfMS WS-VLAM using the workflow BLAST. This practice ensures that the input for the two systems is the same and the different outputs of the systems can only be caused by the different concepts.

The goal of the comparisons is to find out which system is responsible for providing which provenance data. It will also be analysed, which of the systems can be extended more easily to collect the needed provenance data and which data can be analysed more easily. Additionally, the XML structure and security questions will be discussed and the more efficient and secure system will be pointed out. The study of the provided provenance data will be done in close contact to bioinformaticians who use the workflow and need the provenance data to understand errors and the output data. This is necessary to develop a provenance system, which is adapted to the needs of the users and will be accepted by them. The comparison of the two systems helps to find out the advantages and disadvantages of OPM and history tracing and to develop a lineage system that combines the goals of both systems.

To be able to analyse the differences between the two provenance systems some implementations are necessary, which will be discussed in the thesis. The thesis begins with a chapter describing the most important background topics. This section will be followed by a chapter that describes the term of provenance in detail with requirements that can be found in literature and that are expressed by users. It follows a chapter that explains the history tracing sys-

tem and which adaptations are necessary to combine it with WS-VLAM. Then, the Provenance Query Interface that visualizes the provenance data is described. The next chapter discusses the Open Provenance Model specification and the XML schema. After that, the comparison of the two provenance models is pointed out. The thesis closes with a look on future work that can be done in the area of history tracing system and the adaptation to WS-VLAM.
Besides the comparison of the two provenance systems the main focus of the thesis is to find out if the history tracing system can be used to save provenance data from WS-VLAM. It will be pointed out if all necessary provenance data can be collected and if the system is configurable enough to fulfil the users' needs. As a consequence it will be considered if the history tracing system is an alternative to the OPM implementation for WS-VLAM and if both systems can be used as standard provenance systems.

2 Background

This section introduces the most important terms used in this thesis. First, definitions of workflow and workflow management system will be given. Descriptions of the workflow management system WS-VLAM and the used BLAST workflow follow.

2.1 Workflow and Workflow Management System

The term *workflow* is understood as a sequence of single activities, which are often dependent on each other or parallel.
First, workflows were adopted for the description of business processes. But nowadays they are also used to describe scientific experiments or problems like the simulation, analysis or execution of experiments. The workflows help scientists to describe complex scientific challenges from the analysis, via the development to the collection of results, without using programming languages. The piece of software, with which the user can design a workflow, is called *Workflow Management System* (WfMS). Workflow management systems hide the complexity of the workflow execution on complex systems such as grid infrastructures.
A workflow management system contains several components and consists at least of:

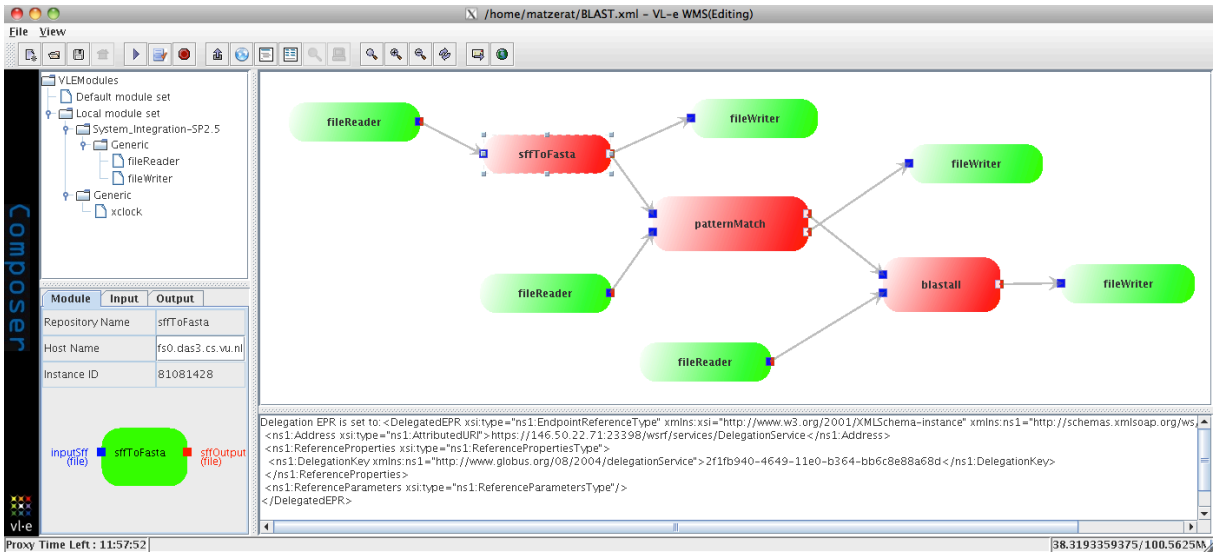
- **Workflow description model:** It defines the structure of the activities and their dependencies in the workflow. It describes the order of the modules and the input and output parameters or files.
- **Workflow engine:** The engine executes the defined workflow on any computing resources from desktop computer to grid resources.
- **User interface:** Provides tools to describe, execute and control the workflow and its results. [2]

As already mentioned, the workflow management system used in this thesis is WS-VLAM (Virtual Laboratory Abstract Machine). WS-VLAM was developed at the University of Amsterdam and is the second generation of the VLAM software. VLAM was developed between 2002 and 2004. VLAM was re-implemented to WS-VLAM to follow the new standards of Web Services and SOA. How the workflow management system works will be described in the following section.

2.2 WS-VLAM

The workflow management system WS-VLAM consists of three components. First, the graphical user interface, also called the composer, second the *Run Time System Manager* (RTSM) service and third the run-time part, which consists of the libraries for executing the modules. The user just gets in contact with the graphical user interface, which is based on the JGraph library and allows creating workflows per drag and drop. The modules describe the tasks and are displayed as vertices on the GUI. The connections between the vertices are defined by the input and output ports of the modules.

Figure 1: WS-VLAM GUI with BLAST workflow



As a result, the composer creates a file in XML format containing the workflow description. Then, the user can create his credentials to get the authority running the modules on the grid infrastructure. When the credentials were delegated correctly the user can press the *play* button and run the workflow. [8] The following list shows the actions, which run in to execute the workflow.

1. Workflow engine starts as a *Run Time System Manager* service:

   The content of the XML file describing the workflow is transmitted to the RTSM. This is necessary to set the parameters and connections at a later state.

2. RTSM creates an instance:

   The instance gets a unique ID and a Java object, which can receive events from runtime. If events occur during execution they are passed to a queue with the unique RTSM ID and the module ID, which is assigned at composing time and is unique in scope of the workflow template. The RTSM-ID is passed back to the client as an end point reference (EPR). The EPR is a XML element to specify a communication end

point for messages to a web service. It can contain the address, security tokens and other metadata needed for the interaction with the instance.

3. Client sends run command to the RTSM instance:
   The instance is using the end point reference and a reference to the credential and RTSM puts the job description to the job queue with assigned credentials.
4. Modules get instantiated on the remote resource:
   The instantiation is done with the *Resource Specification Language* (RSL), a language used to describe how to run a concrete task on a remote resource. Among other things RSL contains the host name of the resource, the path to the executable and the working directory.
5. Job submitter creates a *Global Access to Secondary Storage* (GASS) Server:
   It retrieves stdout and stderr for the modules of the experiment. GASS is used to propagate stdin and stdout and the server uses secure http for authentication and data transfer. [14]
6. Job description is taken out of the queue and passed to *Globus Resource Allocation Manager* (GRAM) system:
   GRAM is a component of the Globus Toolkit, which is an open source toolkit for building computing grids. GRAM locates, submits, monitors and cancels jobs on grid computing resources. Then, the task is instantiated on the explicitly specified remote resource. [15]
7. Modules get started:
   The modules contact the RTSM and register their selves and also create an association between the module representation and the running program. If this is done an *MODULE_REGISTERED* event is generated. GRAM can also generate events, which are created by module execution and pass them to RTSM.
8. When all modules are registered:
   RTSM connects the modules according to the XML file it received before, sets all parameters and sends the run command to each module. For each operation an event is generated and passed to the RTSM.
9. The modules receive the run command:
   The execution is started and the state is changed to *RUNNING* and RTSM waits until all modules finished execution.

10. When all modules are finished:

    If no errors occurred in GRAM the experiment is considered as successfully finished. Thus, if GRAM reports an error the experiment is considered to be failed.

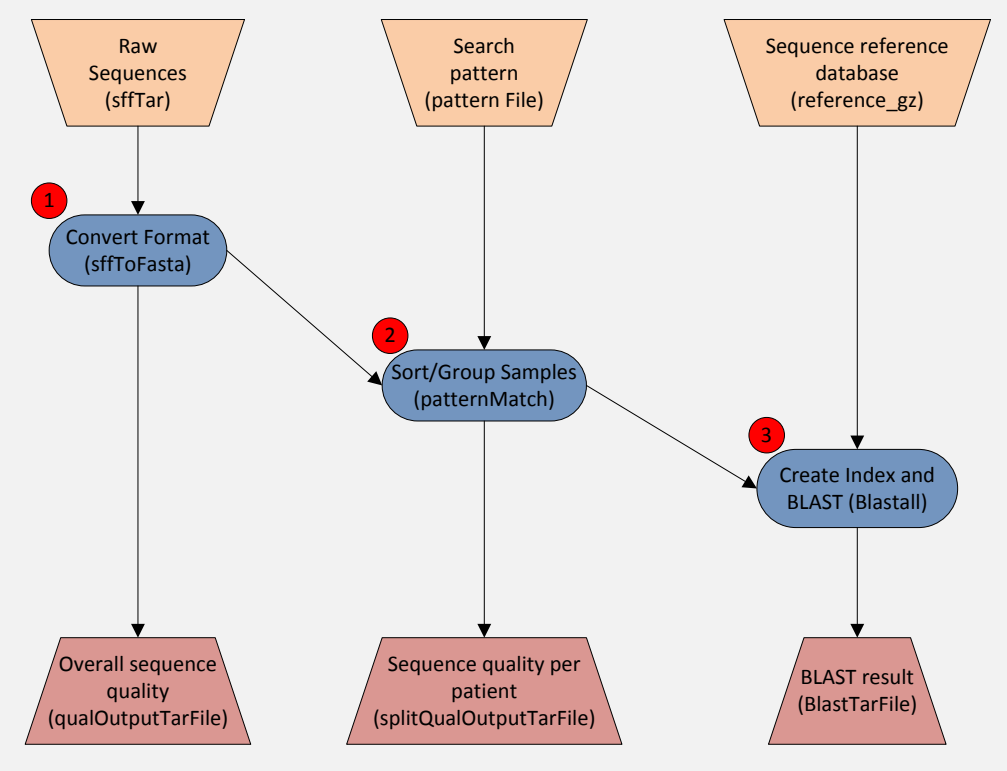
All events will be displayed in a window for every module so that the user can view them during execution and see if the workflow runs properly. WS-VLAM also allows to run the same workflow many times parallel with other input values or other parameters. Those parameters are read from lists or different input and output files can be defined when the user defines the workflow. A parallel run like that is called farmed run and WS-VLAM gives every single workflow run of a farmed run the same farming ID so that all instances can be assigned to the farmed run.

2.3 The BLAST workflow

The BLAST workflow conducts similarity search in DNA sequences. If DNA sequences are read with DNA sequencers, one of the first questions is which kind of DNA sequence was recorded. To answer this question, a scientist usually compares the sequence with the content of a public database like GenBank. Programs written for the alignment search in DNA sequences finally accomplish the comparison. The fastest of these programs is the BLAST (Basic Local Alignment Search Tool, http://blast.ncbi.nlm.nih.gov/Blast.cgi) program, also being used in the BLAST workflow. The algorithm is based on mathematical methods, which have been developed by Stephen Altschul and Samuel Karlin in the 1990s. BLAST points out a comparison of the analysed sequences and calculates the probability of analogies being real matches or random background hits. [21]

Since modern DNA sequencers analyse giga- or even terabytes in a short time it is not possible to analyse the data on a local computer, but the grid has to be used. As other steps have to be taken before the BLAST task can be run, a workflow should be utilised to automate the activity.

Figure 2: Generic DNA sequencing workflow ([9], p.4)



As can be seen in Figure 2, the first task reads the experimental data. It converts the data from the so-called sff-format to the fasta-format. This conversion is necessary because the BLAST task can only handle data in the fasta-format.

The second task also fulfils some preparing. Since the researcher can analyse more than one sample or patient in one sequence experiment, it is necessary to add a sequence label (also called MID or barcode) to the DNA fragments making sure that the data is assigned to the right patient. These labels are read by the *patternMatch* task as an input file, added to the sequences and sorted in different files for each expression.

The BLAST task gets the arranged experimental data and the data from the database as input to compare them. The output of the workflow is a text file, which contains all found matches. This file will then be given to the biomedical researchers who analyse the data.

The researchers of the Academisch Medisch Centrum (AMC) of the University of Amsterdam use the workflow for their research for which the MOTEUR workflow management system is used. MOTEUR is a workflow management system that was developed by the Modalis Team and can be used to create and execute workflows on the e-BioScience Infrastructure. The creation and running of a workflow involves the following steps. First, one describes the workflow in Simple Conceptual Unified Flow Language (SCUFL) specifying all workflow inputs, outputs, the programs to be executed and how they are connected. This can be done by a graphical user interface running as a plugin in the VBrowser. Second, the user starts the work-

flow with that plugin and MOTEUR then executes the workflow on the grid using the gLite middleware. The execution progress is visualized on HTML pages. [16]

To run the workflow in WS-VLAM some changes, described in the following section. Since the modules already exist it is not necessary to re-implement them but one has to create legacy applications are necessary (More details on this are available at http://staff.science.uva.nl/~gvlam/wsvlam/Documentation/WS-VLAM-tutorial-II.ppt). Those legacy applications are used in the WS-VLAM graphical user interface to create the workflow with all file readers, file writers and parameters. The resulting workflow looks as depicted in Figure 3.

Figure 3: BLAST workflow generated with WS-VLAM

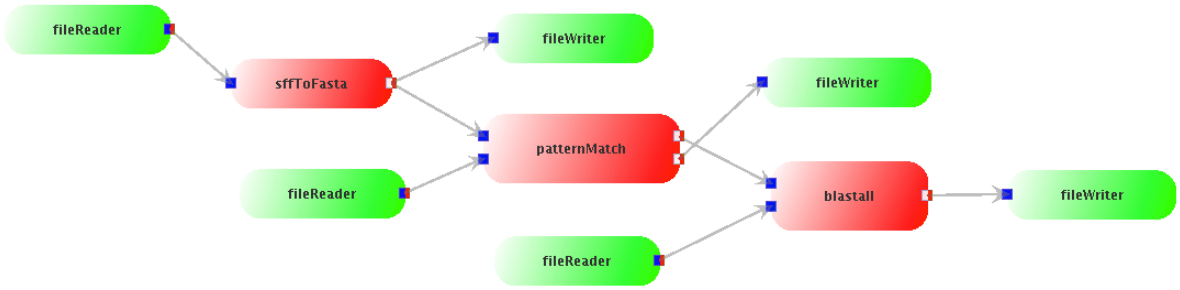


Figure 3 illustrates the workflow used for the feasibility study of the history tracing provenance system, which poses two challenges in the thesis. On the one hand, the use of the workflow helps the bioinformaticians to envisage the needs and chances of provenance in their daily work and they will formulate requirements for the provenance system. On the other hand, the BLAST workflow is the first scientific application, which is used in context with the history tracing system. This helps to find out possible shortcomings of the original system and necessary changes to the system. [9]

3 Provenance

This chapter points out the importance of provenance for the analysis of scientific workflows. Therefore, the term of provenance will be defined first. In the second section use cases and technical requirements will be explained including an analysis of which data has to be collected. It follows a description of reproducibility and the data needed to answer reproducibility before the suggestions of BLAST workflow users will be described. The chapter closes with a discussion highlighting, which data can be collected in a provenance system used in the WS-VLAM workflow management system.

3.1 The term of provenance

The recent launch of workflow management systems as well as the growing power of computer systems and grid computing systems around the world caused a huge increase in the use of large-scale applications. Those applications help to accomplish complex scientific experiments and are too complicated to find errors or analyse the results without the assistance of metadata. Metadata also describe the applied and generated data of the programs, which are often in the size of giga- or even terabytes. No scientist can handle or reuse them without the utility of additional information. [3] [5]
Provenance (also called lineage or audit trail) is one kind of metadata, which differs from other types because it is based on relationships among data and processes. With the aid of the recorded relationships, it is possible to pertain to the derivation history of a data product starting from its original sources. Hence, analysing the provenance data helps to understand which processes and/or persons are involved and which input data and parameters are used to generate them. [4]
Groth et al. ([1], p.2) define provenance as follows:

> *"The provenance of a piece of data is the process that led to that piece of data."*

A demonstrative example, for which the storage of provenance data is sensible, is the following experiment of a bioinformatician: A bioinformatician downloads sequence data of a human chromosome from a database and performs an experiment analysing the data. The bioinformatician later performs the same experiment on data of the same chromosome, which is

downloaded from the same database. He compares the two experiment results and notices a difference. Therefore, he determines whether the difference was caused by a change in the experimental process or configuration, or by differing chromosome data (or both). [6]

This example highlights some questions, which should be answered by the collected provenance data. A user could query:

- Who created this data product and when?
- When was the data product modified and by whom?
- What was the process used to create the data product?
- Were two data products derived from the same raw data?

([7], p. 1)

Besides these questions, provenance data can capture a more complex process analysis, which will be analysed in the chapter "Provenance in Literature".

The previously mentioned questions are only one aspect of workflow analysis with provenance. The other aspect is the reproducibility - a detailed record of input values allowing other scientists to validate and re-run the experiment. Reproducibility is one of the forces of provenance since one can monitor which input data and processes were used to create the results. A more precise description of reproducibility and the necessary provenance data to permit reruns is given in the chapter "Reproducibility".

Besides the collection of data, formatting is an essential aspect for the usability of provenance data. The collected lineage data is normally stored in XML format or in databases. These formats are not easy to be read for non-computer scientists and have to be displayed in a graphical user interface. Graphs are used as a technique of visualization by almost all provenance systems as they have the advantage of being able to show the dependencies in an intuitional way. An additional aspect is that workflows in workflow management systems are also visualized by graphs and users can easily understand the provenance graphs since both diagrams look similarly.

Another important facet that should be provided by provenance systems is the transfer of outputs to other WfMS, which are able to read the collected data. On the one hand, it is desirable since scientists want to compare their own provenance files and origin of the data with the ones of their colleagues. On the other hand, the transfer facilitates the reproducibility on other systems to verify the results.

3.2 Provenance in Literature

In this chapter different use cases and technical requirements will be analysed, which are described in *The requirements of using provenance in e-Science experiments* by Miles et al. (2006) from the University of Southampton. First of all, the methodology will be described with which the authors succeeded in the paper, which will be followed by a numeration of the use cases and technical requirements with listings of needed data. The paragraph closes with the discussion what the use cases stand for and which are the most important. [6]

3.2.1 Methodology

From the computer scientists' point of view it is not easy to find out which data has to be collected by a provenance system. Indeed, they create the concepts and implement the data lineage systems, but they do not have to analyse the workflow results or re-run the workflows. Thus, they have to talk to bioinformaticians, physicists, chemists or biologists who do the analysis with their year-long experience to finally identify which information is relevant and has to be stored.

One analysis of requirements for provenance architecture is described in the paper *The requirements of using provenance in e-Science experiments*. The paper was written by Miles, Groth, Branco and Moreau who have delved into provenance for several years and belong to the authors of the Open Provenance Model. The authors also faced the problem that they needed some feedback of workflow users, telling them which audit trail data they require. The computer scientists wanted to implement a user friendly and accepted architecture to collect and show provenance data with this information. For that, they asked different scientists of diverse disciplines to accomplish their workflows and write down the desired lineage data.

With the help of this method the authors formulated use cases and technical requirements their architecture should achieve. The requirements are formulated in the way "*PASOA should provide…*" PASOA is the architecture the authors wanted to implement. However, the technical requirements are good for all provenance architectures and a cornerstone of the analysis for the history-tracing XML-based provenance framework.

3.2.2 Use cases and technical requirments

This section describes the use cases and technical requirement of the mentioned paper. The use cases are classified into nine topics: Types of Provenance, Structure and Identity of Data, Metadata and Context, Sessions, Query, Processing and Visualisation, Non-Repudiation, Re-

Using Experimental Process and Aggregated Service Information. In every of these topics the use cases are briefly summarized and a list of necessary information to be collected of a provenance system are listed. At the end of every part a summary in form of a technical requirement is given, which will also be analysed. All eighteen use cases described in the original paper are inserted in the "Appendix" in the section "Use cases described in literature".

TYPES OF PROVENANCE

The three different views of provenance listed below are necessary if a user runs an experiment twice and wants to know why the results differ or a bioinformatician wants to examine the source and intermediate data used to produce the result data.

The three different views are:

1. Interaction: A record of the interaction between services that took place, including the data that was passed between them.
2. Actor State: Extra information about a service participating in the experiment at the time that the experiment was run.
3. Relationship: Information on how one data item in a process relates to another.

The required data for the analysis of the use cases in this section is:

- Interaction between services
- Data passed between services
- State or version of any service used in the experiment
- State or version of any input data
- State or version of the workflow management system
- Input data, intermediate and end results
- Relationships between data
- Relationships between data and processes

TECHNICAL REQUIREMENT 1:

The architecture should provide for the recording and querying of interactions, actor states and relationships.

This means that the provenance architecture should contain a mechanism to store and visualize the data in a way that different users with different intentions can adapt the system so that they see the data, which is interesting for them. This should be in a way that the graphical user

interface is dynamically adaptable with minor changes. Especially the three mentioned views are necessary since users asked most often for them.

STRUCTURE AND IDENTITY OF DATA

Identification of data is necessary if users need information of links between input data and output data or used data in workflows.

The required data for the analysis of the use cases in this section is:

- Add identifier to data to identify and follow them
- Identifier has to be usable in provenance queries
- One must be able to reference element by id
- A message system if input data sets are changed
- Link from workflow to input data
- State or version of input data

TECHNICAL REQUIREMENT 2:

The architecture should provide for association of identifiers with data, so that it can be referred to in queries and by data sources linking experiments together.

TECHNICALREQUIREMENT 3:

The architecture should provide for referencing of individual data elements contained in message bodies recorded in the process documentation.

These two technical requirements mean that there should exist a mechanism to refer end results to input data. Normally logging files have to be searched for the linking and this can be very complex depending on the workflow. The technical requirement says that the information should be stored in a provenance system and a consistent mechanism should be implemented.

METADATA AND CONTEXT

Metadata is important for many experiments. Thus, it can be used if a user wants to check the possibility whether an experiment is adequate for the used input data or if the planned tasks of an experiment are matching the executed tasks.

The required data for the analysis of the use cases in this section is:

- Link from process documentation to metadata
- Different views on data
- Input data and parameters
- Store metadata in separate store
- Output data
- Used modules and connections between them
- Task describing data
- Documentation of modules
- Link from provenance data to metadata

TECHNICAL REQUIREMENT 4:
The architecture should provide for process documentation and associated metadata in different stores to being integrated in providing the answer to a query.

The technical requirement says that the lineage system should contain a system that collects metadata in provenance files from input data or modules. This can be necessary to get background information about the content or the algorithms of data and modules. They can be important in a provenance file since an analysis of the outputs is easier with the metadata.

SESSIONS

Different sessions are useful to check if different paths of a workflow are calculating the same results and which of them are more efficient.

The required data for the analysis of the use cases in this section is:

- Should be able to compare different workflows
- Data passed between services
- Version of Module

TECHNICAL REQUIREMENT 5:
The architecture should provide a mechanism, which groups recorded process documentation into a session, and should allow comparison between sessions.

This technical requirement says that it is very important for provenance architectures to split and compare parallel tasks in a workflow. This is necessary to compare parallel tasks and see if the parallel tasks work correctly and which one is more efficient.

QUERY

Searching over several provenance files is a necessary technique to answer questions like whether input data was used in other experiments and changed by them or to find all workflows generating special output data.

The required data for the analysis of the use cases in this section is:

- Input data, intermediate and end results
- Full contents of the records of several experiments
- All services which were used
- Search mechanism over different provenance files to find queried values

TECHNICAL REQUIREMENT 6:

The architecture should provide for the process documentation to be returned in the groups specified at the time of recording or searched through on the basis of contextual criteria.

The purpose of this technical requirement is to provide searching over several provenance files for used input or output data. This can be important if the user wants to find out the usage of special files in all workflows. A mechanism for this kind of searching has to be implemented to the user interface of the provenance system.

PROCESSING AND VISUALISATION

An example in this section is a bioinformatician B performing an experiment. B publishes the results and makes a record of the experiment details available for the interest of B's peers.

The required data for the analysis of the use cases in this section is:

- Input data, intermediate and end results
- All services which were used
- All settings of a user/process
- All actors of tasks
- Create different views to visualize the information which is interesting for each query

- Relationships between data and processes
- State or version of any service used in the experiment
- State or version of any input data
- State or version of the workflow management system

TECHNICAL REQUIREMENT 7:
The architecture should provide a framework for introducing processing of process documentation of all three types discussed in „TYPES OF PROVENANCE“ (interactions, actor states and relationships), using various methods, then visualising the results of that processing.

The seventh technical requirement is similar to the first one. It also requires different views of the provenance files and it should be able to visualize them in a Provenance Query Interface.

NON-REPUDIATION

If workflow results have to be checked by a patent reviewer they want to know if the used input data is also free for commercial use or which of the results were generated earlier.

The required data for the analysis of the use cases in this section is:

- State or version of database and input data
- Processes that were used
- Input data, intermediate and end results
- All services which were used
- Relationships between data and processes
- State or version of any service used in the experiment
- Times of all events that happened
- All data have to be stored in an unmodifiable way

TECHNICAL REQUIREMENT 8:
The architecture should provide a mechanism for recording adequate process documentation, in an unmodifiable way, to make results non-repudiable.

The technical requirement mentions non-repudiation in provenance files. Signatures in XML files can for example do this. This is an important aspect since provenance is often used for legal affairs like contracts, in the medical area or for waste disposal.

RE-USING EXPERIMENTAL PROCESS

Biologists or bioinformaticians running workflows often use databases for their analysis. If the data from the databases is updated they want to re-run the workflow with the new data.

The required data for the analysis of the use cases in this section is:

- Input data, intermediate and end results
- State or version of any service used in the experiment
- Times of all events that happened
- A message system if input data sets are changed
- Link from workflow to input data

TECHNICAL REQUIREMENT 9:
The architecture should provide for the use of process documentation to re-enact an experiment using the same process but new inputs, and to reproduce an experiment with the same process and inputs.

This technical requirement means that re-running of experiments should be possible. Therefore, the same or other input parameters should be taken, which signifies that it must be possible to connect the Provenance Query Interface together with the workflow management system to simplify the re-running process.

AGGREGATED SERVICE INFORMATION

If users want to calculate statistical values of different workflow runs a search algorithm over different provenance files is necessary.

The required data for the analysis of the use cases in this section is:

- Calculate extra information like duration of process from saved information
- Query over different workflows

TECHNICAL REQUIREMENT 10:
The architecture should provide for querying, over process documentation of multiple experiments, about the aggregate behaviour and properties of services.

This technical requirement needs searching over several provenance files, too. In contrast to technical requirement six, especially information about administrative questions are handled.

3.2.3 Analysis of use cases and technical requirements

This chapter summarizes the awareness achieved by the use cases and the resulting technical requirements for the provenance system to be developed. Since it is not possible to develop all technical requirements in this thesis, the following analysis will help to identify the most frequently used ones.

The following table gives an overview about the technical requirements that are necessary to answer the different use cases. The occurrence is a degree indicating the priority to implement the technical requirements in the Provenance Query Interface. In any case, it has to be discussed if the data can be collected and which the users would prioritize.

Table 1: Use of Technical Requirements for Use Cases

Technical Requirement	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18
TR1: Types of provenance	×	×	×		×		×				×	×	×					
TR2&3: Structure and identity of data	×	×	×	×						×				×	×	×	×	
TR4: Metadata and Context	×			×	×	×	×	×	×		×			×	×	×	×	
TR5: Sessions								×										
TR6: Query				×					×	×						×		×
TR7: Processing and Visualisation	×	×	×		×		×	×			×	×	×					
TR8: Non-Repudiation										×			×	×	×			
TR9: Reusing experimental process	×			×		×							×			×	×	
TR10: Aggregated service information									×	×								×

As shown in the table, the different technical requirements are used unequally often to answer the use cases. The technical requirements one and seven, which provide different views of the provenance data, are used very often. Collecting and showing the documentation of processes and data in different views is very important because users with other interests need a dissimilar look on the lineage data. While it is more important for a scientist analysing the results to have a view on the generation of data with input data and parameters, a scientist debugging the process of the workflow has more interest in data that is showing the dependencies between the modules and their influence on the results. To fulfil these two requirements, the part of the provenance system collecting the data is not able to make a change between the different views. It always has to collect the provenance data important for all users since it does not know which kind of user is considering the data. Indeed, the Provenance Query Interface to be implemented has to make sure that different users can decide which data they want to get advised.

Other very important technical requirements are number two and three. They say that the system should provide to add IDs to the data that is processed in the workflow so that end results can be allocated with the input data. The results are useless without utilizing these IDs since it is not possible for the user to connect input with output data and therefore, the meaning of the results cannot be understood. Although those data is important to analyse the results, it is very hard to save the IDs in the provenance data. The reason is that the IDs have to be generated by the modules themselves and only the provenance system can store them. However, not every used module, which is often provided by third parties, generates the data. Therefore, the provenance system has no chance to collect the data. This is also the reason why this technical requirement will not be supported and users still have to acquire the data from log files.

The technical requirement answering most use cases is the fourth. It shows metadata of input data and processes to ensure that the right information was used or the module was indeed designed to process with the used files. However, this requirement cannot only be provided by the provenance system. The used data and processes, which are often allocated or implemented by third party scientists, have to provide this information. The input data needs a special field at the beginning before the real content begins, which describes the input. Therefore, a standard format for metadata has to be adopted to make sure that the provenance system can collect them.

The fifth technical requirement is a special case, which is rarely occurring. It is only used if a workflow has parallel tasks, which should be compared. It is required for an analysis of the workflow tasks to find the most efficient module and to analyse whether they produce the

same output data. If the technical requirement is slightly modified, one could imagine that the architecture should provide for the comparisons between different workflow runs. The reformulated technical requirement is more often used than the original one because scientists can compare the influence of different parameters on the results more easily. Since the lineage data is the same for every workflow run, the Provenance Query Interface has to visualize the comparison between the runs.

It follows the technical requirement saying that it would be sensible to store contextual information and search over different workflows. This could be useful if a working group that executes many workflows is using a special database. The user can search in all previously executed workflows for the database if it is updated and the found workflows can be reproduced. For this kind of search method special search algorithms are necessary because it can take a long time to search for the information in all generated provenance files.

Another interesting requirement is the eighth saying that a provenance architecture should provide for a mechanism to store the lineage data in an unmodifiable way. This is necessary for legal affairs, as mentioned in the use cases. Additionally, saving the provenance data in a way that every author can be identified also helps to find out who executed the workflow and who created errors. Although this technical requirement answers only three use cases it is important. To achieve a high acceptance of the users they must be sure that they can rely on their data. To attain acceptance, a security model has to be implemented by using signatures or encryption.

Moreover, the ninth technical requirement has to be analysed. Users who executed a workflow must have the possibility to collect all data to re-run the workflow with the same or optimised data. An analysis which data is necessary to provide reproducibility is given in the chapter Reproducibility".

Very similar to the sixth technical requirement is the last one. It also states that a provenance system should provide for searching in the provenance data of all previously executed workflows. Instead of analysing the data, the focus in this technical requirement lies on administrative questions like calculating the runtime of previously executed modules to finally be able to evaluate the costs of new workflows using the tasks. Here, a search algorithm has to be implemented too, as in the case of requirement six.

3.3 Reproducibility

This chapter describes the term of reproducibility and the importance in provenance. The first section defines what is understood as reproducibility and the second section describes which data has to be collected to provide reproducibility.

3.3.1 Definition of Reproducibility

Results calculated with scientific workflows are often published in papers or in publicly available databases. However, these results are only accepted if the publisher can prove that the results are calculated correctly. One very accredited form that guarantees trust in the provided data is reproducibility.

A few years ago, the workflow-based community recognized that the reproducibility is a very important utility to arouse trust. By means of an example in the field of climate change, one can see the importance of reproducibility and the record of all data and processes that led to results in the field of science. As a result of the *Climatic Research Unit email controversy* (http://en.wikipedia.org/wiki/Climatic_Research_Unit_email_controversy), a parliament committee enacted that it is necessary to publicize all raw data and computer codes used in the research. By dint of the decision, the fact-finding commission saw the only chance to retrieve trust of the publicity in science. This way of thinking does not only exist in climate science but also in clinical trial results for drug approval or in social science to justify determinations on objective evidence. [5]

The workflow-based community also considered how reproducibility can be provided and came to the result that provenance is most adequate. The usage of provenance is best adapted for this use case since a rich representation of provenance allows steps to be reproduced and all intermediary and final results to be checked and validated. The community points out the importance of provenance in reproducibility as follows: ([5], p.1)

> *"Reproducibility requires rich provenance information, so that researchers can repeat techniques and analysis methods to obtain scientifically similar results... In order to support reproducibility, workflow management systems must capture and generate provenance information as a critical part of the workflow-generated data."*

Besides collecting and displaying the data necessary for reproducibility, one should consider whether the architecture can provide the ability to automatically re-run the workflow with the collected data. The reproducibility data should be linked to the workflow management system to run a workflow with the collected data. If the lineage data is published in context of a release of calculated data it must be ensured that the reproducibility data is readable for other workflow management systems too. This means that the data has to be stored in a structure that many organisations use to store their provenance data. One standard format is the Open Provenance Model. [5]
Thus, one can summarize the key benefit of using provenance for reproducibility, by means of the conclusion of Davidson and Freiere: ([7], p.3)

> *"A detailed record of the steps followed to produce a result allows others to reproduce and validate these results."*

This means that the recording of provenance data for result analysis or debugging can be easily used to reproduce the results since most necessary information will be collected anyway.

3.3.2 Required data for Reproducibility

A list of necessary queries and data to ensure reproducibility will now be given. The list was created by conversations to workflow users.

- Version of workflow management system:
  The different workflow management versions can handle the jobs differently and results can slightly differ.
- Connections of tasks:
  Since data is often modified and passed between tasks it is important to know which modules the results passed.
- Version of every task:
  The calculation algorithm can be changed or different errors corrected so that results can differ.
- Grid node where the module was executed:
  Different operating systems or processors on the nodes can calculate different results. The execution time and the communication can also differ due to different architectures.
- Start and End time of every module:

The execution time is a hint if the execution was successful, interrupted or deadlocks exist. This helps to compare if the re-execution was successful.

- Version of input data:
  The version is necessary if databases are used as input data since they are often updated and new results can follow-up.
- Path of input data:
  The path of input data is important since it must be recorded which files were used as input.
- Parameters of modules:
  The parameters are essential for reproducibility since they are a kind of input values and have impact on the results.
- Path of output data:
  To be able to compare results of different workflows it is necessary to save where the data is stored.

3.4 Suggestions of BLAST users

As mentioned before, a close contact to the users executing the workflows and analysing the results is a very important part of this thesis to make sure that the provenance system collects the needed data and will be accepted by the clients. Hence, several meetings with users were arranged to discuss the provenance systems. The results of those meetings concerning the data to be collected will be discussed in this chapter.
The two aspects, which are in the eyes of the bioinformaticians most important, are the simplification of information search and reproducibility. Table 2 will help to understand which kind of information should be searched.
The table contains statistical values for every workflow execution; those are listed in rows. To fill the columns of the table the authors needed to analyse many different logging files, and information from the workflow management system. They need a long time to collect all required information like number of samples (#Sam), number of executed tasks for all modules and those correctly finished. The wish of the bioinformaticians is that this data can be provided by the provenance system in a more user friendly way.

Table 2: Performance data for BLAST workflow executions ([24], p.1)

Exp	#Sam	#Seq	Workflow Tasks			Workflow Results			Jobs			Time (hrs)
			Sff2Fasta	Blast	ParseBlast	Ok	failed	success	Ok	failed	success	
A	96	37,632	95/96	190/190	189/190	190	3	98.4	474	10	97.9	4.2
B	44	2,338	44/44	88/88	88/88	88	0	100	220	0	100	3
C	48	149,949	48/48	96/96	95/96	96	1	99.0	239	3	98.8	3.2
D	93	205,258	93/93	186/186	186/186	186	0	100	465	7	98.5	10.5
E	12	36,721	12/12	24/24	24/24	24	0	100	60	0	100	4.2
F	45	13,974	45/45	89/90	89/89	89	1	98.9	223	3	98.7	4.8
G	24	34,541	24/24	48/48	48/48	48	0	100	120	0	100	3.2
H	45	9,096	45/45	90/90	90/90	90	0	100	225	1	99.6	3
I	27	7,463	26/27	52/52	52/52	52	2	96.3	130	3	97.7	3
J	54	474,821	54/54	106/108	105/106	106	3	97.2	265	14	65.0	4.6
K	53	504,277	53/53	106/106	106/106	106	0	100	265	5	98.1	3.7
L	55	383,796	55/55	110/110	110/110	110	0	100	275	2	99.3	3.5
M	56	368,975	56/56	112/112	112/112	112	0	100	280	8	97.2	3.8
N	56	65,794	55/56	110/110	110/110	110	2	98.2	275	5	98.2	3
O	14	97,252	14/14	28/28	28/28	28	0	100	70	0	100	2.8
ALL	722	2,391,842	719/722	1436/1438	1436/1436	1436	8	99.4	3591	12	99.7	13.7
Total	1444	4,783,684	1438/1444	2871/2876	2868/2871	2871	20	99.3	7177	73	98.9	74.2

The following provenance data has to be collected to define the data in the table:

- Farming ID:

  Every workflow execution consists of a set of workflow runs with different input data or parameter. Since single provenance files will be created it must be possible to group them to the current workflow execution with a unique ID for every workflow execution.
- Number of calls of tasks:

  It must be counted how often which task was called in every run in a workflow execution.
- Number of failed tasks:

  It must be counted how many tasks did not finish correctly.
- Execution time:

  The execution time of every workflow execution must be recorded.

The required values will be collected either directly or indirectly. How exactly the values are collected can be found in the chapter “Provenance in WS-VLAM”.

The other aspect mentioned by the users is reproducibility. The provenance data needed to provide reproducibility has been described in the previous sub-paragraph. Among the listed aspects the following data is in the eyes of the users the most important for reproducibility:

- Version of applied databases or data
- Version of modules
- Saved output files
- Workflow executer

However, the first two aspects cannot be achieved since versions of input data and modules always have to be delivered by the providers of the data or by the programmers of the modules. Furthermore, a standard for versioning was not yet inserted into WS-VLAM.

3.5 Provenance in WS-VLAM

This paragraph will finally discuss the assertions in this sub-chapter with main focus laid on the feasibility of WS-VLAM. It points out which use cases and technical requirements can be replied, which aspects of the reproducibility can be considered and which of the requirements of the BLAST users can be answered. Additionally, it will be described which data is collected and how they help to answer every question.

The following table gives an overview which data can be collected by WS-VLAM and which are currently not supported. A detailed description of the aspects follows.

Table 3: Comparison of collected and not collected provenance data by WS-VLAM

Collected data by WS-VLAM	Not collected data by WS-VLAM
Parameters of file readers and writers containing the input, intermediate result and end result file paths	Content of input, intermediate result and end result files
Data and catalogues for searching information in a set of provenance files	Content of files only passed between modules of a workflow
Execution times of workflows and modules	Metadata of input files or modules
Execution node of a module	Versions of input data, modules and workflow management system
Executer of a workflow	
Number of executed workflows in a farmed run	
Number of executed modules and correctly finished modules	
The collected data provides the possibility to show different views	

One of the most important domains mentioned in the previous paragraphs is used and generated data by a workflow. As provenance describes the derivation of data, all kinds of data used in a workflow should be saved. On the one hand there is the input data. The users want to know which of the input data was used by the modules and what the content of the data is. Since WS-VLAM uses file readers to read the input data it is possible to save the parameter of a file reader, which contains the path of the input file. Thus, the provenance system does not save the content of the files. The reason is that input data may have a size in the range of terabytes and the provenance system cannot handle file sizes like that. On the other hand there are intermediate and end results, which are saved with file writers. They also have a parameter containing the file's path and this path can be saved in the lineage file. Though, there are existing files, which are passed between different modules and will not be available after the transfer. If the user really needs the files for the analysis of the output data he can add a file writer storing the data in a consistently saved file.

Sometimes it is necessary to provide searching for data or comparing of data between different runs of the same or different workflows. One application for this is to search for all workflows using a special input file, which was changed and the user wants to re-run all workflows using it. Because all data required for this searching, like the input file path, is saved anyhow, it is not a problem of the WfMS but it has to be provided by the provenance system or the Provenance Query Interface. The way in which an efficient search algorithm is implemented over all provenance files will be described in the chapter "Transmission to web service".

The use cases from literature also mentioned the necessity of metadata or documentation to describe the modules. This part cannot be supported because there is no possibility to make sure that the used modules have metadata or documentation because they are often from third party organisations or the programmers forgot to describe the tasks.

Also very important are execution times in provenance. In WS-VLAM workflow and modules' starting and ending times can be saved. The starting and ending times also help finding modules, which did not finish correctly. If a module has no finish time and the workflow has finished, it can be inferred that the module finished with error. This method will be used in the provenance system to be implemented.

There is also some information, which is interesting for reproducibility of workflows. Examples for this are the execution node of a module or the executer. WS-VLAM also provides this data and can save it in the provenance file. The executer is ascertained via the credentials, which are generated if the workflow is executed. Other information, which is also important, is the version of input data, modules and workflow managements system. However, none of

this data is accessible by WS-VLAM because it is not in the hand of the WfMS. Another aspect is that the Provenance Query Interface will not be combined with the WfMS to re-run it automatically as the connection is not required by the users currently. This connection is possible in WS-VLAM but the feature will not be supported.
Most of the other provenance values can be provided by WS-VLAM like the number of executed workflows in a farmed run, the number of executed tasks and the number of correctly finished modules. The number of workflows of one run can be calculated by the number of workflows that have the same farming ID. The connection of modules, the parameters and the grid node that calculates the module are saveable by the provenance system, too.
The possibility to show different views with a provenance system is also possible with the data collected by WS-VLAM. All information, which is interesting for the different users, is collected. Thus, the visualization and the definitions how the different views are defined have to be conducted by the provenance system itself.
Therefore, one can say that some of the required data in literature and from the users cannot be answered because most of them are not in the area of WS-VLAM itself. Here some rules for the implementation of modules and input data has to be made and after adapting WS-VLAM this data could be collected, too. However, the most important data describing the derivation of the results can be collected.

4 History tracing XML-based provenance framework for workflows

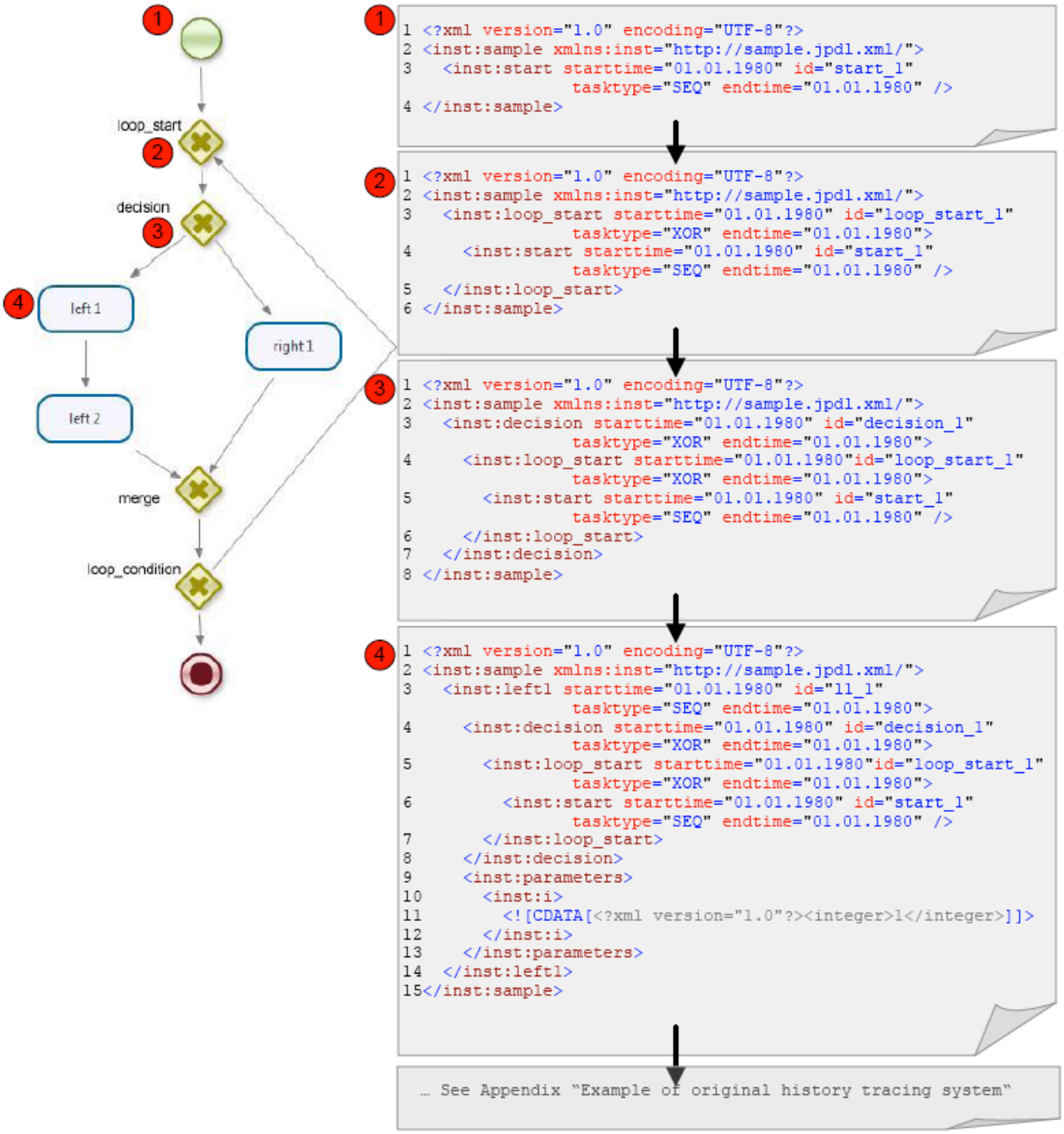
This chapter describes the history tracing XML-based provenance framework for workflows and starts with an overview of the provenance system. Then, the original system will be explained with main focus on the architecture, the web service, the XML provenance files and the signature of data. It is followed by the description of the re-implementation of the lineage system for WS-VLAM with focus on the same aspects like for the original system. The paragraph closes with a conclusion.

4.1 Introduction

The history tracing XML-based provenance framework for workflows, also called history-tracing system, was developed during the master's thesis written by Michael Gerhards at the University of Applied Science in Aachen in 2010. [11]
Within the context of the "History tracing XML for an Actor-driven Grid-enabled Workflow System (HiX4AGWS)" project the provenance system was developed. Besides collecting the provenance data, an important requirement was to save the data in a legally binding way. To provide this requirement the data was saved in XML whereas the workflow modules are saved in recursively nested layers with the youngest task containing all older ones. This structure allows signing the values of the current workflow task and all of the history in the XML document. After signing the document it is not possible to change the previous steps and the provenance data can also be used for legal affairs.
Figure 4 shows the process of collecting provenance data during the execution of the workflow. When the execution of the workflow starts (1), the provenance file is generated. First, the root element with the workflow name and second the start tag are inserted into the XML file. The file is saved on the hard disk with the collected information to enable the user to also analyse the provenance file during runtime of the workflow. Then, the workflow reaches the *loop_start* element (2) and the tag will be inserted in the XML file. This will be the child of the root element in the XML structure and its predecessor is the child element of the *loop_start* element. It follows the decision (3), which again gets the root's child element with the forerunner as child element. As one can see, the decision runs along the left path because the *left1* element follows into the provenance file (4). This XML tag contains a difference

because it owns a second child element called *parameters* containing the method's input parameters in a serialized type. The following element *left2* has also return values, which can be seen by the existence of the tag *results*. The elements *merge* and *loop_condition* follow. Since the parent element of the *loop_condition* tag is *loop_start*, the condition of the loop was true and a new run of the loop starts. This time the decision executes the right path before the loop condition comes false and the workflow stops after reaching the end element. This is as well the root's child node of the XML file since it is the last action of the workflow.

Figure 4: Example of building provenance file



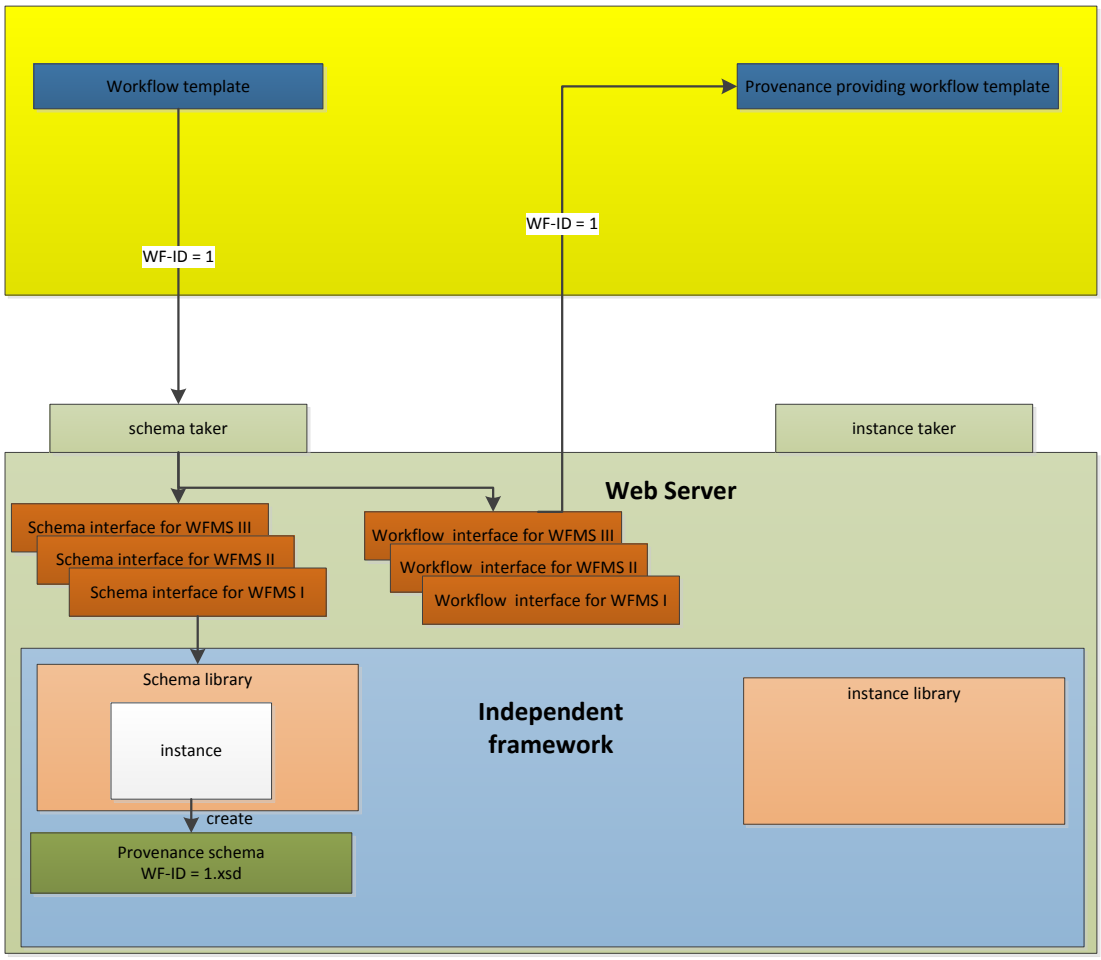
4.2 The original history tracing system

This section explains the idea of the history tracing XML-based provenance framework for workflows and the steps, which are necessary to collect the provenance data. A special focus will be put on the architecture, the necessary transmissions from the workflow management system to the web service, the structure of the XML provenance file, and the signature of the data.

4.2.1 Architecture

At this stage, a detailed description of the original architecture will be given. It is mainly based on the graphics shown in this chapter.

Figure 5: Architecture of preparing steps for provenance collection ([10], p. 6)



The XML tracing system contains two components. On the one hand is the workflow engine or workflow management system, which defines and executes the workflow. On the other hand is the webserver, which runs the functions to create the schema, the extended workflow and collects the provenance data.

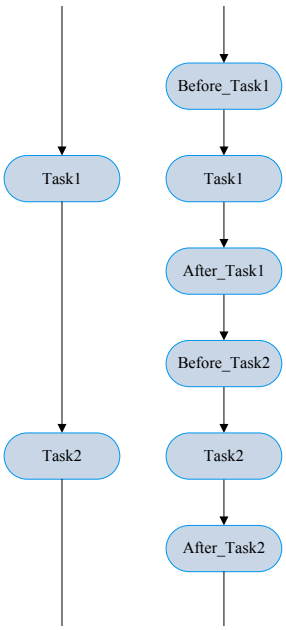
As always, the whole process starts with the creation of the workflow in the workflow management system of the users choice. When the user is ready with implementing the workflow it will be saved in XML format by the WfMS. However, to be able run the workflow and collect provenance data some additional steps have to be taken.

As a first additional step, the user has to create a schema file, which describes the provenance file generated by the history tracing system. For this, the user has to call the schema interface method on the web server. The method reads the content of the workflow file, analyses the structure and describes every workflow module that is used in the workflow with all input values, output values and the dependencies. Since the XML format of the workflow file is different for every workflow management system, it has to be adapted whenever a new workflow engine is used. The content and structure of the schema and the XML files will be described in the chapter "Transmission to web server and web service methods".

After analysing the structure and collecting the relevant data for the schema file, the data is passed to the schema library, implemented in a workflow engine independent framework. Subjoining the schema interface, which transforms the data in the independent format, enables the schema library to be independent from the workflow engine.

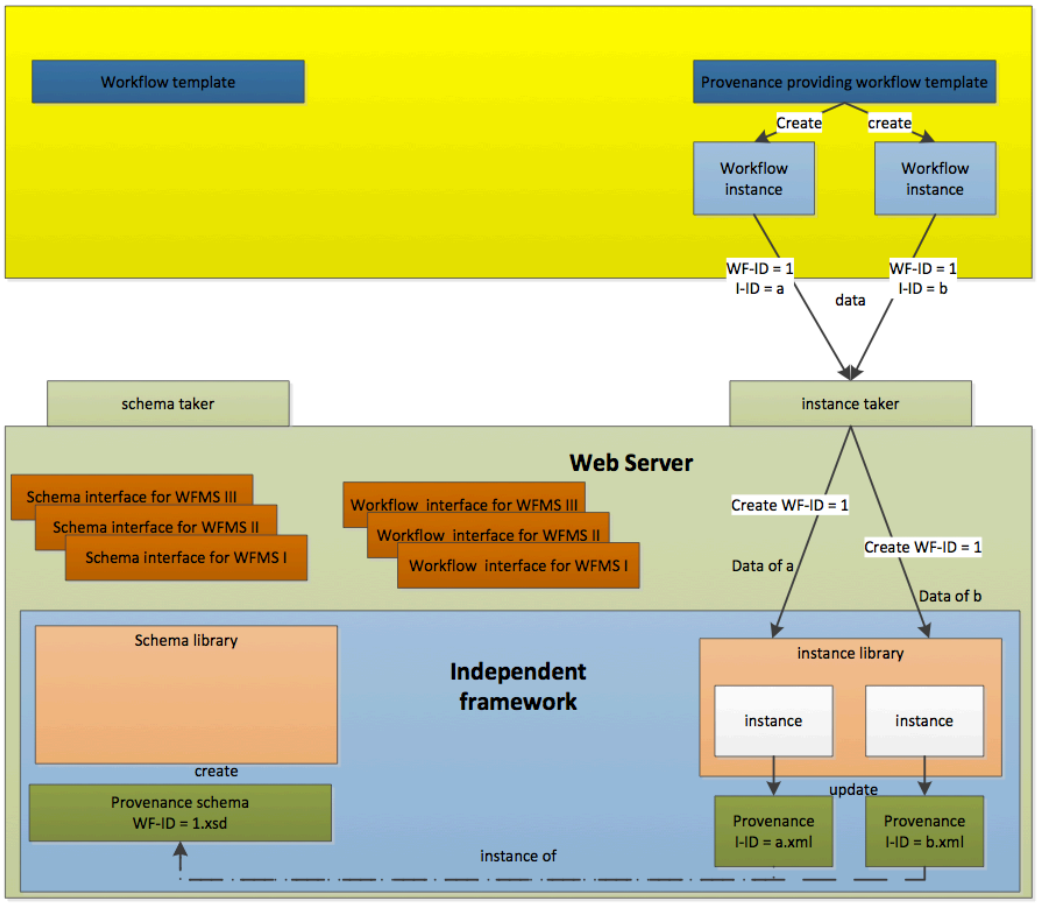
The second additional step extends the original workflow by tasks that collect all provenance information like the input and output parameters of the workflow tasks. To create the extended workflow file, another web service method is called – the instance interface. This method has also to be adapted to every workflow engine, because it writes a new WfMS independent workflow file, which has to be readable by the workflow management system.

Figure 6: Original workflow and workflow with additional steps ([11], p. 103)



As shown in Figure 6, every workflow step gets forerunner and follower tasks. Those tasks are necessary to record the modules and collect the input and output data of every original task. Since the history tracing system should be independent of a WfMS it is not possible to automatically extend the modules or the workflow engine with provenance data collecting functionality. In the case of the JBPM workflow engine the original tasks are Java methods. The added tasks are methods passing the name of the module, the values and the parameters to the web server. These modules have to be created by the user for the current workflow management system. However, those have to be created only once per workflow engine.
Furthermore, the described actions add the same unique identifier to the schema file and the extended workflow file. This identifier is called workflow-ID and helps combining the schema file with the provenance file.
At this stage, the preparations are finished. If the user now wants to execute the workflow he does not execute the workflow created with the workflow engine but the extended one, created by the instance interface. This means that a user has to execute these two web service methods always if he changes the original workflow to be sure that the modifications influence the results.

Figure 7: Architecture of provenance file creation ([10], p. 6)



The first action of the workflow management system after executing the workflow template with documentation modules is the creation of workflow instances. Every run of the abstract workflow creates a new workflow instance with the workflow-ID on the one hand and the instance identification on the other hand. The instance ID is necessary to execute several workflows parallel and being able to differentiate them so that the provenance system can assign the provenance data to the right provenance file.
The next step is a call of the web server method by the workflow management system to create an empty xml file. After that, the workflow engine executes all workflow tasks. Always if a documentation task is executed it calls web service methods and passes information to the provenance file. The corresponding web service method saves the values that are adequate to the schema in the XML lineage file.
The XML file gets filled with the provenance values after every workflow step. The speciality in the structure of the xml file is that it is valid against the schema at any time. The reason is that a workflow can break suddenly and the user must still be able to analyse the file in that case because a very important application range of provenance is the debugging of workflow executions and locating errors.
As soon as the workflow finishes, a web service method is called for the last time to conclude the provenance file and to deallocate the instance-ID. The user can analyse the created provenance file after concluding all these steps. Provenance queries can be answered and debugging or result analysis accomplished. As the inserted workflow tasks do not have any influence on the results of the original modules it does not make any difference which of the two workflows are executed. They will always produce the same results.

4.2.2 Transmission to web server and web service methods

As described in the previous section, the communication between the workflow management system and the provenance recording system is based on web services. This practice is necessary due to the different programming languages Java in JBPM and C# on the web server. In order to create the web service client methods, the program *wsimport.exe* from the Java Development Kit (JDK) is used to automatically create the java classes. Thereby it is very simple to create the interface between JBPM and the web service methods since they just have to be inserted in the project and can be called by the workflow tasks. {1}
In the following, it will be described how the methods are exactly called to pass all necessary information from the workflow to the provenance file. As mentioned before, a user executes a

workflow template with additional steps. All the additional steps are calls of Java methods. The following table shows which additional methods exist and their meaning.

Table 4: Java methods in WfMS ([11], p. 114)

Method name	**Functionality**
vor **Parameters:** String	It is called if the additional task before the real task is running. The method is called if no input parameters are used for the task Name of the workflow task that follows
vor **Parameters:** String Object	It is called if the additional task before the real task is running. The method is called if one input parameter is used for the task Name of the workflow task that follows Input parameter of the workflow task.
nach **Parameters:** string	It is called if the additional task after the real task is running. The method is called if no input parameter is used for the task Name of the workflow task
nach **Parameters:** string Object	It is called if the additional task after the real task is running. The method is called if a return parameter is used for the task Name of the workflow task Return value of the workflow task

These methods again call the web service client methods. Since web services can just pass strings from the client to the server, all parameters have to be converted from the original type to a string. Marshalling the input and return values to a string is achieved by the method *objectToString*, which converts the given *Object* to a string. Thus, the web service methods are called with the name of the workflow task and the converted input or output values. The name of the workflow task is necessary to create new layers in the XML file or to assign the values to the existing nodes. The following web service methods exist with the adapted meanings.

Table 5: C# web service methods and meaning ([11], pp. 121-122)

Web server methods	Functionality
Instance **Parameters:** string string	Creates a documentation instance by specifying WF-ID and I-ID WF-ID of workflow template to identify the schema ID of the workflow instance to differentiate documentation instances
Before **Parameters:** string string params string[]	Inserts new task steps to the provenance file with input parameters by giving the I-ID I-ID to assign the data Workflow task name Parameter values (only for Java workflow tasks)
Behind **Paramters:** string string string (optionally)	Finalizes workflow tasks by inserting return values by the I-ID I-ID to assign the data Workflow task name Return values (only for Java workflow tasks)
Close **Parameters:** string	Deletes the I-ID from the allocation table I-ID to delete

The values of the workflow engine are transmitted to the web server by calling these methods. Now it will be explained in detail how the provenance file creation is done on the web server.

The *Instance* method is a kind of constructor. It has the job to create the provenance file and add the object of the provenance file to a table containing all provenance instances. This is necessary to be able to execute the provenance system several times simultaneously and attach the provenance data to the right file. Every time the *Before*, *Behind* or *Close* methods are called, the right file can be found by scanning the instance table for the given instance ID.

The *Before* function is the forerunner of the real workflow task. It creates a XML tag with the given task name. Parameters of the tag, which are also created, are an id, important for the signature and the time stamp when the method was called. Beyond, the given input values are saved as a child of the workflow task. These are marshalled Java values so that they can be unmarshalled for the re-run of the workflow with the same values.

Executing the *Behind* method completes the task initialized by the *Before* action. The function therefore inserts a time stamp and optionally the return values of the function.

When the workflow is completed the file has to be closed and deleted from the provenance instance table. After that the used instance id is free and can be used for other workflows.

To picture the complete process an example follows showing how the single methods create the provenance file. The content of the XML provenance file will be described in the chapter "The content of the XML provenance file".

The following code abridgement contains the web service methods with the parameters creating the provenance file.

Figure 8: Web service calls on client side ([11], p. 68)

```
(1) InstanceFramwork ifw = new InstanceFramework("SequenceSchema.xsd",
                                "SequenceInstance.xml");
(2)
(3) ifw.Before("Start");
(4) ifw.Behind("Start");
(5)
(6) ifw.Before("Miss", "Master's Thesis");
(7) ifw.Behind("Miss");
(8)
(9) ifw.Before("Search", "Office");
(10) ifw.Behind("Search");
(11)
(12) ifw.Before("Find");
(13) ifw.Behind("Find", "Desk");
(14)
(15) ifw.Before("End");
(16) ifw.Behind("End");
```

The call of the constructor (1) creates a XML file with the name *SequenceInstanz.xml*, an instance of the schema *SequenceSchema.xsd*. The constructor also inserts a root element into the provenance file. The next method call (3) creates the innermost layer of the XML file with the XML tag *Start* and the parameters *datetime*, *id* and *wftype*. To know at which position the tag has to be inserted in the provenance file the instance library on the web server searches the XML schema to find the location in the instance file. The Behind method (4) sets the tag as closed. If the *Before* and *Behind* functions have a second parameter (6,9,13) it is the parameter of the corresponding workflow task. The task *Miss* (6) for example has the value *Master's thesis* and will be inserted in the element Parameter as a child of the *Miss* tag. The created provenance file looks as follows.

Figure 9: Provenance file of searching master thesis example ([11], p. 68)

```
<?xml version="1.0" encoding="UTF-8"?>
<inst:Root xmlns:inst="http://sequenz/" p1:schemaLocation="http://sequenz/SequenceSchema.
                xsd" wftype="ROOT" xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
  <inst:End datetime="2010-07-19T21:47:45.2671937+02:00" id="End_C_0" wftype="SEQUENCE">
    <inst:Find datetime="2010-07-19T21:47:45.2515687+02:00" id="Find_C_0" wftype="SEQUENCE">
      <inst:Search datetime="2010-07-19T21:47:45.2359437+02:00" id="Search_C_0"
                                                                wftype="SEQUENCE">
        <inst:Miss datetime="2010-07-19T21:47:45.1890687+02:00" id="Miss_C_0">
          <inst:Start datetime="2010-07-19T21:47:45.1734437+02:00" id="Start_C_0"
                                                                wftype="SEQUENCE"/>
          <inst:Parameter>
            <inst:Object><![CDATA[Master's thesis]]></inst:Object>
          </inst:Parameter>
        </inst:Miss>
        <inst:Parameter>
          <inst:Room><![CDATA[Office]]></inst:Room>
        </inst:Parameter>
      </inst:Search>
      <inst:Return>
        <inst:Location><![CDATA[Desk]]></inst:Location>
      </inst:Return>
    </inst:Find>
  </inst:End>
</inst:Root>
```

4.2.3 The content of the XML provenance file

This chapter gives an overview of the contents and structure of the provenance and schema file. This survey is important to point out which provenance information is collected. The kind of lineage data affects the usability of the provenance information and the application areas.

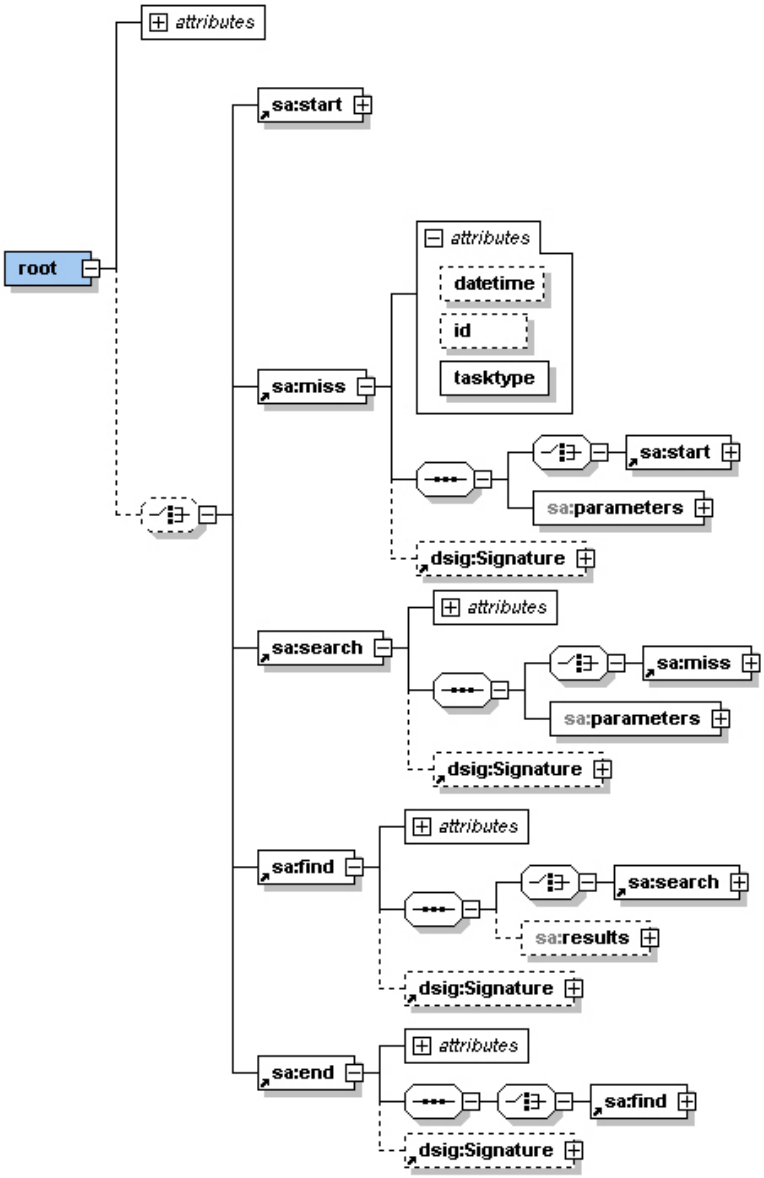
As stated above, the XML file contains the provenance data in a nested layered style. Whereby all layers are recursively saved with the youngest container comprising all older ones. This document style has the advantage that the workflow steps can be signed one by one and the workflow structure can be redrawn by just following the XML tags from the inmost layer to the root.

A schema file exists for all provenance files. To create it, all necessary information is collected from the workflow file. The workflow file contains the following information, which is necessary for the schema creation:

- Successor and Predecessor of every task
- Parameter values
- Return values
- Type of every task

Figure 10 shows a sample of a schema file with layers.

Figure 10: Schema of layered provenance file



The layered structure gets clear in this example because the *start* element is referenced in the *miss* element and this one in the search element. The start element, which has no child, is the starting element and the element *end* is not referenced, as it is the last element in the workflow. All possible signatures and parameters are also visible in the schema file.

When the workflow is executed and the lineage file created, the web service methods use the structure information from the schema file to sort the modules into the right order and to be able to know if a choice or parallel region follows. Ancillary to the structure are other information saved for every module, listed in the following table.

Table 6: XML tags in provenance file

XML Element	Meaning
Workflow name	The root element of the XML provenance file
Workflow task name	Representation of the workflow task which was executed
Parameters	Input values of the parent workflow task
Parameter name	Child of Parameters tag and contains the value of the input value
Return	Return values of the parent workflow task
Return name	Child of Return tag and contains the value of the return value
Regions	If a parallel region exists in the workflow the region element is inserted. It has as many children as parallel paths. The children are named like the names of the workflow tasks
Signature	Contains the signature of the parent element with all children excluding the signature tag

Additionally, parameters will be added for every tag shown in the following table.

Table 7: XML attributes in provenance file

Parameters of workflow tag	Meaning
datetime	Creation time of the tag. The web server inserts the time
id	Identification to combine tag with signature
wftype	The type of the current element:

wftype	Meaning
SEQUENCE	Normal workflow task
ROOT	Root element
REFERENCE	If a parallel region exists
CONTAINER	Contains a parallel region
PARALLEL_REGION	The complete region where two paths are parallel
PARALLEL_SPLIT	If two parallel paths start at the task
SYNCHRONISATION	If two parallel paths come together
XOR	If two paths start at the task
MERGE	Here a region ends, which started with a XOR.

As one can see, a parameter, called *wftype,* is part of every XML tag. This differentiation between different task types is made because they have to be handled differently. A *SEQUENCE*, for example, has always only one child element whereas the *PARALLEL_REGION* element has as many children as parallel paths exist. Hence, the different workflow types are a kind of identifier so that the web service can decide how it has to insert the element into the

file. The same is true for a Provenance Query Interface, which shows the provenance data. The *wftype* element also tells the Provenance Query Interface how to handle the data and the children.

As one can see in Figure 10, the schema file also contains an optional *Signature* element. Thus, the user can decide whether it is reasonable to add a signature to the provenance file to trust the content. The signature element is always a child of the signed tag. How the signing process works will be explained in the chapter "Signature".

Figure 9 shows that all parameters and return values are inserted as new tags of the current workflow task. They contain a serialized value in the simple data type CDATA. Indeed, it is not always necessary to save the values in CDATA data type, but all data is converted in CDATA data type to make them more equal.

Thus, one can sum up that the provenance XML file is always an instance of the schema file. The schema file describes the structure of the workflow and the runtime information is inserted by the web server methods in the provenance file. The provenance file is always valid against the schema to allow the analysis during runtime of the workflow.

4.2.4 Signature

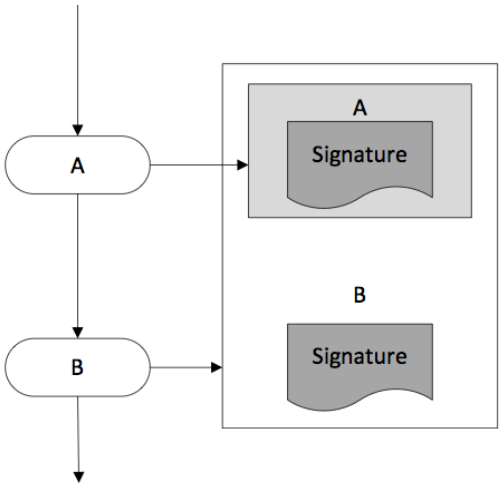
Often it is inescapable in the area of provenance to save the data in a legally binding way. As described in the chapter "Use cases and technical requirments" legal affairs exist in the use cases 14 and 15. The legal affairs require a mechanism to provide non-repudiation. Another example is if acts of sale are accomplished. In this case all values have to be saved in a way that the parties cannot doubt the contracts. In order to support a legally valid provenance, each executer of a workflow could be asked to sign his achieved part.

The signature is done with the XML-DSig and XAdES standards. "Both standards provide a general framework for digitally signing documents. XAdES, however, specifies precise profiles of XML-DSig for use with qualified electronic signatures and is thus an implementation that follows the European Union Directive 1999/93/EC". ([10], p. 3) In fact, the enveloped signature is used to follow the layered structure of the provenance files. The enveloped signature is used because it enhances the layered structure and the structure of the file is not disturbed.

Figure 11 depicts how the signing procedure is working. Workflow task *A* sends the lineage data in the provenance file. Then, the element is signed and the signature inserted in the XML tag *A*. After that, the workflow task *B* runs and inserts the provenance data in the file. *B* contains element *A* with regard to the layered structure described before. When the signature is

generated not only the element *B* is signed but also all content of the tag including *A* and its signature. This advances the security and non-repudiation once again. On the web server, the *Behind* method signs the tag as it is not changed after that method.

Figure 11: Mapping of workflow tasks to layers with signature ([10], p. 3)



The layered structure of the provenance file and signing every workflow step separately has the advantage that different users of the workflow can always sign the part they executed. Another advantage is that not all layers have to be signed and only important parts get a signature.

4.3 History tracing system in WS-VLAM

This chapter describes the changes, which are necessary to adapt the original history tracing XML-based provenance framework for workflows, tested with JBPM, to the workflow management system WS-VLAM. The changes and enhancements will be segmented in the parts "Architecture", "Transmission to web service", "The content of the XML provenance file" and "Signature". In every subchapter the problems of the original system will be discussed and at the end of every paragraph follows the concrete realisation.

4.3.1 Architecture

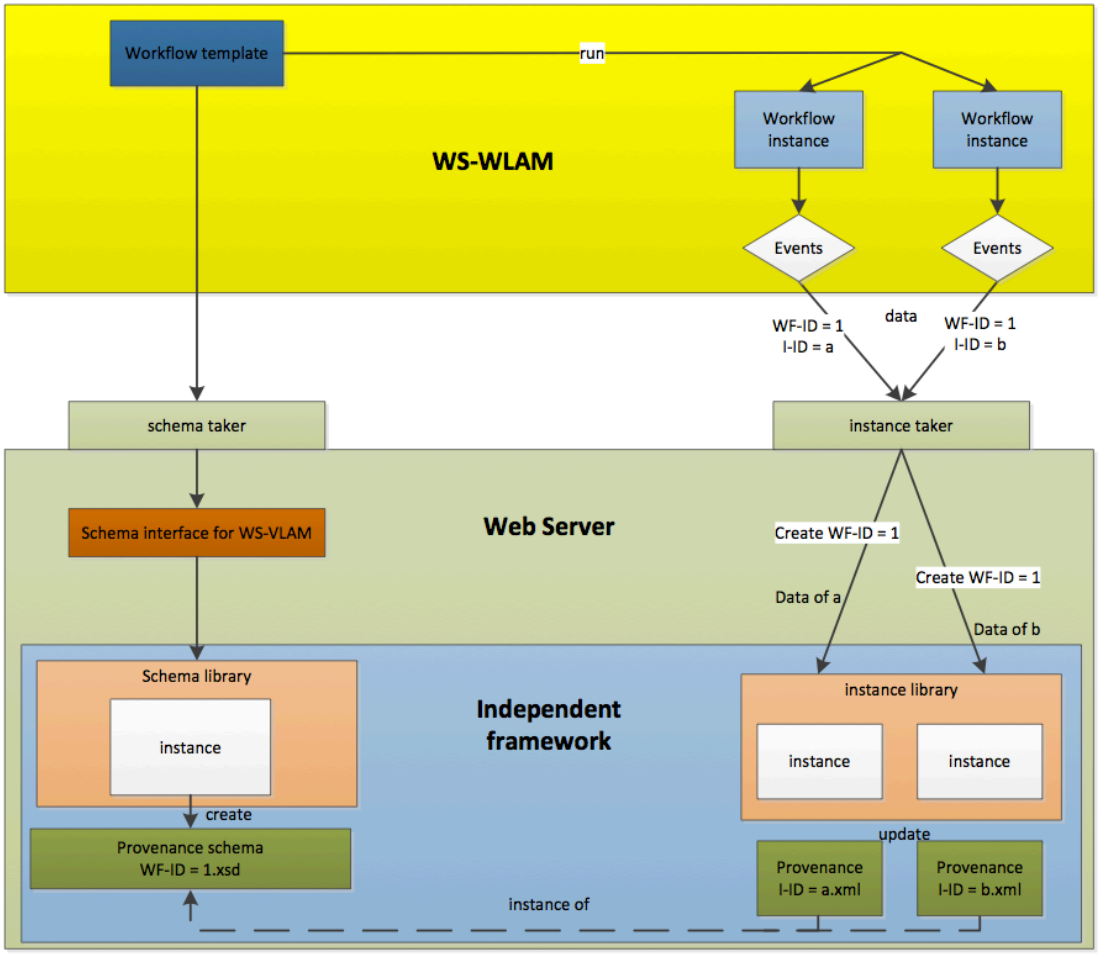
The new architecture differs immensely from the original system. Here, the necessary changes are especially inherent in the interface of the web server and the collection of the provenance values. There are different reasons that make changes in the architecture inescapable.
The first reason why a new architecture has to be implemented is that WS-VLAM is running on a grid, whereas the proof of concept achieved with the WfMS JBPM is running on only

one machine. The problem occurring when running a workflow on a distributed system is that the *Before* and *Behind* mechanism is not applicable. Since the WfMS distributes the workflow tasks on different computers on the grid it cannot be guaranteed that the *Before* and *Behind* tasks are running on the same execution node like the main module. A collection of parameters or other values would be unusable.

Another important fact is that WS-VLAM has a completely different method to program modules. In opposite to the in Java implemented tasks of the JBPM WfMS, every module in WS-VLAM is a single program written in programming languages like C/C++ or Perl. The communication is based on streams passing file contents from one to another module. This means that input and output values as in Java methods do not exist. Adding tasks, which read all provenance data is not possible to automate like for the additional tasks in JBPM.

The argument of using the additional tasks in order to be independent of the source code of the WfMS is not important for WS-VLAM. The author has full access to the source code and the WfMS can be extended to generate the provenance information.

Figure 12: Architecture of history tracing system with WS-VLAM



However, not all parts of the architecture have to be changed. Indeed, the framework part of the web server shown in Figure 12 can stay unchanged from the architecture's point of view. A more detailed description of the new architecture will now be given by describing the following figure.

In the new structure, the user's first action is to create a workflow and save it in a XML file. As a next step the user has to create the schema of the provenance file. The workflow file is sent to the schema taker of the web server and analysed by the schema interface. As stated above, it is obligatory to adapt the schema interface to the XML structure of the WS-VLAM workflow files. This is not a problem since the structure of the workflow file provides all needed information unambiguously and the file is in XML format, which is a condition to create the schema file. The schema library is from its architectural perspective the same as the one in the original provenance system. It just has to be adapted to the new collected provenance values. This will be discussed in the chapters "Transmission to web service" and "The content of the XML provenance file".

The workflow ID still exists in the new architecture. It keeps the same functionality and helps assigning the workflow to the generated schema and provenance files. The workflow ID is always added to the files and is the name of the workflow file. The workflow name is accessible during the schema creation because the workflow file must be passed to the web service and the workflow file name is useable for the provenance file creation as the RTSM can query the file name.

The first big difference between the original and the WS-VLAM versions of the history tracing system is the omission of the creation of the extended workflow. The demonstrated problems require another method to collect the provenance data. This means that the only preparation step is the generation of the schema file.

The provenance data is collected in the re-implementation of the history tracing system by creating events. Hence, when the user runs the original workflow in WS-VLAM, events will occur always if data relevant for the provenance system appear. Every kind of event calls different web service methods to send the data to the web server, which is storing the data in the lineage file. This means two different methods passing provenance data to the web server do not exist, but methods for every kind of event. Examples for generated events are the registration of a module, the setting of parameters or the finish of the execution.

As described previously, the procedure in the framework is the same for both versions of the history tracing system. After passing the event values to the web server, the provenance file

with the matching instance-ID is opened and the data is stored in the given module tag. The instance-ID is generated by the workflow name and the current time stamp.
Indeed, it is easy to collect new provenance data. The developer has only to add a new event and a new web service method to save the data. This is easy to realize because the new web service methods do not affect the existing events and methods.
Anyway, it is not guaranteed that the events appear in the chronological order. It may happen that a *fileWriter* module is destroyed before the forerunner is destroyed. This has an influence on the signature and will be discussed in the chapter "Signature". Another point arising by the random order and the many different events is that the provenance file has to be searched recursively to add the provenance data at the right module. This needs some extra time and slows the workflow engine down. However, this effect is not too huge for small workflows with just a few dozen modules like the BLAST workflow.

4.3.2 Transmission to web service

As stated above, the communication between the workflow management system and the web server writing the provenance file remains. The structure is maintained because only minor changes are necessary in the web server and the problem of the two programming languages with Java for WS-VLAM and C# for the web server remains.
The web service client methods were inserted into the WS-VLAM source code after generating them with the *wsimport.exe* application. {1} After adding them to the source code of WS-VLAM it is possible to call the web service methods by the WfMS if an event happens during the runtime.
The following events are generated and handled by WS-VLAM. If they occur they call the web service methods to pass information to the provenance file.

Table 8: Events of WS-VLAM, which call web service methods

Event	Meaning
All events	Every event has the following parameters ***moduleName:*** The name of the module ***moduleID:*** The unique ID of the module ***dateStump* :** The time stamp of the event
ModuleRegisteredEvent	Occurs if a module is created in the Run Time System Manager (RTSM)

Event	Meaning
PortResolvedEvent	Occurs if the end of a connection between two modules was set ***port:*** The port of the connection ***portName:*** The name of the modules' entry point. The user sets the name ***hostname*** **:** The name of the host containing the connection
ConnectionDoneEvent	Occurs if the beginning of a connection between two modules was set ***port:*** The port of the connection ***portName:*** The name of the module's leaving point, which is set by the user ***hostname*** **:** The name of the host containing the connection
ParameterSetDoneEvent	Occurs if a parameter of a module was generated with the initial value ***parameterName*****:** The name of the parameter defined by the user ***parameterValue*****:** The initial value, which is assigned to the parameter
ParameterChangedEvent	Occurs if a parameter value was changed **parameterName:** The name of the parameter defined by the user **parameterValue:** The changed value, which is now assigned to the parameter
StdoutEvent	Occurs if a module writes text on the standard out ***message*****:** Contains the text written on the standard out
StderrEvent	Occurs if a module writes text on the standard error ***message*****:** Contains the text written on the standard error
StdoutClosedEvent	Occurs if the standard out stream is closed after finishing the execution
StderrClosedEvent	Occurs if the standard error stream is closed after finishing the execution
ExecutionFinishedEvent	Occurs if the execution of an event was finished.
ModuleFinishedEvent	Occurs if an event is destroyed and removed from RTSM. This is the ExecutionFinishedEvent time plus some overhead

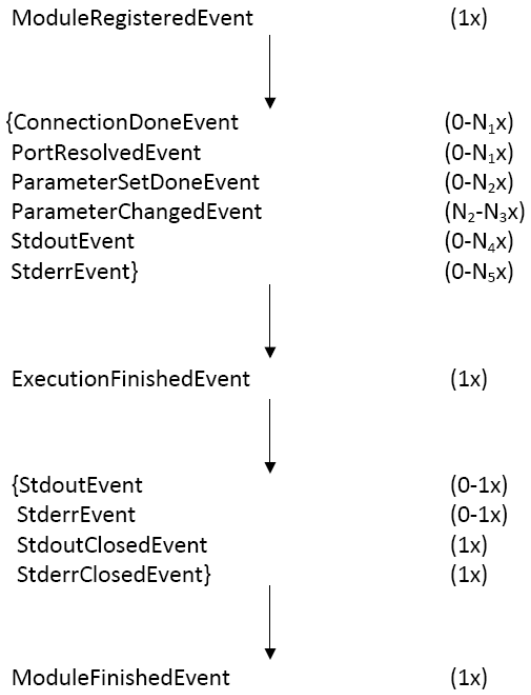
There are also two important states, which are described now.

Table 9: Finish states of workflows in WS-VLAM

State	Meaning
DONE	The workflow was finished without any error
FAILED	The workflow exited with an error that cannot be fixed

If the status changes to DONE or FAILED it will be registered by RTSM and a web service method is called to finish the creation of the provenance file with or without error flag.
The events for one module cannot appear completely randomly, as there are some rules that define an order. This is visualized in Figure 13.

Figure 13: Order of events of WS-VLAM



The event blocks will be generated in the order as shown. However, the events in such a block can appear in a random order. Every event is obligatory without the ModuleRegisteredEvent, which must be generated if the module starts. If the module finishes the events ExecutionFinishedEvent and ModuleFinishedEvent are mandatory, too. In the right column the number of appearance is given whereas the latter *N* shows that the event can occur arbitrarily often. The small number next to the *N* shows if other events can happen as often another one.
If one of the described events or states occurs, the adequate web service method is called. The existing kinds of methods as well as the functionality will be described in the following table. The parameters are not individually mentioned since they are the same as the variables described together with the events. There is also one additional parameter that is important for every method. It is the parameter *provenanceFileName*, which is the instance ID making sure that the values are inserted into the right provenance file. It is created by the name of the workflow and the time stamp.

Table 10: Web service methods and meaning

Method name	Functionality
Instanz	Creates a new instance of the provenance file and inserts the provenance file in the catalogue
schemaFile	The path of the schema file describing the provenance file
provenanceFileName	The path of the provenance file, also instance ID
timestampHuman	The time stamp of the creation in a human readable way
timestamp	The time stamp of the creation in a technical way
executerName	The credentials name of the executer
workflowName	The name of the workflow determined by the workflow file name
farmingID	The ID of a farming run. To connect all files from one farming run
SendCredentials	Is called if the first event is generated. It sends the content of the credential file to the web server
provenanceFileName	The path of the provenance file
credentials	The content of the credentials file
Initialize	Is called if a module is registered. Adds the tag with the module's name with the starting time and the ID to the provenance file
Function for every event	Are called if an event is generated. Adds the information of the connection to the appropriate module
ExecutionFinished	Is called if the execution of a module is finished. Adds time stamp to the appropriate module
End	Is called if a module is destroyed. Is the last event of the module and inserts the time
Close	Is called if the state changes to DONE or FAILED. Adds the endtime or errortime to the provenance file. Closes the file and removes it from the instance list. It is always the last action of a provenance file
provenanceFileName	Name of the provenance file. Is the instance ID
timestamp	The time stamp when the events happen
status	The exit status of the module. Either DONE or FAILED

The last part of this paragraph describes the exact behaviour on the web server and how the provenance file creation is accomplished. As soon as a workflow is executed, the first action on the server is the creation of a provenance file and adding the name of the provenance file to a table containing all provenance file names. This task is done by the method *Instanz*. It is called before any event is generated during the initialization phase of the RTSM. After that all web service method calls are achieved if an event occurs. The queue of events always begins with the *ModuleRegisteredEvent* that will appear at any time and call the function *Initialize*. This adds the tag of the new module to the provenance file. Because it creates the tag of the module it has to be the first event describing any lineage values of the task. Thus, problems are circumvented since the *ModuleRegisteredEvent* occurs directly after the initialization of the task and no other event can happen before.

Consecutively, the events of all modules are able to occur in a random order. As the order of the events of one and the same module and the order between the different tasks are random,

every module has to be searched recursively in the provenance file. This might take some time if there are many modules but another procedure is not allowed because of the randomness. Every module tag is completed if the *ModuleFinishedEvent* occurs and one can be certain that no other event concerning the same task will happen after that. One can be sure that it is the last event since the *ModuleFinishedEvent* is always thrown when destroying the module was finished. After receiving the last *ModuleFinishedEvent* all modules finished and the workflow has to be completed. If the state of the workflow changes to *DONE* or *FAILED* the method *Close* is called to close the provenance file. If the workflow finishes with the state *DONE*, it stops without any errors and the parameter *endtime* is added in the root element. In the other case it closes the provenance file with the parameter *errortime*. Certainly, an error can happen at any time during the execution of the workflow and immediately when the error happens the provenance file is closed. The reason for immediately stopping the lineage collection is that no other events will happen and no provenance information will be collected and written to the provenance file. Finally, the instance ID of the provenance file is deleted from the instance list.

In order to select the created provenance files in the Provenance Query Interface the path of every provenance file will be inserted in a catalogue file when the lineage file is created. The catalogue file is a XML file that contains tags with every created provenance file and some properties. The currently saved properties are workflow name, creation time, status of workflow, the name of the executer and the farming ID. The farming ID helps to find all lineage files of one farming run in the catalogue. The entry in the catalogue file is accomplished by the method *Instanz* and the finish status is inserted by the function *Close*. Moreover, two other catalogue files exist: on the one hand a catalogue containing the paths of input files and on the other hand the paths of all output files. The entries are created always if the method *AddParameterChanged* is called and the module is a fileReader or fileWriter. An explanation of the way in which the files are used in the Provenance Query Interface is given in the chapter “Provenance Query Interface”.

The code abridgement in Figure 14 shows some of the web server methods and their parameters. Running this code calls the constructor of the class *InstanceFramework*. It creates an instance of the schema file *BLAST.xsd* with the name *BLASTInstanz.xml* and inserts the path of the instance in the catalogue with the time stamp, executer, workflow name, and farming ID. After that, the two modules are initialized and the tags are inserted in the XML file with ID and starting time as parameters. Then, the other provenance values follow and will be inserted as children of the given module. All parameters are inserted into the element *infor-*

mation and ordered by the meaning of the lineage values. When the execution finishes the *ExecutionFinished* function is called and the time stamps are inserted as parameters to the module tags. The same happens with the time stamps of destroying the modules done by the function *End*. At the end the function *Close* inserts the *endtime* of the workflow and sets the status of the provenance file in the catalogue as *DONE*. The resulting XML file can be seen in the "Appendix" in "Example of XML file by history tracing system for WS-VLAM".

Figure 14: Web service calls of a fileReader/fileWriter example

```
(1) InstanceFramework ifw = new InstanceFramework("C:\\BLAST.xsd",
                "C:\\BLASTInstance.xml", "Thu Feb 10 16:58:54 CET 2011",
                "2011:02:10:16:58:54", "matzerat", "BLAST", "5793845");

(2) ifw.Initialize("fileWriter", "916452136", "Thu Feb 10 16:58:56 CET 2011");
(3) ifw.Initialize("fileReader", "741287972", "Thu Feb 10 16:58:55 CET 2011");

(4) ifw.PortResolved("fileReader", "741287972", "Thu Feb 10 16:58:59 CET 2011", "1234",
                "port1", "node.node1");
(5) ifw.ConnectionDone("fileWriter", "916452136", "Thu Feb 10 16:58:59 CET 2011", "1234",
                "port1", "node.node1");

(6) ifw.AddParameters("fileWriter", "1464339253", "filename", "C:\\Temp\\file.xml",
                "Thu Feb 10 16:59:03 CET 2011");
(7) ifw.AddParameters("fileWriter", "1464339253", "farmed", "false",
                "Thu Feb 10 16:59:03 CET 2011");
(8) ifw.AddParameters("fileReader", "741287972", "filename", "C:\\Temp\\input.tgz",
                "Thu Feb 10 16:59:05 CET 2011");

(9) ifw.AddParameterChanged("fileReader", "1464339253", "filename",
"C:\\Temp\\input.xml",
                "Thu Feb 10 16:59:06 CET 2011", "C:\\BLASTInstanz.xml");

(10)ifw.ExecutionFinished("fileReader", "1464339253", "Thu Feb 10 16:59:30 CET 2011");
(11)ifw.ExecutionFinished("fileWriter", "916452136", "Thu Feb 10 16:59:35 CET 2011");

(12)ifw.End("fileReader", "1464339253", "Thu Feb 10 16:59:37 CET 2011");
(13)ifw.End("fileWriter", "916452136", "Thu Feb 10 16:59:38 CET 2011");

(14)ifw.Close("Thu Feb 10 16:59:40 CET 2011", "DONE");
```

4.3.3 The content of the XML provenance file

After explaining how the data is transmitted to the web server, this chapter describes how the data is saved in the provenance XML file.

First, it can be said that the XML structure within the provenance file remains as in the original history tracing system. It is just adapted to the new architecture with collecting the provenance data via events and it is expanded to save the values described in the previous chapter. This means that the layered structure is still used in the new history tracing system. The advantages of the layered structure as including the visualization of the workflow dependencies and the possibility to sign every workflow module are still valid. Thus, one crucial difference exists between the structures of the two history tracing versions.

Due to multiple starting and ending points in a workflow, created with WS-VLAM, it is necessary to break the layered structure. This problem was not handled because workflows in JBPM always have only one starting and one ending point and all paths starting at a parallel split always synchronize at the same point. Thus, this requirement is not existing in WS-VLAM and another XML structure has yet to be developed. The following example shows how the new structure looks like.

Figure 15: Workflow with two end modules

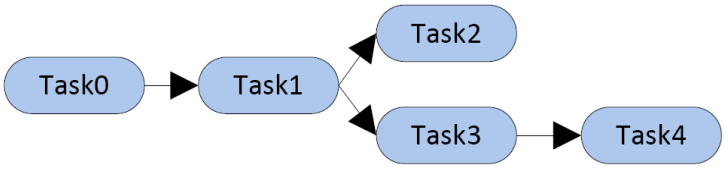


Figure 16: XML provenance file with two end modules

```
<Workflow>
  <task2 id="345435433" starttime="12.02.2011 12:34:56">
    <task1 id="759576896"/>
  </task2>
  <task4 id="235454667" starttime="12.02.2011 12:34:56">
    <task3 id="678456332" starttime="12.02.2011 12:34:56">
      <task1 id="759576896"/>
    </task3>
  </task4>
  <task1 id="759576896" starttime="12.02.2011 12:34:56">
    <task0 id="46556767" starttime="12.02.2011 12:34:56"/>
  </task1>
</Workflow>
```

As the example shows, *task1* is the starting point of two independent paths of the workflow. Since both paths are relying on *task1* it is necessary to save it in both elements. In order to keep the overhead as small as possible *task1* is defined as an element on the highest layer and only references are inserted into the two paths. Much overhead would arise to save the complete history of *task1* in the tasks two and three instead of the reference. Another characteristic of a reference is that it has no parameter.

Adding a single starting point and a single ending point accessorily to the workflow to generate a workflow structure used in the original history tracing system is also no solution. At first view this looks like a solution of the problem but it makes the creation of the provenance file more complicated and only slides the problem to another point of the provenance file. If one inserts one starting and one ending point it is not possible to achieve the other requirements described in the original history tracing system regarding forks, joins and other elements. This is a general problem because it is not easy to describe workflows defined as graphs with mul-

tiple starting and ending points by XML documents, which are trees. However, the use of the references is a good way to solve this problem by saving as few overheads as possible and keeping the layered structure as unchanged as possible.

As stated above, one schema file is created for every provenance file. The workflow file provides some information for the schema as well, which is used as provenance data, too. It contains the following information:

- The names of the execution nodes of the modules
- The names of the modules
- The IDs of the modules
- Successor and predecessor of every task
- How many connections start and end at the modules
- How many parameters every module has

Using the information provided by the events and the schema file provide, the provenance file is created if the user runs the workflow. Besides the information that are determined by the schema file like predecessor and successor as well as name of the execution node, the web server will also insert the information transmitted by the events in the provenance file. The tags that can occur in the lineage file are listed in the following table. For some elements a code sample is inserted to visualize the content.

Table 11: Possible tags in history tracing XML file

XML Element	Meaning
Workflow name	The root element of the XML provenance file
Workflow task name	Representation of the workflow task, which was executed
information	It is a child of the module; it describes and contains all provenance information, which are collected for the module

parameters	It is a child of information tag. It contains all lineage information that is dealing with parameters.
	`<inst:parameters>`
	`  <inst:filename>`
	`  <inst:initialValue time="date">C:\\input.tgz`
	`  </inst:initialValue>`
	`  <inst:parameterChanged time="date">C:\\input.xml`
	`  </inst:parameterChanged>`
	`  </inst:filename>`
	`</inst:parameters>`

XML Element	Meaning
portStart	It is a child of information tag. It contains all provenance information that is dealing with the starting point of a port `<inst:portStart><inst:start time="Thu Feb 10 16:58:59 CET 2011"><inst:id>4875756667</inst:id><inst:portName>toOutput</inst:portName></inst:start><inst:cluster>node.node1</inst:cluster><inst:port>1234</inst:port></inst:portStart>`
portEnd	It is a child of information tag. It contains all provenance information that is dealing with the ending point of a port. `<inst:portEnd><inst:start time="Thu Feb 10 16:58:59 CET 2011"><inst:id>890349345</inst:id><inst:portName>fromInput</inst:portName></inst:start><inst:cluster>node.node1</inst:cluster><inst:port>1234</inst:port></inst:portEnd>`
events	It is a child of information tag. It contains events that are appearing at every module and which are not dependent of the workflow structure or settings
stdoutReady	It is a child of events. It contains all text that is written on standard out `<inst:stdoutReady><inst:text time="Thu Feb 24 17:39:57 CET 2011">Output Text 1</inst:text><inst:text time="Thu Feb 24 17:42:37 CET 2011">Output Text 2</inst:text></inst:stdoutReady>`
stderrReady	It is a child of events. It contains all text that is written on standard error. `<inst:stderrReady><inst:text time="Thu Feb 24 17:39:57 CET 2011">Error Text 1</inst:text><inst:text time="Thu Feb 24 17:42:37 CET 2011">Error Text 2</inst:text></inst:stderrReady>`
stdoutClosed	It is a child of events. It contains the time stamp of the closure of the standard out stream.
stderrClosed	It is a child of events. It contains the time stamp of the closure of the standard error stream.
Signature	It contains the signature of the parent element with all children excluding the signature tag.

Every module tag and the root node have also parameters, which are specified in the following table. If a module tag is missing the parameter *starttime* is a reference of the task and is described in detail at another position in the provenance file as a root's child. Some of the

XML tags that describe events also have a *time* attribute and will not be explained in the following table. The appliance of the attribute can be seen in the examples of the previous table.

Table 12: Parameters in history tracing XML file

Parameters	Meaning
starttime	It is a root and module parameter. The time of the workflow / module start
endtime	It is a root and module parameter. The time when the module is destroyed / the workflow ends
errortime	It is a root parameter. The time when an error occurs in the workflow
executefinish	It is a module parameter. The time when the execution of the module finishes
node	It is a module parameter. The name of the node, which executes the module
id	It is a module parameter. The unique identifier of the module
name	It is a module parameter. The name of the module shown on the GUI of WS-VLAM

In contrast to the original history tracing, the new system saves no parameters describing the type of the module. Storage of this value is not necessary anymore because no different types of modules are consisting in WS-VLAM. Searching and writing data in the provenance file is achieved by a recursive search in the document now. The loss of the information, which elements are inserted and in which order, makes it obligatory to apply the recursive search.

Like in the original system, a signature is also available for the WS-VLAM implementation of the history tracing system. The signature will be inserted automatically by the provenance system generated with the credentials of the executer. A more detailed description of the new signing technique will be given in the next sub-chapter.

An example of the schema file and a XML provenance file for the BLAST workflow can be seen in the "Appendix" in the sections "Schema file for BLAST workflow" and "XML provenance file file for BLAST workflow with the history tracing system".

4.3.4 Signature

The considerations of security are for the new implementation of the history tracing system as important as for the original one. This is the reason why this feature is still available in the re-implementation. As the re-implementation runs together with another WfMS, which has many approaches in the field of security it will be adapted to the new possibilities.

Beforehand, it should be said that the implementation in this thesis is just a proof of concept and some aspects do not achieve all security concepts due to a limited amount of time. None-

theless, one can see the possibilities an enhancement could provide for security. The potential problems of the current signing process will be pointed out in this chapter and possible solutions will be discussed.

As stated above, WS-VLAM provides a security system, which can also be used in the provenance system. The security in WS-VLAM is based on credentials and is used to ensure the authenticity of the user running the workflow and using the resources on the grid. Now these credentials, which are anyway created for workflow execution, will be used for the signing process of the provenance data. The complete process for signing is working as follows.

When the workflow is started the credentials are created and can be accessed by the RTSM process of WS-VLAM. If the first event is generated, the credentials are transmitted to the web server by calling the method *SendCredential.* A string containing the credentials and private key is transmitted to the web server. It is obvious that this practice is not secure because other persons could read the credentials and the private key so that security is not assured anymore. One alternative could be to remove the process of sending the credentials via web service methods and use the MyProxy Credential Management Service instead. This allows saving X.509 proxy certificates with a temporal password in a repository and later being able to download the certificate above the network. In the case of the history tracing system the WS-VLAM process can save the credentials with the private key in the MyProxy repository and the web server can download the information to sign the document. [23]

As a next step, after transmitting the certificate file to the web server, the information has to be saved in a file in X.509 format. Since the C# classes in the namespace *System.Security.Cryptography.Xml* cannot handle the format transmitted from WS-VLAM it is necessary to convert the saved file. This is done by the openssl command and as a result a file in the PKCS#12 format with the file ending *.pfx* is created. {3} The new format contains the same information but saves them in another structure than the old one. [22]

If a module finishes the execution now and if the event is occurring, the signing process has to be started using the private key saved in the *.pfx* file. To be sure that the signature is not destroyed one has to reassure that no event happening after the signing process is inserted in the signed module. Here, a problem occurs due to the fact that the modules can happen randomly. As stated above is it possible that a successor is destroyed after the predecessor. A signature would be invalid if the values of the predecessor are inserted in the signed tag. To be sure that the elements are occurring in the right order the *executionFinished* event was developed, which is always in the right order. Thus, events like *stdoutClosed* are always happening after the *exectutionFinished* event. These cannot be collected when the user wants to sign the prov-

enance file. However, this is no problem since these events are not very important for the analysis of the results.
Now the signing process is finished and the data can be validated with the same certificate after finishing the workflow. The validating process could be accomplished in the Provenance Query Interface to be implemented by clicking a button because this is the user-friendliest technique.

4.4 Conclusion

This main chapter begins describing the original history tracing system and lies the main focus on the aspects “Architecture”, “Transmission to web server and web service methods”, “The content of the XML provenance file” and “Signature”. In the course of the chapter the necessary re-implementations in the history tracing system and the WfMS are pointed out with the main focus on the same aspects as for the original system. This paragraph summarizes the main results of the discussion in the chapter.
There are two facets, which completely changed compared to the original history tracing system. The first is that the creation of the extended workflow is replaced by the provenance collection by events. Thereby, the accountability of the provenance data collection changed from the workflow to the workflow management system. This means that the functions that send the provenance data to the web server are not called by the workflow anymore but have to be added to the source code of the WfMS. It also implies that access to the source code exists. Ancillary to the necessity to be able running the provenance system with WS-VLAM there are also advantages like the possibility to collect more different data. The connection to the WfMS has the advantage that many data is not passed to the workflow and this data could not be collected with an extension of the workflow including the workflow name or the executer.
The other aspect, which is adapted in the re-implementation of history tracing, is the enhancement of the XML structure. On the one hand the adaptation of multiple starting and ending points has been added and on the other hand the XML file was adapted to the event-based structure with events the users also asked for. The enlargement to be able to save workflows with multiple starting and ending points was necessary due to the fact that this kind of workflows was not examined. The new data that is saved in the provenance file do not change the XML structure in an immense way. Thus, the new data immensely appreciate the history tracing system because the system is more adapted to the user’s needs now.

Summing up, one can say that most parts of the architecture of the history tracing system could be taken over for WS-VLAM and the deployment of the web server based technique was successful and only minor changes are necessary in the web server for a new WfMS. Other aspects of the original system like the security aspect and the schema creation are also good fundaments to adapt the system to WS-VLAM or other WfMS.

5 Provenance Query Interface

The *Provenance Query Interface* was developed in close collaboration with the bioinformaticians of the *Academisch Medisch Centrum* (AMC) of the University of Amsterdam who are using the BLAST workflow for their research. After discussing which provenance data can and should be collected, the Provenance Query Interface was designed to support them in doing their analysis on the workflows.
In this chapter, it will be discussed which analysis can be done with the provenance data and how the Provenance Query Interface should be designed to display the data first. The second paragraph describes the resulting implementation and the functionality of the interface.

5.1 Provenance Query Interface Analysis

The need for implementing a Provenance Query Interface is obvious. The structure of a XML file as it is used in the history tracing system can be very complex and for many users it is not easy to find out the information they are looking for. An interface can help to visualize the data concerning the needs of the scientists. To learn about the requirements of the users, especially from the BLAST users, several meetings were held with the intention to develop a Provenance Query Interface feasible by users of other workflows and designed in a way that different user groups can use it.
At the beginning of the discussions it was emphasised that it is indispensable to select provenance files by categories. A selection that filters the provenance files by different categories is necessary as the complete data set of provenance files will extend after every run and it gets too confusing to handle all files at once. The categories could be the name of the workflow described by the provenance file the time when the workflow was executed. The latter one should be specified by a time span since the user does not always know exactly when he executed the workflow. Moreover, it is reasonable to categorize according to the executer of the workflow, because one may not be interested in the results of others. Beyond, it is possible to select all workflows by the state, which can be *Running, Finished Correct* or *Finished Error*. To be able to select workflows by input or output files it is achievable to insert a file name and get all provenance files using or generating these files.
When the choice is done, all found provenance files should be listed and the user can select them in order to display the provenance information in detail. Here, it is possible to select a

single provenance file or a set of provenance files that were executed in one farmed run and achieve the settings. The selection of a farming set is sensible because it helps to see at first glance which of the workflows finished correctly and which not, so that the definition of statistical values is facilitated.

After collecting a single provenance file, the user should be able to view the provenance data saved in the file. On the one hand, an overview should be given by re-drawing the workflow as a graph showing all modules that were executed so that it can be seen at one sight which modules are dependent on each other and which modules were not executed. On the other hand, all other provenance data selected by the events has to be depicted. It was decided to do this with a replay mechanism that prints out the contents of the events that occurred during the runtime. During the replay, all events will be displayed in the same time gap as they occurred in the real workflow helping to demonstrate the events or errors in time dependence. Visualization like that helps to point out the dependence between single actions of the WfMS and how errors arose.

Since persons with different roles and a different view on the data will use the Provenance Query Interface it must be possible to select messages that should be displayed. That means that a scientist developing the workflow might want to know information about the dependencies and if the execution finished correctly, whereas an end user is more interested in the input and output files. At the end, the user interface should be useful for almost all operators and for this, a selection of displayed provenance data should be implemented that allows to select the messages, which can answer his/her queries.

The query interface should also contain some features helping the users to create a table like the one in the section “Suggestions of BLAST users”. Two possibilities will be implemented to provide the users the needed data. The first one saves all provenance data in a text file because some information is only required for one workflow or for one publication. The format of the file should be easy to parse in order to deal with the data by shell programs or by Excel macros, which can collect the required data that is too rarely used for implementing them on the Provenance Query Interface. The other feature in the Provenance Query Interface is a kind of graph that shows the execution times of the modules and those failed. A user can see the modules causing errors and those using most of the resources. Beyond, it should be possible to get also an overview over all runs of a farming execution. Here it would be beneficial to see all execution times of workflows and which of them did not finish correctly. Using this additional application makes it possible to find all provenance files with errors, identify them,

and select the provenance file to analyse all details causing the error during execution and analyse it.

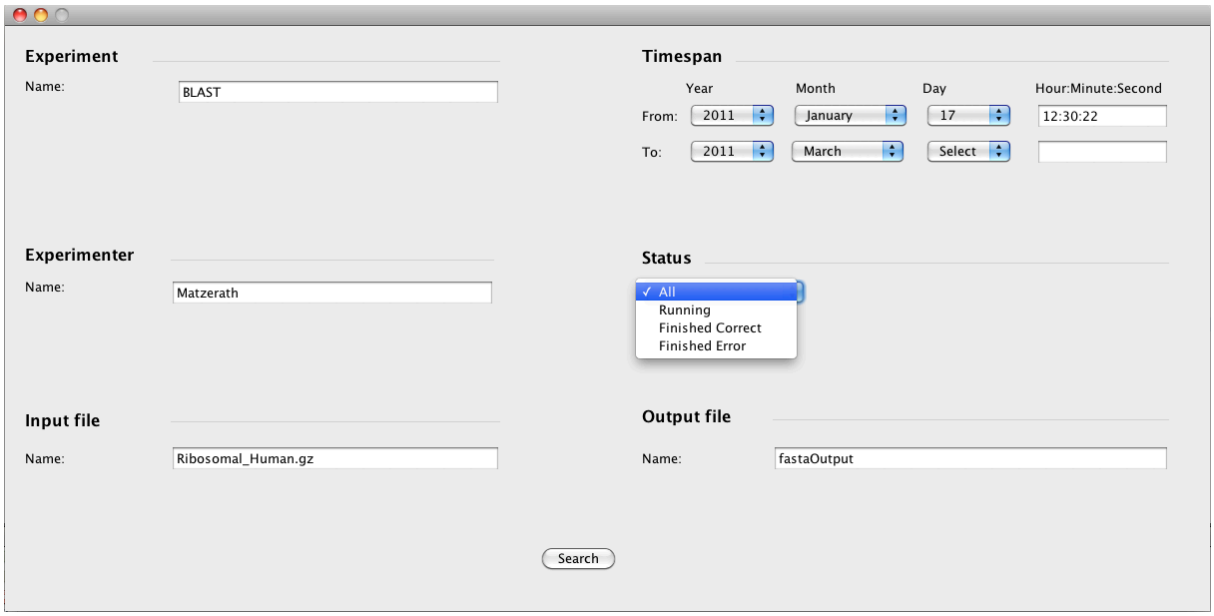
To summing up, one can say that the Provenance Query Interface is a tool to browse and read the provenance files. The user can see the complete course of action of the workflow and analyse it in detail. The architecture of the interface was made in close collaboration with the users and many of the features were added by request of them.

5.2 Provenance Query Interface Implementation

This section describes the design and the functionality of the Provenance Query Interface. It will be described how the interface looks like and how to work with it.

When the program is started, a window appears that allows the user to select a set of provenance files by categories. Figure 17 shows the parameters, which can be chosen to narrow the collection of provenance files down. The user can make settings like the workflow name, a time span during which the workflow was executed, the name of the executer, the status of the workflow, the input file name and the output file name. It is feasible to fill out any field but it is not mandatory.

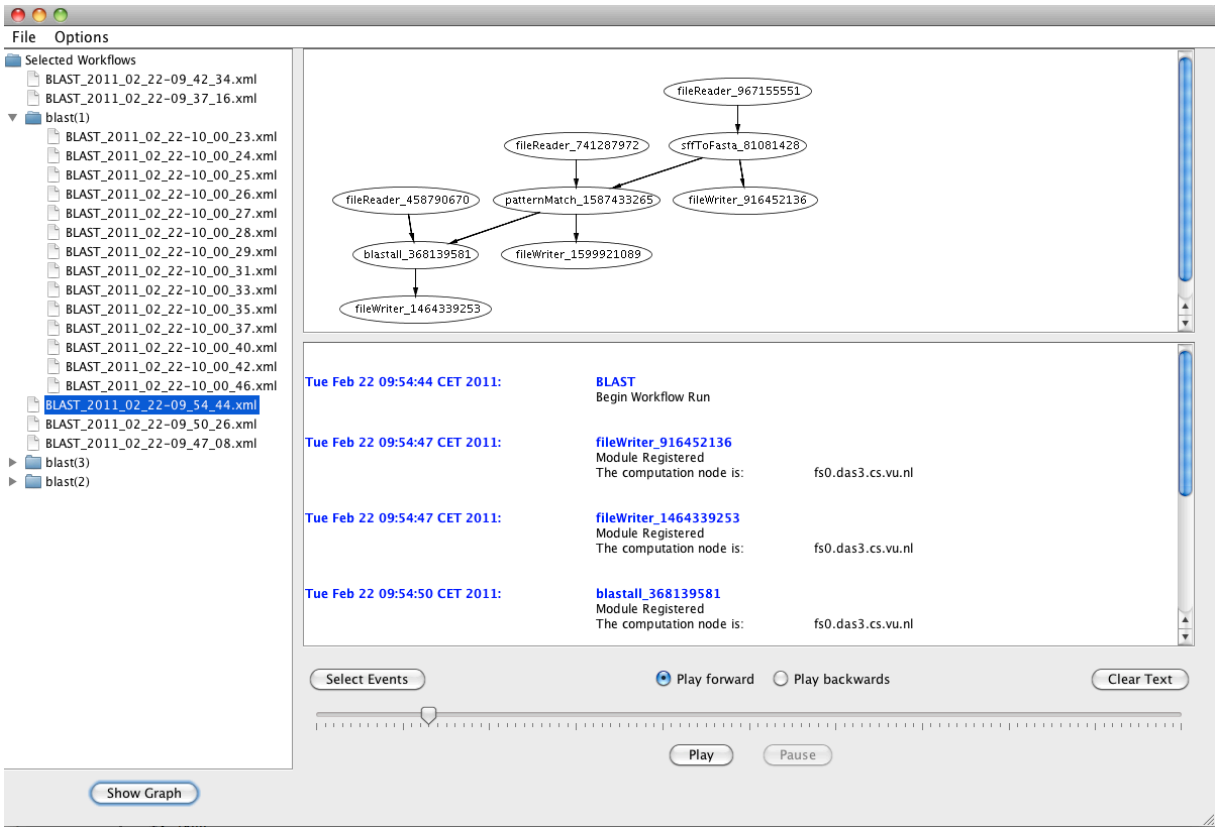
Figure 17: Starting window of Provenance Query Interface



Pressing the *Search* button starts a search algorithm running through the catalogues and checking if a provenance file fulfils the settings made in the interface. All provenance files achieving these requirements will be mentioned and listed at the left side in the tree view of the window that appears. Besides the listing of every single workflow, folders containing runs of one farming workflow are also shown. The aggregation of the provenance files of one

farming run is meaningful as they have a common context and they belong together to one run with different parameters. The belonging to one farmed run can be ascertained using the catalogue file because every instance of the farmed run gets the same farming ID.

Figure 18: Main window of Provenance Query Interface

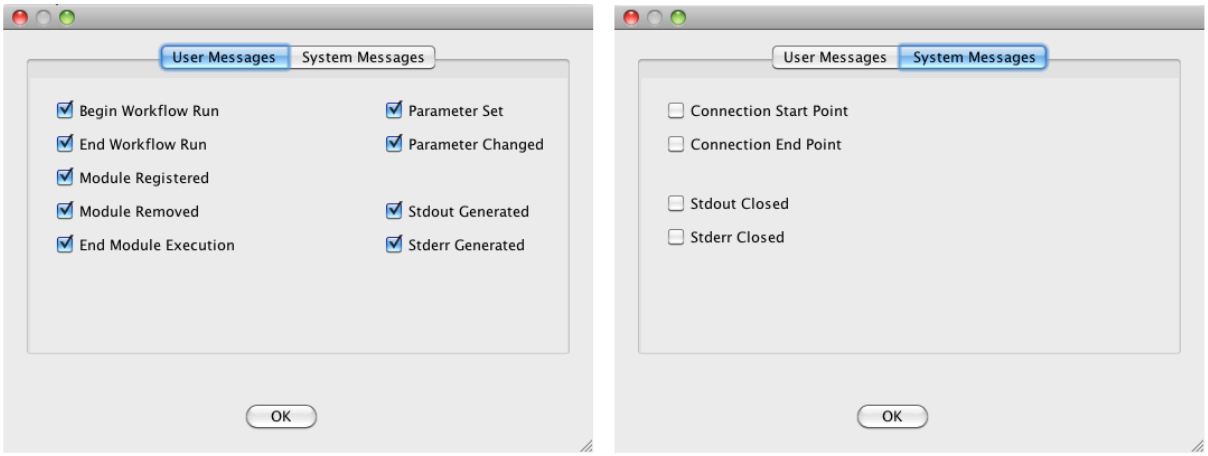


After pressing on a single provenance file a graph appears in the area on the right top of the window. This graph contains the executed modules of the workflow and is drawn with the *Graphviz dot* program that draws circles as well as lines and sets the positions automatically if the predecessors and successors are given. {2}

The buttons on the bottom of the window are also changing by selecting a provenance file. They get enabled and allow the user to start the replay that visualizes the provenance data. Clicking the *Play* button starts moving the time slider and the events are printed in chronological order on the text area on the right bottom of the window. All events are printed in the same format with the time and the module name in the heading making it easier to see when a new event appears. The other content of the event and all other values saved in the provenance file are printed below the heading. By using a menu entry the user has also the possibility to change the duration of the replay from sixty seconds to any value of his/her choice. In that case, the replay mechanism maintains the time spans between the events but compresses it to the replay length that was set.

The remaining buttons have the following functionality. The *Pause* button stops the replay process and can be continued later with the *Play* button. The *Clear* button deletes all printed events in the text area. The radio buttons *Play forwards* and *Play backwards* allow running the replay from beginning to end or vice versa. This option allows configuring the Provenance Query Interface for the needs of every user. Additionally, the *Select Messages* button opens a window allowing the selection of the messages, saved in the provenance file, which should be printed in the text area. The selection of the messages is divided into two tabs with the first tab containing all messages, which are important for end users analysing the output and the process of the workflow. These messages are selected as default because users working on those topics will mainly use the interface. The second tab contains messages, which could be more interesting for workflow or application developers and these messages are not selected as default. Figure 19 shows which messages can be selected in the two tabs.

Figure 19: Message selection window of the Provenance Query Interface



Below the tree view showing the provenance files is the *Show Graph* button that shows a diagram with different content depending on the selection in the tree view. If a single workflow is selected in the tree view, the diagram shows the starting and ending times of every module. The green bar shows the time span between start and execution finish, the blue one shows the time between execution finish and destroying the module. If a bar is printed in red colour the module was cancelled and did not finish properly. Generally, the time scale of the diagram is in seconds and the start of the workflow has the value zero and the highest value is the last measured time in the provenance file. If the user selects a farmed run of a workflow in the tree view and clicks the *Show Graph* button the diagram contains the starting and ending times of all workflows of the farming run. Here, every bar stands for a workflow and not for a module like in the other kind of diagram. In this graph, the green colour shows the starting and ending times of a correctly finished workflow and the red colour the execution times of a

workflow with errors. The diagrams will help to see at one sight which of the workflows or modules finished with errors and how long they need. The figures show the two different graphs.

Figure 20: Graph with times of modules

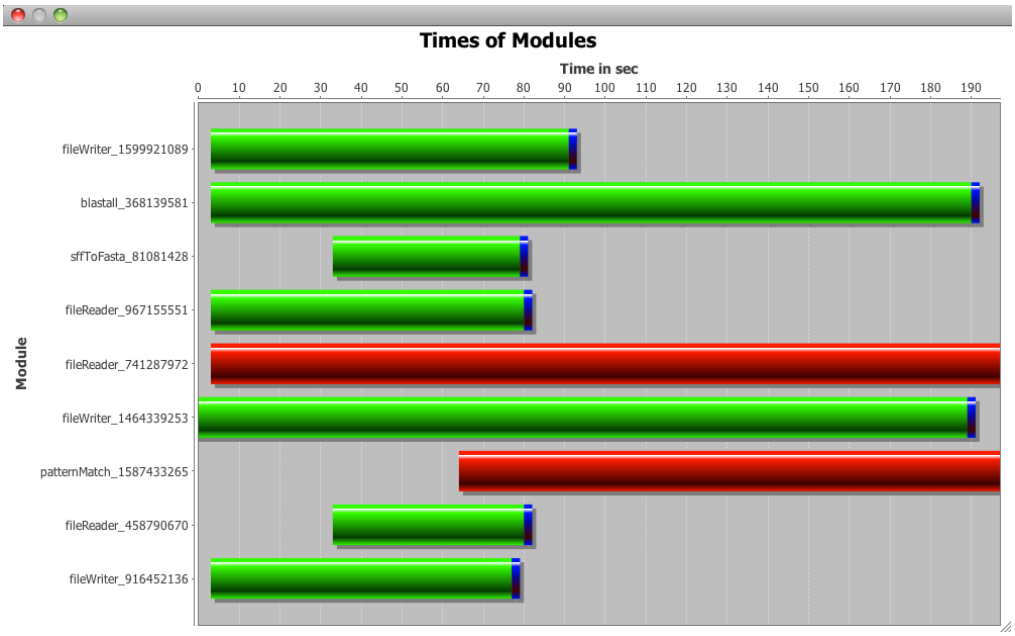
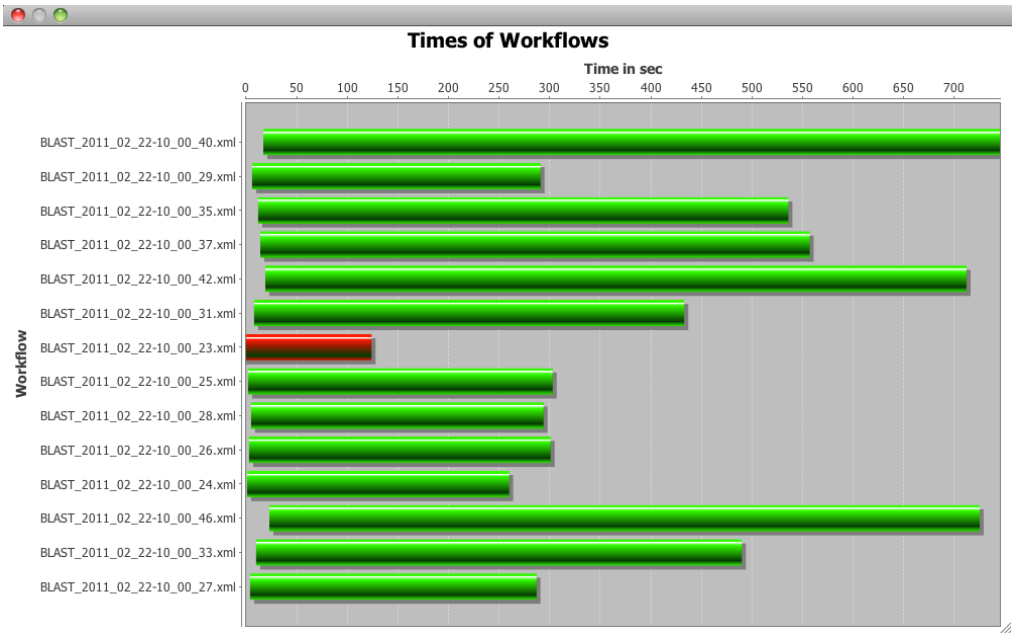


Figure 21: Graph with times of farmed run



The last options on the interface are located in the *Options* menu with three different items. The first is *Delete All Files,* which removes all files shown on the Provenance Query Interface from the hard disk and all references of those files from the catalogues. The second menu item is *Set Replay Length* opening a text input field to set the new replay length in seconds. Finally, the menu item *Export Output To File* transforms the events saved in the provenance

file to a text format. This format contains every event in a single line and separates the information time stamp, module name, event description and all other values by a tabulator so that it is easy to parse the text or insert it in Excel.

Figure 22: Menu of Provenance Query Interface

To sum up, the Provenance Query Interface shows a set of provenance files, which can be defined by setting parameters and helps to comprehend the temporal activity with a replay mode. Some additional features were implemented like the diagrams or the transformation to a text file to fulfil the users wishes and give them a tool to collect the needed information more easily.

6 The Open Provenance Model

The goal of this chapter is to explain and analyse the Open Provenance Model (OPM). After giving a short introduction of the background of OPM, the specification of the model will be explained in detail. The chapter closes with a view on the XML representation of the OPM.

6.1 Introduction

As stated above, provenance is one of the best approaches to analyse results and the adjustment of errors generated by scientific workflows. Workshops were initiated to discuss the chances and problems of provenance for workflow management systems. The *International Provenance and Annotation Workshop* (IPAW ’06) is one workshop in the abundance of workshops in the area of provenance. The main topic of the meetings was to find a standardization of provenance. Since all participating teams introduced different provenance systems and because it was not easy to find out the pros and cons of all systems, the responsible persons decided to initiate a *Provenance challenge* to compare the different systems. Up to twenty teams participated in the standardization of a provenance system during the four challenges between 2006 and 2010.

The four challenges focused on different areas of the provenance collection. The first challenge was a general workshop to understand every single concept and which data is necessary in the term of provenance. The second challenge focused on the interoperability of the data between the workflow management systems. As a result of this challenge, the first specification of the Open Provenance Model (OPM v1.00) was defined. The third challenge was held to test the provenance specification practically with a given workflow and the fourth challenge brought all results, collected in the previous challenges, together. It specified all parts of the provenance from understanding the provenance, over problems of inter-operability to testing the OPM solution.

In the meantime two enhancements of the OPM v1.00 specification were developed with the specifications OPM v1.01 and OPM v1.1. These specifications of the Open Provenance Model are very abstract definitions that point out which kind of data and types of causal dependencies can be stored for provenance. However, it is not defined in which format the data should be stored or how the workflows should provide the data. Hence, the Open Provenance

Model does not provide any implementation or protocols, which can be used to save provenance data. [17]

In order to collect provenance data, following the Open Provenance Model, scientists of the University of Amsterdam implemented the specification for the workflow management system WS-VLAM. Originally this implementation was supposed to be used to compare it with the history tracing system but it was not possible to run the program and collect provenance data, which can be compared to the output of the history tracing system. For that reason the comparison will be on a theoretical layer. However, the procedure and the results will be discussed in more depth in the section "Comparison of OPM and history tracing system".

6.2 The Open Provenance Model specification

This section describes the content of the Open Provenance Model specification version 1.1. [13] The different definitions and elements of OPM will be described separately with different headings.

Requirements and Non-Requirements

First, the OPM specification provided as well as specially not provides will be depicted. The authors established a list of requirements that the new specification should fulfil.

1. The OPM has to allow the exchange of provenance information between different implementations of the provenance system following the specification. Therefore a compatibility layer is inserted to the model.
2. It also supports the possibility to share tools operating with OPM data of other people to advance the spread of the OPM idea.
3. Another very important requirement that will be answered is that the paper gives a very precise and technology-agnostic definition of provenance so that everybody has the same background on provenance.
4. In the qualification it is also mentioned that a digital representation of provenance for any "thing", whether produced by computer systems or not, will be defined.
5. The specification also allows different levels of lineage paths to co-exist.
6. The last requirement of OPM is that a core set of rules has to exist that allows making inferences on the provenance representation.

Thus, there are also some non-requirements that will not be answered by the specification and have to be replied by every developer himself. These non-requirements are explicitly men-

tioned because the authors want to highlight that OPM does not contain technical restrictions and is inter-operable.

1. The first non-requirement includes that it is not the purpose to define the internal representations of the provenance data and how to store and manipulate it.
2. This also means that the developer can decide for example if he uses XML, databases or any other storage systems to save the provenance data. Thus, other documents exist that define realisations of OPM in XML, RDF or other types.
3. The specification does not define protocols saying how to store the provenance data in repositories.
4. The techniques of querying provenance information will not be answered by the specification, either.

Nodes

As Provenance always is a state of all described objects in the past with the main focus on what caused the objects or "things" to be as they are, OPM consists of directed graphs, which show those dependencies. The following part of the specification discusses the main components of the graphs and how they help to visualize the lineage data.

One component is the nodes that are representing material objects like digital data or cars and immaterial objects like decisions. The nodes are describing the "things" at a special point of time in the past and how the saved state came out to be. The following three different kinds of nodes exist in the OPM graph representation. ([13], pp. 2-3)

- **Artifact**
  Immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system.
- **Process**
  Action or series of actions performed on or caused by artifacts, and resulting in new artifacts
- **Agent**
  Contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution.

Artifacts are always immutable objects that exist at a special point of time. Processes can change the artifacts and as a result a new artifact is created. The processes can be observed by agents, which can for example be persons or computing nodes that control the activity of the processes.

Since OPM does not only define which data should be collected but also how it should be visualized different icons for the three nodes are defined, depicted in the following figure.

Figure 23: Illustration of Artifact, Process and Agent ([13], p. 5)

Causal Relationships

In order to constitute dependencies between the processes, the specification defines the directed graph with the nodes artifact, process and agent and the following five edges. ([13], p. 4)

Artifact Used by a Process: In a graph, connecting a process to an artifact by a used edge is intended to indicate that the process required the availability of the artifact to complete its execution. When several artifacts are connected to a same process by multiple used edges, all of them were required for the process to complete.

Artifacts Generated by Processes: In a graph, connecting an artifact to a process by an edge wasGeneratedBy is intended to mean that the process was required to initiate its execution for the artifact to be generated. When several artifacts are connected to a same process by multiple wasGeneratedBy edges, the process had to have begun, for all of them to be generated.

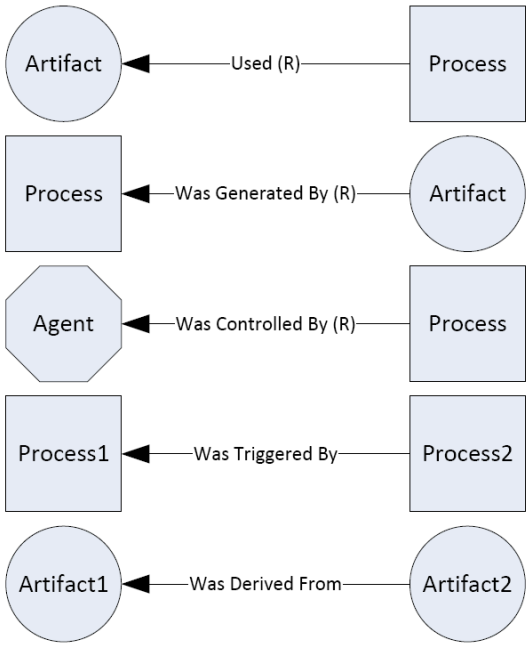
Process Controlled by Agent: The assertion of an edge "was controlled by" between a process P and an agent Ag indicates that a start and end of process P was controlled by agent Ag.

Process Triggered by Process: A connection of a process P2 to a process P1 by a "was triggered by" edge indicates that the start of process P1 was required for P2 to be able to complete.

Artifact Derived from Artifact: An edge "was derived from" between two artifacts A1 and A2 indicates that artifact A1 may have been used by a process that derived A2.

As one can see, every of the five defined edges represent another combination of the three nodes. The following picture summarizes the different edges and the involved edges.

Figure 24: Edges defined in OPM ([13], p. 5)



Summing up, one can define causal relationships, to which edges belong, as follows: ([13], p. 6)

Causal Relationship: A causal relationship is represented by an arc and denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause).

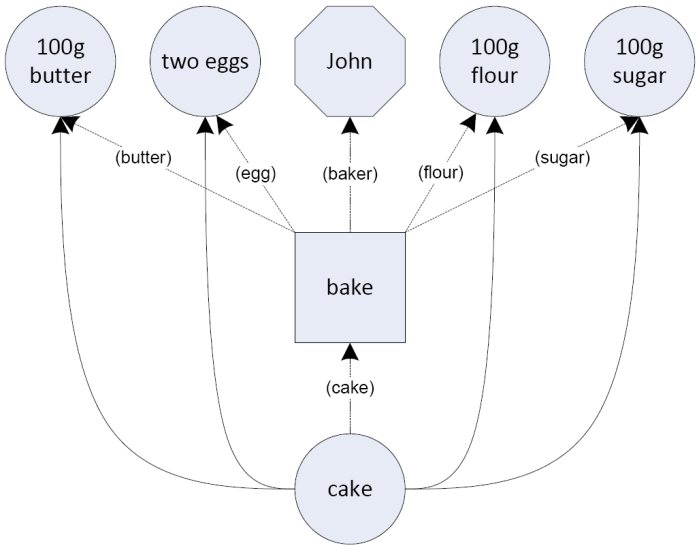
Roles

The edges *Used*, *Was Generated By* and *Was Controlled By*, shown in Figure 24, additionally have denoted by the letter *R*. Roles can be necessary and helpful if several artifacts are used by a process or different processes are controlled by one agent. Different roles help to differentiate the edges and make the allocation of the edges to the nodes possible. An example is a division using two numbers with the roles dividend and divisor, and producing two other numbers with the roles quotient and rest. The roles have just the task to help the user to differentiate the artifacts and allocate the edges and the meaning.

The two edges W*as Triggered By* and *Was Derived From* can be used to show the OPM graph in a data view or in a process-oriented view. The *Was Triggered By* edge creates a process-oriented view as artifacts between two processes are faded out in the graph and a more detailed view on the processes that took place is possible. If the processes are hidden as in the *Was Derived From* edge a data view is shown in the lineage graph. Here, a more detailed look on the generated objects is provided and it helps to analyse the results.

The following example depicts an OPM graph with all defined nodes and most causal relationships. The example shows that *John* is baking a cake with the ingredients butter, eggs, flour and sugar. Since the process bake has several *Used* edges they all contain a role to differ them more easily.

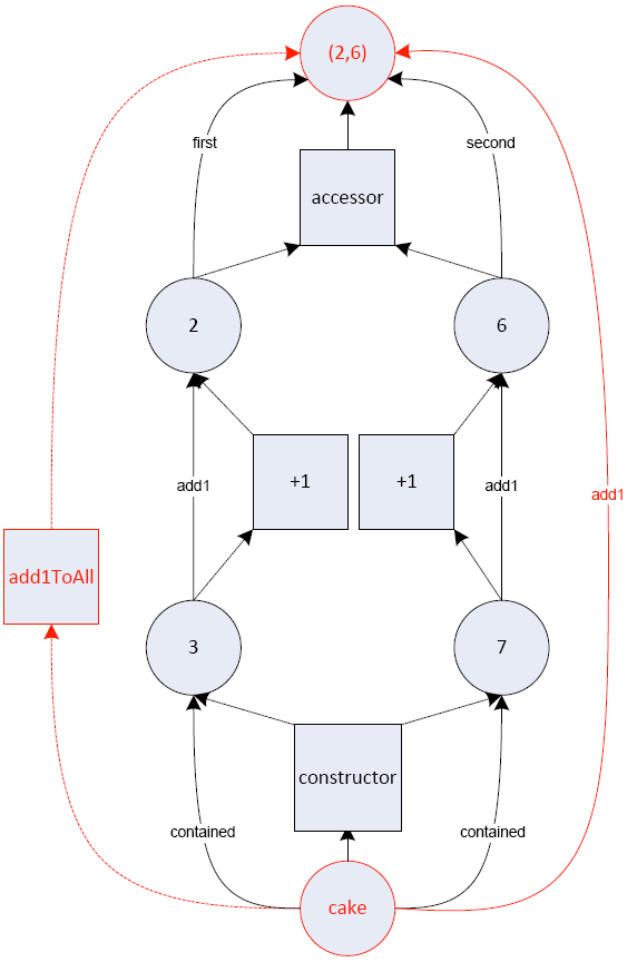
Figure 25: Example of OPM graph with baking process ([13], p. 10)



Overlappings and Hierarchical Descriptions

An enhancement of the previously described graph is *overlappings* and *hierarchical descriptions* of the OPM data. An example, which points out the appliance of the hierarchical graphs, is adding one to all values of a list with the values (2,6). The OPM graph shows how the resulting list (3,7) was generated. The following figure shows the process in an OPM graph in two different levels of details. The two sub-graphs are called *overlapping accounts* because they share the same common nodes. As the darker graph contains more details than the red one it is a refinement of the red sub-graph. A problem that might occur when overlapping graphs are used is that cycles are generated. Then it is necessary to insert *Was Derived From* edges to accent that the different sub-graphs show the workflow in different refinements.

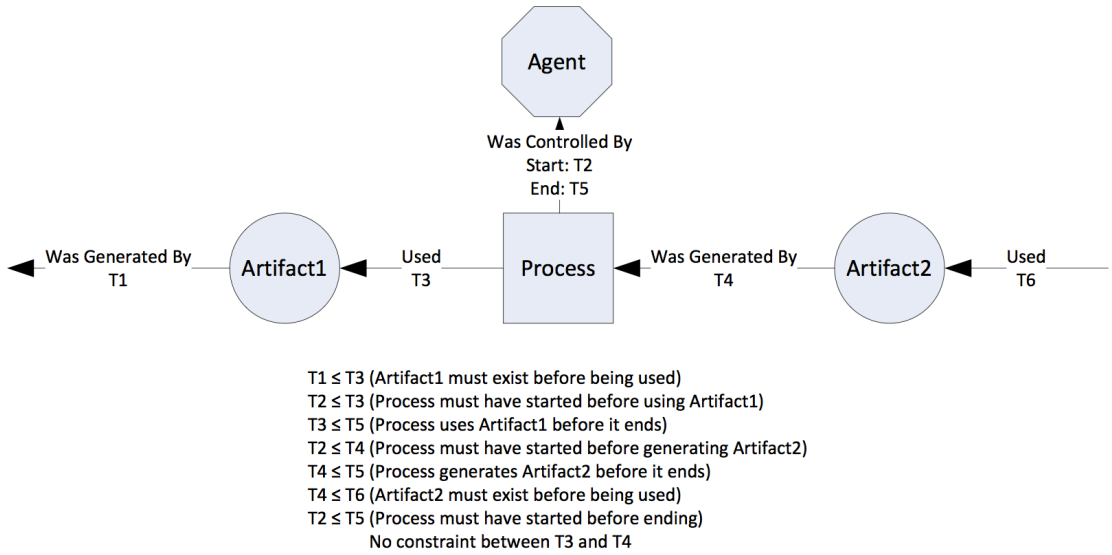
Figure 26: OPM graph with overlapping sub-graphs ([13], p. 12)



Times

Times can also be added to the provenance data and to the graphs. An important aspect is that times in OPM are not influencing the causality of modules. The edges are generated only by the five defined causal relationships. If synchronized clocks are used for all provenance data, the time of the cause has to be greater than the origin of it. Time can be used for validating causality claims in the case of a single clock, too. Four different times exist in OPM, at which *creation* and *use* times are utilized together with artifacts and the times *starting* and *ending* for processes. The following figure shows the usage of the different times and the dependence of the times that exist due to causal relationships.

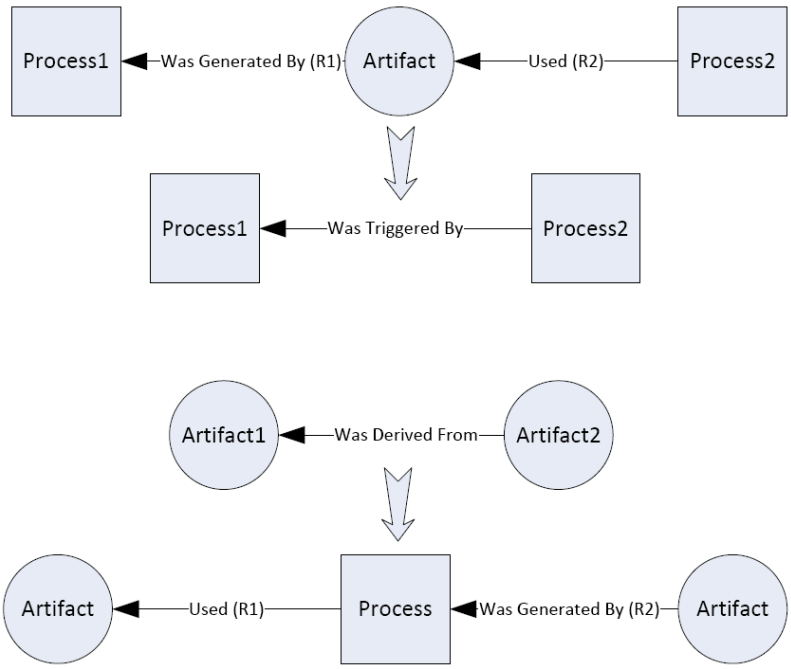
Figure 27: Time dependencies in OPM ([13], p. 16)



Completion and Multi-Step Edges

The two edges *Was Triggered By* and *Was Derived From* are so-called *completion rules*. The name is used as they reduce the OPM graphs by the artifacts or processes. Figure 28 shows the completion processes.

Figure 28: Completion in OPM ([13], p. 17)



If users need information about causes of artifacts, it is sometimes necessary to view not only the direct neighbours but also indirect causes with multiple transitions. To show edges that express these indirect causes four additional edges are defined: the *Multi-Step Was Derived From* and three kinds of the *Secondary Multi-Step* edges. ([13], p. 18)

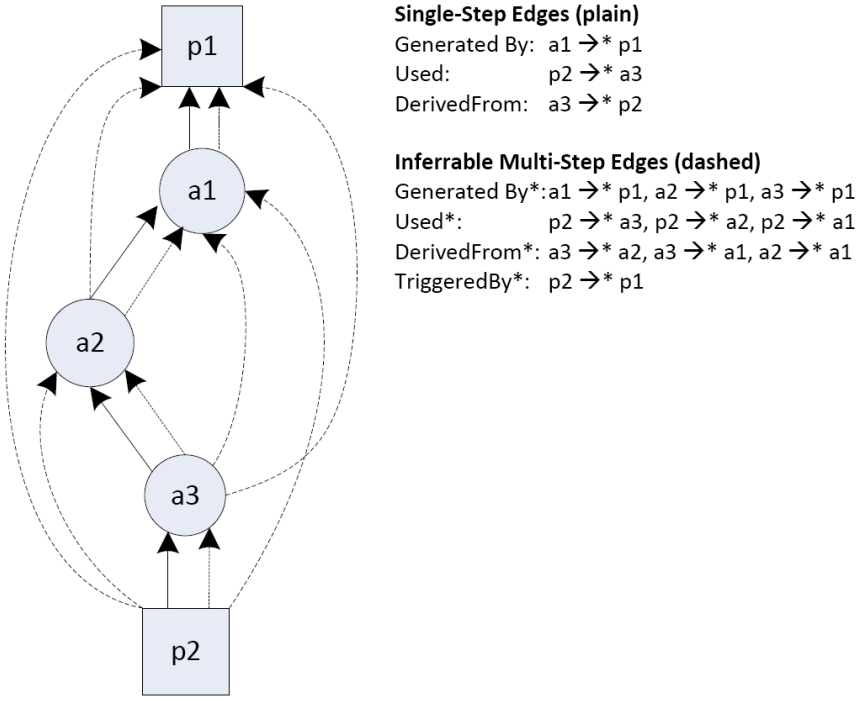
Mulit-Step WasDerivedFrom: An artifact a1 was derived from a2 (possibly using multiple steps), written as a1 $\rightarrow^*$ a2, if a1 *Was Derived From* an artifact that was a2 or that was itself derived from a2 (possibly using multiple steps). In other words, it is the transitive closure of the edge *Was Derived From*. It expresses that artifact a2 had an influence on artifact a1.

Secondary Multi-Step Edges:

- Process *p* used artifact *a* (possibly using multiple steps), written p $\rightarrow^*$ a, if *p* used an artifact that was *a* or *Was Derived From* a (possibly using multiple steps).
- Artifact *a* was generated by process *p* (possibly using multiple steps), written a $\rightarrow^*$ p, if a was an artifact or was derived from an artifact (possibly using multiple steps) that *Was Generated By p*.
- Process *p1 Was Triggered By* process *p2* (possibly using multiple steps), written p1 $\rightarrow^*$ p2, if *p1* used an artifact that was generated or *Was Derived From* an artifact (possibly using multiple steps) that was itself generated by *p1*.

"The four relationships, and associated inferences, are illustrated in Figure 29. In this figure, plain edges represent single-step dependencies, whereas dashed edges represent multi-step dependencies. For instance, from p2 → a3 → a2 we can infer p2 $\rightarrow^*$ a3 $\rightarrow^*$ a2 and p2 $\rightarrow^*$ a2, by "eliminating" a3." ([13], p.18)

Figure 29: Example of Multi-Step edges in OPM ([13], p. 19)



Causality Graph Data Model

Finally, all previously defined components of the OPM will be given in a causality graph data model without the time-set. The figure shows which components exist and how they are generated.

$$
\begin{aligned}
ProcessId &: \text{primitive set} && (\text{Process Identifiers})\\
ArtifactId &: \text{primitive set} && (\text{Artifact Identifiers})\\
AgentId &: \text{primitive set} && (\text{Agent Identifiers})\\
Role &: \text{primitive set} && (\text{Roles})\\
Account &: \text{primitive set} && (\text{Accounts})\\
Value &: \text{application specific set} && (\text{Values})\\
Process &= ProcessId \longrightarrow Value \times \mathbb{P}(Account)\\
Artifact &= ArtifactId \longrightarrow Value \times \mathbb{P}(Account)\\
Agent &= AgentId \longrightarrow Value \times \mathbb{P}(Account)\\
Used &= ProcessId \times Role \times ArtifactId \times \mathbb{P}(Account)\\
WasGeneratedBy &= ArtifactId \times Role \times ProcessId \times \mathbb{P}(Account)\\
WasTriggeredBy &= ProcessId \times ProcessId \times \mathbb{P}(Account)\\
WasDerivedFrom &= ArtifactId \times ArtifactId \times Value \times \mathbb{P}(Account)\\
WasControlledBy &= ProcessId \times Role \times AgentId \times \mathbb{P}(Account)\\
Overlaps &= Account \times Account\\
Refines &= Account \times Account\\
OPMGraph &= Artifact \times Process \times Agent \times \mathbb{P}(Used) \times\\
&\quad \mathbb{P}(WasGeneratedBy) \times \mathbb{P}(WasTriggeredBy) \times\\
&\quad \mathbb{P}(WasDerivedFrom) \times \mathbb{P}(WasControlledBy) \times\\
&\quad \mathbb{P}(Overlaps) \times \mathbb{P}(Refines)
\end{aligned}
$$

([18], p. 16)

6.3 The Open Provenance Model XML schema

Although no XML representation is described in the Open Provenance Model specification, the OPM community developed a XML schema. This schema is available on the official *Open Provenance Model* website and contains the schema itself and Java libraries which handle XML provenance files. OPMX defines which types for entities exist and how they are defined. This chapter will explain the content of a XML file based on the OPMX schema. [17] [18]

One of the goals of the schema is to define a XML serialization that can be converted into RDF format and vice-versa. RDF stands for *Resource Description Framework* and is a family of the World Wide Web Consortium (W3C) specifications and helps to define directed graphs. The data can be serialized by the XML format, databases or other formats. [19]

All nodes and edges, described in the previous paragraph, are defined in this schema and the information that can be saved in addition to the elements is also defined. To describe the XML content more in detail the following sample is given.

Figure 30: Example of OPM graph

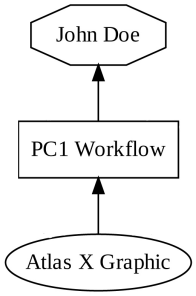


Figure 31: XML representation of OPM provenance file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opm:opmGraph xmlns:opm="http://openprovenance.org/model/v1.1.a"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://example.com/" id="gr_114">
  <opm:accounts>
    <opm:account id="black"/>
  </opm:accounts>
  <opm:processes>
    <opm:process id="p1">
      <opm:account ref="black"/>
      <opm:label value="PC1 Workflow"/>
    </opm:process>
  </opm:processes>
  <opm:artifacts>
    <opm:artifact id="a5">
      <opm:account ref="black"/>
      <opm:label value="Atlas X Graphic"/>
    </opm:artifact>
  </opm:artifacts>
  <opm:agents>
    <opm:agent id="ag1">
      <opm:account ref="black"/>
      <opm:label value="John Doe"/>
    </opm:agent>
  </opm:agents>
  <opm:causalDependencies>
    <opm:wasGeneratedBy id="g_107">
      <opm:effect ref="a5"/>
      <opm:role id="r_106" value="x"/>
      <opm:cause ref="p1"/>
      <opm:account ref="black"/>
    </opm:wasGeneratedBy>
    <opm:wasControlledBy id="c_113">
      <opm:effect ref="p1"/>
      <opm:role id="r_112" value="user"/>
      <opm:cause ref="ag1"/>
      <opm:account ref="black"/>
    </opm:wasControlledBy>
  </opm:causalDependencies>
</opm:opmGraph>
```

The XML serialization shown in Figure 31 represents the graph in Figure 30. The three different nodes *process*, *artifact* and *agent* are stored in different tags to delimit the three different kinds of nodes. They all contain three values: the unique ID, the account they belong to and the label, which will be presented in the graph. The account is also a separate tag, which contains all accounts that exist in the provenance file. The last category that can be found in an OPM file is the tag *causalDependencies*. This contains XML nodes that represent the five different edges. In this example the connections *wasGeneratedBy* and *wasControlledBy* exist and embody information like beginning and ending node of a connection and ID.

7 Comparison of OPM and history tracing system

In the course of the thesis the discussion of the term of provenance and which data have to be collected in a lineage file, the description how the adaptation of the history tracing system was accomplished and the definition of the Open Provenance Model were performed. This chapter points out the differences between the Open Provenance Model and the History-tracing XML-based Provenance Framework for Workflows with the achieved knowledge about the two models. The main focus of the comparison will be laid on the technical requirements, which were discussed and if the models can answer them. The sub-sections will describe the concepts of the provenance systems and the adaptation to WS-VLAM as well as the adjustment to other WfMS. Furthermore, accomplished and future adaptations of the history tracing system learned by OPM will be discussed. The chapter closes with a conclusion of the comparisons. At the end of three sections tables are added to summarise the conclusions and contrast them.

7.1 Approach of comparison

As stated in the "Introduction", one of the main goals of this thesis is to compare the two provenance systems especially in conjunction with the workflow management system WS-VLAM. As the comparison has to be laid on a common basis both systems were adapted to the workflow system WS-VLAM. This has the advantage that all provenance data generated by the workflow management system and used as input for the provenance systems is the same. After running the two systems with WS-VLAM workflows the provenance data can be compared and advantages and disadvantages can be highlighted.

This was the plan to accomplish the comparison as effectively as possible with most benefit for the developers and users of the workflow management system and the provenance systems. A comparison like that is very close to the users' needs and can be discussed with them very well because it is based on real provenance output. Theoretical comparisons often cannot cover all eventualities because the systems are too complex.

The connection of the history tracing system with WS-VLAM was no problem and provenance data could be collected using the re-implementation of the history tracing system. In order to integrate OPM with WS-VLAM the implementation of the University of Amsterdam should be used, as this implementation already exists. However, it was not possible to collect

provenance values with that OPM implementation and it was not possible to fix the problems during this thesis.
Nevertheless, there will be a comparison of the two systems with a shifted focus from the practical to a theoretical comparison of the two provenance systems. Examples of topics that will be compared are the concepts, the security or the possibilities to expand the systems. The fact that a comparison on a theoretical layer is not detailed enough is only conditionally true. The knowledge about the underlying WfMS and the re-implementation of the history tracing system for WS-VLAM weakens the disadvantage of a theoretical comparison.

7.2 Comparison of concepts

This section will point out the differences between the Open Provenance Model and the history-tracing XML-based Provenance Framework for Workflows. The focus of the comparison is laid on an abstract level and decoupled from any workflow management system. The comparison will be structured in sections that are oriented on the technical requirements of the chapter “Use cases and technical requirments”. Every part discusses if the provenance systems answer the technical requirements and the way in which they do it.
The goal of the two systems is the same as they are running during the execution of a workflow to record how a specific status at a certain point of time was created. The provenance systems can save the states of variables and parameters of a workflow, and allow tracing back to the origin of the results. Thus, there are differences in the way they collect the data, which will be discussed in this chapter. The specification of the Open Provenance Model is an abstract definition that appoints the kinds of data and types of causal dependencies that can be stored. It does not state in which format the data should be stored or how programs can access the data of the workflow. Hence, the Open Provenance Model does not provide any implementation or protocols in the core specification, which can be used to store provenance data. In contrast, the history tracing system is a ready-implemented framework, which can be used to store the provenance data of arbitrary workflow engines. For the comparison of the two systems, the consideration of the history tracing system will be brought to the same abstract level as the OPM specification.

Different Views on Provenance Data

As previously described, different views on provenance data are a very important technique to customize the provenance interface and show only the interesting data to every user group.

However, not only the interfaces are important to visualize different views but also the data collecting and storing part of the provenance systems. The differences in saving and visualizing provenance data for different views will now be described.

OPM contains three different nodes, which represent a state of a thing (artifact), an action performing an artifact (process) and an entity that controls a process (agent). Besides the nodes different edges consist, which show the dependencies of the nodes. Using the *Completion and Multi-Step Edges* described in the chapter "The Open Provenance Model specification", it is additionally possible to show the graph in a process or data oriented view, interesting for different user groups. The dataflow-oriented view consists mainly of artifacts and does not visualize the processes that are in between the artifacts. In opposite to that the process-oriented view shows only the processes and hides the artifacts that are generated or used. This can be interesting for users that have a different background and want to analyse other aspects of a workflow.

In opposite to OPM, in the history tracing system exists only one kind of node that stores all provenance information of the workflow, and is called *sequence* in the history tracing system. The other nodes that exist represent only different workflow actions like fork, join or decision and define the course of the graph. However, the single task contains information that can be allocated to the OPM nodes artifact and process. The stored changes of a workflow module like parameters or input and output values are an artifact combined with processes that change the modules. This means that a node of history tracing does not only contain the state of a "thing" at a special point of time, but all input as well as output values. It also contains the parameters with all changes during its lifetime. Most times the agent can also be found in history tracing system since it saves the executer of a module and the computing node. Thus, an agent can be built by different information saved in the history tracing system. Although the history tracing system does not describe different views in the concept, it is possible to provide the data that is necessary to show the graph in the two views. To show different views an additional logic has to analyse the collected data and allocate it to the three nodes.

Knowing that history tracing saves all data that is necessary to differ between artifacts, processes and agents it is obvious that the five different edges can be shown in the history tracing system, too. Summing up, one can say that both systems save all data necessary for the creation of artifacts, processes and agents so that different views are easy to provide. However, OPM saves the data in the storage system making it easier for the provenance interface to visualize the data. For history tracing most of the complexity to show different views has to be implemented in the provenance interface. Moreover, the Open Provenance Model orders

data belonging to one of the three categories artifact, process and agent whereas the history tracing system sorts the modules according to their forerunner and follower.

Reproducibility in Workflows and Exchange Provenance Information

Reproducibility is one of the most important aspects in provenance. This means that the provenance system should allow using the provenance data to re-run the workflow. In the area of the concepts of the provenance systems are only one part that defines the usability of reproducibility. The concepts can only define which provenance data will be saved. Both systems save the most important parts of the reproducibility data so that reproducibility can be provided after the handling of the data limitations are especially given because of the data that cannot be provided by the WfMS.

The exchange of provenance information is different in the two systems. Since up to twenty teams from institutions around the world participated in the composition of an OPM specification, it is much more accepted. Another important fact is that many of the participants already implemented OPM for their WfMS and thus the exchange of provenance data is possible. The history tracing system is not as widespread as OPM and other scientists cannot use the data and compare it with their provenance output of the WfMS. However, it is not too much work to adapt the history tracing system too other WfMS due to the web service architecture. More on the adaptation to other WfMS will be described in the chapter “Comparison of implementations”.

Security in the Provenance Systems

Some users of workflows need provenance systems that store the data in a legally binding way. To provide the reliability on the data the history tracing system contains a mechanism to sign the XML tags that contain the lineage data of a module. In opposite to that, OPM contains no security mechanism to save the data in a legally binding way. For OPM it is not easy to implement a mechanism to store the data reliable either, because the inter-operability is not assured and an exchange of provenance to other scientists is not possible anymore. The existence or absence of the security mechanisms can be a reason to decide for or against the provenance systems.

Other Technical Requirements

The technical requirements asking for metadata and references between input and output data as well as the search of data in different workflow management systems will not be described

in this section because the accountability is at the WfMS and not in the concepts of the provenance systems.

Other Differences

A similarity is that both systems are able to store the execution times of any action happening in the workflow. The distinction between the two methods is that OPM stores the times only optionally, whereas they are stored mandatorily in history tracing. Thus, in both concepts the dependencies determined due to workflow addictions are more important than the dependencies based on time. The reason is that the WfMS only relies on defined causal dependencies and time does not matter for the execution order.

A difference in the data types that are supported by the two systems also exists. History tracing defines XML explicitly as the format to be supported. In opposite to that OPM says nothing definite about data types. It is just mentioned that other documents exist telling how the provenance data should be stored in XML and RDF. However, saving in databases is also possible. The reason for being so imprecise in OPM is that the specification does not want to tell anything about internal representations of OPM in a WfMS.

Summing up, one can say that the concepts differ especially in the way the data is sorted. The history tracing system stores module-oriented and OPM saves category-oriented. Thus, both systems store almost the same data and the differences in the other categories are insignificant in the view of the collected data.

Table 13: Differences in the concepts of OPM and history tracing system

Category	OPM	History tracing system
Different nodes	Yes	Data are stored but have to be ascertained
Different views	Yes	Data are stored but have to be ascertained
Reproducibility	Main data are stored	Main data are stored
Non-Repudiation	No	Yes
Time of execution	Optionally	Mandatory
Datatypes	Not defined	XML

7.3 Comparison of XML provenance files

The comparison of the XML provenance file created by the two provenance systems will be based on the history tracing system schema framework and *The Open Provenance Model*

XML schema (OPMX). OPMX can be found on the official web site of OPM and describes a possible XML file for the provenance data storage. [20]

As described in the previous chapter, the history tracing system is based on a module-oriented storage and OPM on a category-based storage of provenance information. This fact also mirrors in the XML structure of the two systems.

The history tracing system uses a layered style and saves the course of the workflow modules in layers. This means that the dependencies are saved indirectly by the use of the layered structure and makes it easy and fast to walk through the xml file and find the predecessors and successors. In opposite to that OPM uses no layers to describe the course of the saved data. It saves the artifacts, processes and agents without dependencies so that the dependencies have to be saved separately. The dependencies are saved in a way that all have to be passed through to find all forerunners and followers of a node. The absence of the layered structure gets increasingly important for the provenance interface if the provenance files grow in complexity because the drawing of the provenance graph and the visualization of the data takes more and more time. On the other side, the storage in the three categories in OPM has the advantage that the interface can read the different nodes from the file and does not have to classify the data in one of the three categories. However, there is also an advantage of saving the data in the simple style since the time necessary for saving provenance data is decreasing due to a lowered complexity that limits the time for the workflow execution. But the difference should not be too big since the history tracing system inserts only one new tag as the root's child if a *sections* is inserted and only forks, joins or other complex paths could use slightly more time as they insert multiple tags.

Another advantage of the layered structure is that the possible paths of the workflow execution can be saved more easily in a schema file to find out if the workflow ran the right course. However, this possibility is also caused by the fact that a new schema file is created for every workflow saving the application flow of the workflow and help to validate the correctness of the predecessors and successors by the schema file.

Security can also be implemented more easily and effectively for the history tracing system in the layered style. If a layer is finished one knows that there will not be any changes in that element or any of its children and an inserted signature will not be falsified by new values. Inserting signatures is also possible in an Open Provenance Model XML file. Thus, for every artifact, agent, process and dependency a signature is necessary so that this action needs a much higher number of signatures and slows down the workflow execution, blows up the provenance file size and needs a long time for validation of the high number of signatures.

Furthermore, the signed OPM file will not be compatible to implementations of other WfMS and one of the central requirements, the inter-operability, gets lost.
Summing up, one can say that the adaptation of the layered style brought some advantages in the efficiency of storing and requiring the course of the dataflow and in the possibility of saving the data in a non-modifiable way.

Table 14: Differences in XML files of OPM and history tracing system

Category	OPM	History tracing system
Storage order	Stored in categories artifact, process, agent and dependency	Layered structure
Dependencies	Stored in extra category	Indirect by layered structure
Validation of provenance file	Complex because of variable order	Easy because of layered structure
Security	Every artifact, process, agent and decision needs own signature	One signature for every layer

7.4 Comparison of implementations

This chapter compares the adaptation of the two provenance systems based on three main aspects. First, the adjustment to WS-VLAM will be compared. Then the alignment user's needs and last to other WfMS. In some parts it is necessary to do the comparison on a theoretical layer because the OPM implementation for WS-VLAM could not be used to save provenance values, as it was not possible to get the OPM implementation running and collecting provenance values.
As OPM is only a specification, an implementation is necessary to collect provenance data of WS-VLAM. The history tracing system is also not originally developed for WS-VLAM and some adaptations have to be made, which are described in the chapter "History tracing system in WS-VLAM" in detail. In this section, the assertions for both systems will be pointed out and compared.

Adaptations of the Provenance Systems to WS-VLAM

As mentioned, implementations are necessary in order to collect data with both provenance systems from the WfMS WS-VLAM. The concrete adaptation s are very different and discussed in this section.
As the history tracing system is based on web services most parts of the server side can be taken over from the original system to WS-VLAM. On the server side changes are necessary that save the data of a workflow with multiple starting and ending points. This kind of work-

flow was not considered and a new structure of the layers in the XML file was implemented. However, this is a change that is non-recurring and can be used for all WfMS using the history tracing system in the future. Additionally, the history tracing system was enlarged to store more and other provenance data. Since WS-VLAM not only creates provenance data like input and output values of the modules, it is necessary to enlarge the system and collect all other data like the standard output or the paths of the input and output files. The last change on the server side is writing the schema file that has to be adapted to the new multiple starting and ending points and to the different structure of the XML workflow file. Since every WfMS has its own XML structure to describe the workflow file, it is necessary to adapt writing the schema and collecting the necessary data out of it. Besides, most changes are necessary on the client side. As WS-VLAM cannot use the mechanism of inserting additional tasks to the created workflow, the provenance collection has to be changed to an event-based mechanism. This causes the loss of the independence of the workflow management system.

As OPM is a specification and does not contain an implementation the adaptation to WS-VLAM has to be programmed. The OPM implementation of the University of Amsterdam for WS-VLAM also uses an event based mechanism to collect the provenance data of the WfMS. For OPM it is also necessary to implement the data storing part from the beginning with only a few libraries. The libraries and namespaces for Java are for example *The Open Provenance Model Java Library*, a Java library for creating Java representations of OPM graphs and serializing them to and from XML. [20] However, those are only some low level facilities that can be used for the own implementation and most of the work has to be done by the developers themselves. The system of the University of Amsterdam saved the provenance data in a database but they can be converted in XML files at any time.

For both provenance systems the event mechanism is almost the same and this means that none of the two systems has an advantage in the way they are collecting the provenance data. Due to the fact that they are both using this technique, both systems are dependent of WS-VLAM. OPM has a big advantage in opposite to the history tracing system, as it is a much older provenance system and developed by teams all over the world. A consequence of this is that no extensions have to be accomplished in the schema for multiple starting or ending points or other unconsidered workflow structures. The yearlong development of the specification with many use cases and working groups considers all possible workflow structures. However, the already finished process of writing a specification has also the disadvantage that additional provenance data is not mentioned in the specification result in a loss of the interoperability. Thus, the history tracing system will reach a state like the current status of OPM

with enough concepts and regulations so that it does not have to be extended, either. When the history tracing system is adapted to other workflow management systems and other use cases with other workflows are executed then the new awareness will help to reach a state that does not make changes necessary. At this stage the schema will be frozen and it will not be possible to extend the schema anymore so that the architecture allows interoperability between different workflow management systems using history tracing. One can say that the higher influence of the developers and users on the history tracing system is caused by the fact that it is a new implementation. And this influence can and should be used to develop a provenance system that has very good acceptance of the users.

During the development of the history tracing system one has to make sure that the different versions are compatible. Provenance files created by new developments should be readable by old ones and vice versa. Otherwise, it is problematic to read old provenance files created with older versions of history tracing system, which might be important over a long time span.

Summing up, it can be said that the biggest differences are in the re-use of the web service methods for history tracing system and the complete re-implementations of OPM. Thus, the XML files of history tracing had to be adapted for WS-VLAM and will probably be changed in future when new workflows are used.

Adaptations of the Provenance Systems to the Users Needs

The adaptation of the provenance systems to the users' needs again consists of two components. On the one hand the users define which data they need for their analysis and furthermore the data collecting part of the lineage system has to be adapted. On the other hand the presentation of the provenance data is an important aspect and the users remark requirements that a provenance interface should contain.

If users need new provenance data that is not saved in the lineage file the provenance system should be adaptable to collect the required data. However, the developers should always check if the data could not be ascertained indirectly by available information saved in the file. If the data is not available the developer has to ask the user if the data is only important for a special workflow or for an analysis that will not be repeated in the future. The reason is that saving all data that is available of a workflow run will extend the provenance file so much that it gets increasingly slow and the query interfaces get too complex. If data is used very rarely the users should think about saving the data in log files to query the data.

The adaptation to the users' needs is different for the two provenance systems. As OPM is described in a specification it is not possible to extend the system and collect new provenance

data since all other implementations of OPM are not compatible with the re-implementation of OPM. In WS-VLAM it is possible for the developers to extend the provenance system the system is in the development phase, important advancements and enhancements can be added. Here, the compatibility to older versions has to be considered.

The provenance interfaces of both systems are mainly developed to answer the requirements of the users. As no provenance interface existed for the original history tracing system a new Java based Provenance Query Interface was implemented. During the development much attention was paid on designing the interface in a way that the needs of the BLAST users are answered but other scientists can also use the provenance interface for their research. It is important to develop interfaces that can be used by as many user groups as possible because it is too much work and takes too much time to develop interfaces for every user group or even every workflow that is developed. The interface for the OPM model that was developed by the University of Amsterdam is too much oriented at the wishes of the users and it is hard to find other users being able to utilize it. Although, the OPM community developed many interfaces and most of them use the specification as a basis, it is hard to find an interface that can be used for the particular WfMS as the storage system has to be the same as the particular one and many of the interfaces are too special to re-use them.

Adaptations of the Provenance Systems to other Workflow Management Systems

In the course of this chapter it was mainly described which actions are necessary to combine the provenance systems with WS-VLAM. In the following it will be discussed which parts of the provenance systems can be used for other workflow management systems.

Because of the wide spread of OPM around the world and the usage in many well-known WfMSs like *Taverna, Kepler or VisTrails* it is not often necessary to do the adaptation by oneself. [20] However, if a WfMS does not contain OPM, all parts of the provenance collecting system have to be developed because the architectures are often not usable for other WfMS. The adaptation of the history tracing system is not as comprehensive as the adaptation of OPM since the web service can be used with minor changes. Changes are only necessary for the adaptation of the schema creation that has to be matched to the different XML workflow files. The section collecting the data has to be re-implemented to call the web services and pass the data to the provenance file.

For the history tracing system the Provenance Query Interface can be used without changes since it only reads the provenance files and those are the same for every WfMS. The provenance interfaces of OPM can also be used for other WfMS. However, it might be necessary to

implement a conversion from the storage system of the new OPM implementation to XML so that the system can access the data.

Table 15: Differences in implementations of OPM and history tracing system

Category	OPM	History tracing system
Independence of WfMS	No	For web server yes, For interface collecting events no
Re-usability	Only libraries	Web server part
Extensions necessary?	No	Yes, under development
Compatibility	Yes	Problematic because of continously development
Save data users require	No	Yes
Show data users require	Yes, if affected by available values	Yes
Exchange of provenance interfaces	If same storage system	Yes
Distribution	Used in many WfMS	WS-VLAM and JBPM

7.5 Consequences for History Tracing System learned from OPM

History tracing system is a concept that is in development and some aspects can be advanced to get a more comprehensive and accepted system. In this section, possible and realized enhancements of the history tracing system that are learned from OPM will be discussed. As some aspects take a long time to realize not all were implemented during this thesis. Moreover, some of them should be discussed with the users to learn what they think how the enhancements should be put into practice.

A suggestion that is organizational and done in OPM is the accomplishment of different challenges that are more informational as for OPM. For every challenge the main focus could be laid on a special aspect as the provenance interface, the graphs, the adaptation to the users' needs or the security. At the end of every challenge the pros and cons of the concepts will be discussed and the best concept will be implemented to enhance the history tracing system.

The most obvious aspect that was missing for the history tracing system was a provenance interface that visualizes the provenance data in a user-friendly manner. A first implementation of the Provenance Query Interface was developed to answer the requirements of the users. As OPM interfaces always show provenance data by graphs, a graph is also generated for the new Provenance Query Interface that represents the course of the executed modules. As the history tracing system stores the data according to the forerunners and followers the graph contains the modules that were executed and the graph looks like the originally created work-

flow template. This means that the graph contains not many additional information compared to the workflow file. The user can just see which modules were executed and which not.
As OPM is saving the provenance data in the three categories artifact, process and agent the provenance graph contains more information about the behaviour of the executed workflow. The user can see which parameters were used, the results of the artifacts or which agents controlled the processes. The information provides a detailed look on the provenance data and help to analyse the results. The graph generated by the history tracing system should be extended and elements of the OPM graph should be used to provide more information. The realisation of a graph that contains all OPM information is no problem as all data is saved in the XML file. One problem could be to collect the data fast and simple, as the data has to be completely re-ordered and for this a complex algorithm may be necessary. However, a first step could be to show all or selected information of a module by clicking on it or drawing the module red if it finished with errors.
Another interesting feature that could be used for the history tracing system is the possibility to create sub-graphs with more information or Multi-Step edges to show different views of the same workflow. These features are not only realizable if the graph is drawn in the view of the three nodes but also zooming in a graph that contains loops could show the loop unrolled. Another possibility is to select the messages that are shown if the user clicks on the modules on the graph.
The history tracing system contains no roles either, as they are described in OPM. Roles can be used if many modules are executed in a workflow and it gets confusing to follow the path. Furthermore, tasks are often using different parameters and roles help to allocate the values to the meaning. This also facilitates the understanding of the results more easily.
Summing up, one can say that most of the enhancements of the history tracing system that can be learned by OPM are in the area of graphs. The visualization in graphs was disregarded in the history tracing system in the past and some advancement is necessary. The storage in the XML files is very efficient and allows signing the data. OPM does not contain a better storage system so that changes are currently unnecessary.

7.6 Conclusion

The chapter pointed out the differences between the Open Provenance Model and the history tracing system. Both of the provenance systems have advantages and disadvantages in their

concepts or in the way they save provenance values in XML files. This paragraph summarizes the advantages and disadvantages of the two provenance systems.

First, OPM defines the nodes *artifact, process* and *agent* and five different causal relationships (edges), which exist due to diverse combinations of the tree nodes. The differentiation between different nodes and edges does not exist in the history tracing system. However, the different nodes help to adjust a Provenance Query Interface easier to the user's needs and point out different views like the data or the process view. As the history tracing system saves the data, which is necessary to differ nodes, the data can be collected in order to visualize them in a provenance interface.

The most important difference between the two systems is the XML structure. History tracing uses the layered style to represent the provenance data whereas OPM uses a structure separating the kinds of nodes and the connections. Indeed, OPM avoids the problem with multiple starting and ending points, but the disordered structure arouses the problem of an ineffective redrawing of graphs. This problem does not occur in history tracing because the forerunner and follower of the current child are always the parent and child of the current node in the XML file. The possibility of creating signatures of every module is another advantage in order to assure non-repudiation.

Another aspect, which has to be mentioned, is the fact that OPM is a specification and cannot be changed by the application developer to collect new provenance values. If the developers do not strictly follow the specification it is not assured that the data and tools are compatible with other implementations. Because history tracing is in development it is easier to adapt the system to the user's needs. This may cause the problem that different version are not compatible. However, the developers should expand the system as much as possible so that important values are saved until the system is used with other WfMS.

In conclusion, one can say that the history tracing system has some advantages in the way it saves the provenance values in XML. Anyhow, the Open Provenance Model has the huge advantage that many WfMS use this system to save the provenance values. In contrast to that, history tracing is only running together with WS-VLAM so that one of the most important aspects of provenance, the interoperability to other WfMS, cannot be guaranteed. OPM has a advance in this area and most of the working groups of a WfMS will not be willing to add another provenance system to their WfMS.

8 Future Work

Within this master thesis the main focus was laid on the field of linking up the history tracing provenance system to the workflow management system WS-VLAM and on the area of comparing the Open Provenance Model with the history tracing system. However, the paper does not answer all questions and examines not all possible issues of the history tracing system in general and of the connection of the history tracing system with WS-VLAM in specific. The necessary increments of the systems will be discussed in this chapter and possible approaches will be raised.

This section starts with the discussions of the link-up of the history tracing system with the WfMS WS-VLAM. As the new history tracing system was usually only tested with the BLAST workflow, other sample workflows have to be used to test the provenance system in detail. During running these tests the executer has to focus on different aspects. One facet the tester has to consider is whether all different paths of workflows like loops or forks and joins are supported. The loops have to be inserted in the layered structure of the XML file as enrolled loops. The Provenance Query Interface should visualize both views. One view shows arrows with the loops as an overview and the second view is visualized as an enrolled graph for the detailed view with all parameters and passed data. Moreover, the tester should use workflows, which are in use by scientists and stay in close contact to them to learn from them if all important provenance values are collected and saved in the XML file. When the users need more or other provenance values those wishes can be added to the WfMS and to the lineage system. In future the history tracing provenance system will be the default provenance framework, which is running on WS-VLAM. Therefore, the mentioned tasks will be processed and the developers try to get in contact with the bioinformaticians of the AMC to get workflows and discuss the output with them.

Thus, other enhancements have to be achieved in order to advance the effectiveness of the system. Some of the expansions are already under development to be able to use the provenance system in WS-VLAM as fast as possible. The first action, which has to be changed, is the storage system of the provenance files. Currently the lineage files are saved on the virtual machine running the web server. A better way to save the XML data is to create a database, which contains the provenance data and the catalogue data. This service should be accessible for all users and contain a small interface that allows uploading and downloading the content of the provenance files. Then, the web server and the Provenance Query Interface have to be

changed to access the provenance repository. Furthermore, the creation of the schema file has to be done automatically either by clicking a button in the WS-VLAM client or when the user saves the provenance file. Therefore, a web service method has to be implemented, which is called by the WS-VLAM client and runs the existing methods on the web server creating the schema file automatically. Furthermore, the Provenance Query Interface has to be added to the WS-VLAM client. One possibility is to add another button to the client, which opens the Provenance Query Interface and shows the provenance data. All those changes are minor changes and will not take too long to implement. The bigger task is to adapt the provenance system to the users needs and to save all those data.

The Provenance Query Interface is not finished, either. There are some wishes of the users, which could not be integrated in the Provenance Query Interface due to a limited amount of time especially the comparison of workflows and the collection of statistical data of farmed workflows. The users like to know how many modules finished correctly or how long they ran on average. All this data should be visualized as clearly as possible so that the users have a goal of using the provenance systems for their daily work. Thus, the demonstration of the data that led to a result should not be disregarded. Furthermore, the Provenance Query Interface has to be extended always if new provenance data is collected in the provenance file in order to display them intuitionally.

Another very important challenge for the future is to combine the history tracing system with as many other workflow management systems as possible. The reason for this is that the provenance files have to be shared with other scientist using different WfMS so that they can analyse the data or re-run the workflows.

The comparison of the two provenance systems could also not be finished in the depth as it was planned at the beginning of the thesis. The reason is that the implementation of the Open Provenance Model for WS-VLAM by the University of Amsterdam was not runnable. The provenance system did not collect provenance values and as a result, the discussion of the comparisons was laid on a theoretical layer. In order to prove the conclusions of the comparisons of the two lineage systems for WS-VLAM, the OPM implementation should be repaired. Then the assertions of the previous paragraph can be verified by the output of the provenance system.

Summing up, one can say that the results developed in this thesis are a good basis to continue the research on the history tracing system and the combination with WS-VLAM in the future. However, until the provenance system is usable in real research some enhancements have to be implemented.

9 References

9.1 Bibliography

[1] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, Luc Moreau, *"An Architecture for Provenance Systems"*, http://eprints.ecs.soton.ac.uk/12023/, February 2006

[2] Ekaterina Elts, Hans-Joachim Bungartz, *"Grid-Workflow-Management-Systeme für die Ausführung wissenschaftlicher Prozessabläufe"*, http://drehscheibe.in.tum.de/forschung/pub/reports/2010/TUM-I1004.pdf.gz, February 2010

[3] Yogesh L. Simmhan, Beth Plale, Dennis Gannon, *„A Survey of Data Provenance Techniques"*, http://www.cs.indiana.edu/l/www/ftp/techreports/TR618.pdf, 2005

[4] David A. Holland, Uri Braun, Diana Maclean, Kiran-Kumar Muniswamy-Reddy, Margo I. Seltzer, *„Choosing a Data Model and Query Language for Provenance"*, http://www.eecs.harvard.edu/syrah/pass/pubs/ipaw08.pdf, June 2008

[5] Luc Moreau, *„Provenance-Based Reproducibility in the Semantic Web"*, http://eprints.ecs.soton.ac.uk/21992/1/reproducibility.pdf, 2011

[6] Simon Miles, Paul Groth, Miguel Branco, Luc Moreau, *„The requirements of using provenance in e-Science experiments"*, http://eprints.ecs.soton.ac.uk/12566/1/pasoa04requirements.pdf, 2006

[7] Susan B. Davidson, Juliana Freire, *„Provenance and Scientific Workflows: Challenges and Opportunities"*, http://portal.acm.org/ft_gateway.cfm?id=1376772&type=pdf&CFID=11089175&CFTOKEN=84769110, 2008

[8] WS-VLAM, http://staff.science.uva.nl/~gvlam/wsvlam/, March 2011

[9] Vladimir Korkhov, Silvia Olabarriaga, *„Application description – BLAST"*, September 2010

[10] M. Gerhards, A. Belloum, F. Berretz, V. Sander, S. Skorupa, *"A History-tracing XML-based Provenance Framework for Workflows"*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5671873, September 2010

[11] M. Gerhards, *"Ein Framework zur Nachweisführung der Arbeitsvorgänge eines Workflows in XML"*, August 2010

[12] Luc Moreau, Ian Foster, *"Provenance and Annotation of Data"*, http://www.springer.com/computer/database+management+%26+information+retrieval/book/978-3-540-89964-8, May 2006

[13] Luc Moreau et al., *"The Open Provenance Model Core Specification (v1.1)"*, http://eprints.ecs.soton.ac.uk/21449/1/opm.pdf, December 2009

[14] David Groep, http://www.dutchgrid.nl/Admin/Nikhef/globus-local/node9.html, retrieved in March 2011

[15] Wikipedia – The free Encyclopedia, Grid Resource Allocation Manager (GRAM), http://en.wikipedia.org/wiki/Grid_Resource_Allocation_Manager, retrieved in March 2011

[16] BIOINFORMATICS LABORATORY, http://www.bioinformaticslaboratory.nl/twiki/bin/view/EBioScience/MOTEUR, retrieved in March 2011

[17] Provenance Challenge Wiki, http://twiki.ipaw.info/bin/view/Challenge/WebHome, retrieved in March 2011

[18] Luc Moreau et al., *"The Open Provenance Model (v1.01),* http://eprints.ecs.soton.ac.uk/16148/1/opm-v1.01.pdf, July 2008

[19] Wikipedia – The free Encyclopedia, Resource Description Framework, http://en.wikipedia.org/wiki/Resource_Description_Framework, retrieved in March 2011

[20] The OPM Provenance Model (OPM), http://openprovenance.org/, retrieved in March 2011

[21] Wikipedia – The free Encyclopedia, BLAST, http://en.wikipedia.org/wiki/BLAST, retrieved in March 2011

[22] Wikipedia – The free Encyclopedia, X.509, http://en.wikipedia.org/wiki/X.509, retrieved in March 2011

[23] MyProxy Credential Management Service, http://grid.ncsa.illinois.edu/myproxy/, retrieved in March 2011

[24] BMC Bioinformatics, http://www.biomedcentral.com/content/supplementary/1471-2105-11-598-s1.pdf, retrieved in March 2011

9.2 Auxiliary Means

{1} wsimport.exe
http://download.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html

{2} Graphviz – Graph Visualization Software, dot program
http://www.graphviz.org/

{3} openssl
http://www.openssl.org/

10 Appendix

10.1 Example of original history tracing system XML file

This XML file shows the complete provenance file of the example in the chapter “History tracing XML-based provenance framework for workflows” in the section “Introduction”.

Figure 32: Complete provenance file of introduction example

```
<?xml version="1.0" encoding="UTF-8"?>
<inst:sample xmlns:inst="http://sample.jpdl.xml/">
 <inst:end starttime="01.01.1980" id="end_1" tasktype="SEQ" endtime="01.01.1980">
  <inst:loop_condition starttime="01.01.1980" id="loop_condition_2" tasktype="XOR"
                                  endtime="01.01.1980">
   <inst:merge starttime="01.01.1980" id="merge_2" tasktype="XOR" endtime="01.01.1980">
    <inst:right1 starttime="01.01.1980" id="r1_1" tasktype="SEQ" endtime="01.01.1980">
     <inst:decision starttime="01.01.1980" id="decision_2" tasktype="XOR"
                                  endtime="01.01.1980">
      <inst:loop_start starttime="01.01.1980" id="loop_start_2" tasktype="XOR"
                                  endtime="01.01.1980">
       <inst:loop_condition starttime="01.01.1980" id="loop_condition_1"
                                  tasktype="XOR" endtime="01.01.1980">
        <inst:merge starttime="01.01.1980" id="merge_1" tasktype="XOR"
                                  endtime="01.01.1980">
         <inst:left2 starttime="01.01.1980" id="l2_1" tasktype="SEQ"
                                  endtime="01.01.1980">
          <inst:left1 starttime="01.01.1980" id="l1_1" tasktype="SEQ"
                                  endtime="01.01.1980">
           <inst:decision starttime="01.01.1980" id="decision_1" tasktype="XOR"
                                  endtime="01.01.1980">
            <inst:loop_start starttime="01.01.1980" id="loop_start_1"
                                  tasktype="XOR" endtime="01.01.1980">
             <inst:start starttime="01.01.1980" id="start_1" tasktype="SEQ"
                                  endtime="01.01.1980" />
            </inst:loop_start>
           </inst:decision>
           <inst:parameters>
            <inst:i>
             <![CDATA[<?xml version="1.0"?><integer>1</integer>]]>
            </inst:i>
           </inst:parameters>
          </inst:left1>
          <inst:parameters>
           <inst:i> <![CDATA[<?xml version="1.0"?><integer>1</integer>]]>
           </inst:i>
          </inst:parameters>
          <inst:results>
           <inst:i> <![CDATA[<?xml version="1.0"?><integer>2</integer>]]>
           </inst:i>
          </inst:results>
         </inst:left2>
        </inst:merge>
       </inst:loop_condition>
      </inst:loop_start>
     </inst:decision>
     <inst:results>
      <inst:i> <![CDATA[<?xml version="1.0"?><integer>3</integer>]]>
      </inst:i>
     </inst:results>
    </inst:right1>
   </inst:merge>
  </inst:loop_condition>
 </inst:end>
</inst:sample>
```

10.2 Example of XML file by history tracing system for WS-VLAM

This XML file shows the resulting provenance file that is created by the code abridgement in the chapter “History tracing system in WS-VLAM” in the section “Transmission to web service”.

Figure 33: Provenance file of fileReader/fileWriter example

```
<?xml version="1.0" encoding="utf-8"?>
<inst:BLAST xmlns:inst="http://blast.wsvlam.xml/"
       p1:schemaLocation="http://blast.wsvlam.xml/ BLAST.xsd" tasktype="ROOT"
       xmlns:p1="http://www.w3.org/2001/XMLSchema-instance" starttime="Thu Feb 10
       16:58:54 CET 2011" endtime="Thu Feb 10 16:59:40 CET 2011">
  <inst:fileWriter_916452136 starttime="Thu Feb 10 16:58:56 CET 2011"
       node="fs0.das3.cs.vu.nl" id="916452136" name="fileWriter" executefinish="Thu Feb
       10 16:59:35 CET 2011" endtime="Thu Feb 10 16:59:38 CET 2011">
    <inst:information>
      <inst:events>
      </inst:events>
      <inst:portEnd>
        <inst:start time="Thu Feb 10 16:58:59 CET 2011">
          <inst:id>1234</inst:id>
          <inst:portName>node.node1</inst:portName>
        </inst:start>
        <inst:cluster>fromInput</inst:cluster>
        <inst:port></inst:port>
      </inst:portEnd>
      <inst:parameters>
        <inst:filename>
          <inst:initialValue time="Thu Feb 10 16:59:03 CET 2011">
               C:\\Temp\\file.xml</inst:initialValue>
        </inst:filename>
        <inst:farmed>
          <inst:initialValue time="Thu Feb 10 16:59:03 CET 2011">
               false</inst:initialValue>
        </inst:farmed>
      </inst:parameters>
    </inst:information>
    <inst:fileReader_741287972 starttime="Thu Feb 10 16:58:55 CET 2011"
               node="fs0.das3.cs.vu.nl" id="741287972" name="fileReader"
               executefinish="Thu Feb 10 16:59:35 CET 2011" endtime="Thu Feb 10 16:59:37
               CET 2011">
      <inst:information>
        <inst:events>
        </inst:events>
        <inst:portStart>
          <inst:start time="Thu Feb 10 16:58:59 CET 2011">
            <inst:id>1234</inst:id>
            <inst:portName>node.node1</inst:portName>
          </inst:start>
          <inst:cluster>toOutput</inst:cluster>
          <inst:port></inst:port>
        </inst:portStart>
        <inst:parameters>
          <inst:filename>
            <inst:initialValue time="Thu Feb 10 16:59:05 CET 2011">
                     C:\\Temp\\input.tgz</inst:initialValue>
            <inst:parameterChanged time="Thu Feb 10 16:59:06 CET 2011">
                     C:\\Temp\\input.xml</inst:parameterChanged>
          </inst:filename>
        </inst:parameters>
      </inst:information>
    </inst:fileReader_741287972>
  </inst:fileWriter_916452136>
</inst:BLAST>
```

10.3 Use cases described in literature

This section contains the original use cases and technical requirements described in the paper *The requirements of using provenance in e-Science experiments*. [6] Comments on the use cases and technical requirements can be seen in the chapter "Use cases and technical requirments".

TYPES OF PROVENANCE

USE CASE 1:
A bioinformatician B downloads sequence data of a human chromosome from GenBank and performs an experiment. B later performs the same experiment on data of the same chromosome, again down-loaded from GenBank. B compares the two experiment results and notices a difference. B determines whether the difference was caused by the experimental process or configuration having been changed, or by the chromosome data being different (or both).

USE CASE 2:
A bioinformatician B enacts an experimental workflow using a workflow enactment engine W. W processes source data to produce intermediate data, and then processes the intermediate data to produce result data. B retrieves the result data. B then examines the source and intermediate data used to produce the result data.

Three different views are needed for use cases one and two:

4. Interaction: A record of the interaction between services that took place, including the data that was passed between them.
5. Actor State: Extra information about a service participating in the experiment at the time that the experiment was run.
6. Relationship: Information on how one data item in a process relates to another.

TECHNICAL REQUIREMENT 1:
The architecture should provide for the recording and querying of interactions, actor states and relationships.

STRUCTURE AND IDENTITY OF DATA

USE CASE 3:

A bioinformatician B performs an experiment on a set of chromosome data, from which the exon and intron sequences have been extracted. As a result of that experiment, B identifies a highly compressible intron sequence. B identifies which chromosome the intron originally came from.

USE CASE 4:

A physicist P extracts a subset of data from a large data set, owned by the Collaboration, and performs experiments on that subset over time. The Collaboration later updates the data set with new data. P determines whether the experiments should be re-run based on the new data set.

TECHNICAL REQUIREMENT 2:

The architecture should provide for association of identifiers with data, so that it can be referred to in queries and by data sources linking experiments together.

TECHNICALREQUIREMENT 3:

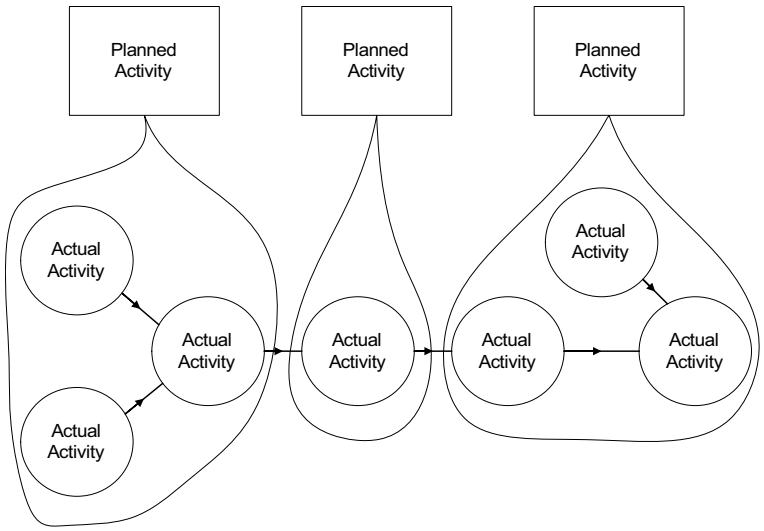
The architecture should provide for referencing of individual data elements contained in message bodies recorded in the process documentation.

METADATA AND CONTEXT

USE CASE 5:

In order to conform to health and safety requirements, a chemist C plans an experiment prior to performing it. The plan is at a high-level, e.g. including the steps of mixing and analysing materials but excluding implied steps like measuring out materials. C performs the experiment. Later, another chemist R determines whether the experiment carried out conformed to the plan.

Figure 34: Planned experiment differs from process documentation



The figure is showing the following content:
In the Second Harmonic Generation Experiment, planned activities do not map exactly to performed activities, because several activities can comprise a single planned activity. The arrows in the figure show some temporal or other dependencies between activities, which may be recorded in process documentation.

Remark to USE CASE 5:
In Use Case 5, the pre-defined plan of the experiment does not necessarily exactly match the actual steps performed. As shown in Figure 1, a single planned activity may map to one or more actual activities. As described in the use case, the plan is produced before any process documentation is recorded, but is used in comparison with the process documentation. It is an example of process metadata: data independent from but used in conjunction with process documentation. Given that process metadata is of an arbitrary wide scope, any framework for supporting the use of provenance must take into account stores of metadata that will be queried along with the process documentation.

USE CASE 6:
A biologist B sets the voltage of a mass spectrometer before performing an experiment to determine the mass-to-charge ratio of peptides. Later another biologist R judges the experiment results and considers them to be particularly accurate. R determines the voltage used in the experiment so that it can be set the same for measuring peptides of the same protein in future experiments.

USE CASE 7:

A bioinformatician B performs an experiment on a FASTA sequence encoding a nucleotide sequence. A reviewer R later determines whether or not the sequence was in fact processed by a service that actually only meaningfully processes protein sequences.

TECHNICAL REQUIREMENT 4:

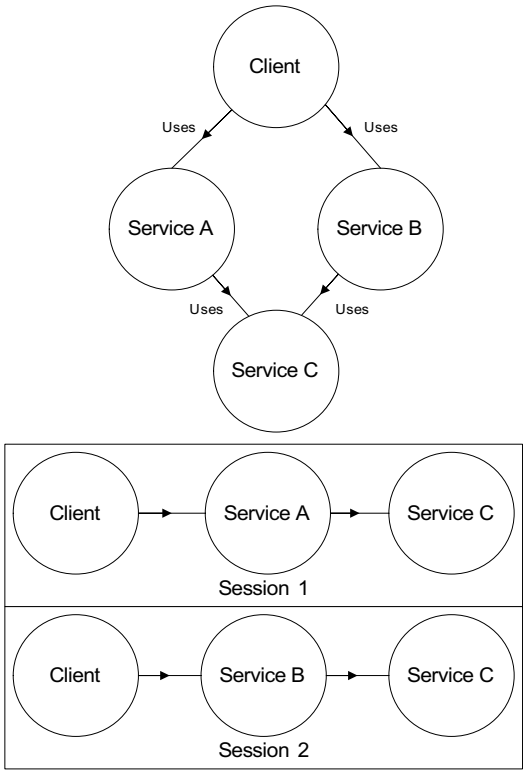
The architecture should provide for process documentation and associated metadata in different stores to being integrated in providing the answer to a query.

SESSIONS

USE CASE 8:

A computer scientist C calls service X, which calculates the mean average of two numbers as (a/2)+(b/2). C then calls service Y with the same two numbers, where Y calculates the average as (a+b)/2. C does not know if X or Y are reliable, so by getting results from both, C can compare them and, if they are the same, be more sure having the correct result (because the same value is produced by two different services). However, X and Y may use a common third service Z behind the scenes, e.g. to perform division operations. If Z is faulty then the results from X and Y may be consistent but wrong. For extra assurance, C determines whether X and Y did in fact use a common third service.

Figure 35: Split parallel processes into two single processes



The figure is showing the following content:
Sessions using the same common service in e-Demand: the client is unaware that two services, A and B performing the same function using different algorithms, rely on a common service C.

TECHNICAL REQUIREMENT 5:
The architecture should provide a mechanism, which groups recorded process documentation into a session, and should allow comparison between sessions.

QUERY

USE CASE 9:
A laboratory receives a batch of chemicals, and samples are distributed to chemists in that laboratory. A chemist C performs an experiment but then examines the results and finds them doubtful. C determines the source material used in the experiment and then which other recent experiments used material from the same batch. C examines the results of those experiments to determine whether the batch may have been contaminated and so should be discarded.

USE CASE 10:
A biologist B performs many experiments over time to discover the characteristics of peptide fragments. The fragments are used as evidence that a peptide is in the analysed material. Usually the discovery of several fragments is required to confidently identify a peptide, but some fragments are unique enough to be adequate alone. B determines that a fragment with particular characteristics is produced most times by a particular peptide and rarely or never when that peptide was not present.

TECHNICAL REQUIREMENT 6:
The architecture should provide for the process documentation to be returned in the groups specified at the time of recording or searched through on the basis of contextual criteria.

PROCESSING AND VISUALISATION

USE CASE 11:
A chemist C performs an experiment to determine the characteristics of a liquid by bouncing laser light off of it and examining the changes to the polarisation of the light. As this method is fairly new, it is not established how to then process the results. C analyses the results through a plan, i.e. a succession of processes, that seem appropriate at the time and ends with potentially interesting results. At a later date, C determines the high-level plan that they followed and re-performs the experiment with different liquid and configuration.

USE CASE 12:
A service X is accessed by an intruder I that should not have rights to do so. Later, an administrator becomes aware of the intrusion and determines the time and the credentials used by the intruder to gain access.

USE CASE 13:
A bioinformatician B performs an experiment. B publishes the results and makes a record of the experiment details available for the interest of B's peers.

TECHNICAL REQUIREMENT 7:

The architecture should provide a framework for introducing processing of process documentation of all three types discussed in „TYPES OF PROVENANCE“ (interactions, actor states and relationships), using various methods, then visualising the results of that processing.

NON-REPUDIATION

USE CASE 14:
A bioinformatician B performs an experiment from which they develop a new drug. B attempts to patent the drug. The patent reviewer R checks that the experiment did not use a database that is free only for non-commercial use, such as the Ecoli database.

USE CASE 15:
A chemist C performs an experiment finishing at a particular time. D later performs the same experiment and submits a patent for the result and the process that led to it to patent officer R. C claims to R that they performed the experiment before D. R determines whether C is correct.

TECHNICAL REQUIREMENT 8:
The architecture should provide a mechanism for recording adequate process documentation, in an unmodifiable way, to make results non-repudiable.

RE-USING EXPERIMENTAL PROCESS

USE CASE 16:
A bioinformatician B performs an experiment using as input data a specific human chromosome from the most recent version of a database. Later, another bioinformatician, D, updates the chromosome data. B re-enacts the same experiment with the most recent version of the chromosome data.

USE CASE 17:
A biologist performs an experiment to identify peptides in a sample. Identifications are made by comparing characteristics of the peptides and their fragments with already known matches in a database. In the experiment, some peptides are identified, others cannot be. Later, after other experiments have been conducted, the database contains more information. The system automatically re-enacts the analysis of those peptides that were not identified.

TECHNICAL REQUIREMENT 9:
The architecture should provide for the use of process documentation to re-enact an experiment using the same process but new inputs, and to reproduce an experiment with the same process and inputs.

AGGREGATED SERVICE INFORMATION

USE CASE 18:
Several bioinformaticians perform experiments using service X. Another bioinformatician B constructs a workflow that uses X. B can estimate the duration that the experiment might take on the basis of the average time X has taken to complete its tasks before.

TECHNICAL REQUIREMENT 10:
The architecture should provide for querying, over process documentation of multiple experiments, about the aggregate behaviour and properties of services.

10.4 Schema file for BLAST workflow

This schema file contains the definition of the BLAST workflow.

```
<?xml version="1.0" encoding="utf-8"?>
<schema targetNamespace="http://blast.wsvlam.xml/" xmlns:wf="http://blast.wsvlam.xml/"
              elementFormDefault="qualified" xmlns:ds=http://www.w3.org/2000/09/xmldsig#
              xmlns:p1="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
              schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-
              core-schema.xsd" />
  <element name="BLAST">
    <complexType>
      <all>
        <element ref="wf:fileReader_967155551" minOccurs="0" />
        <element ref="wf:fileWriter_916452136" minOccurs="0" />
        <element ref="wf:fileReader_741287972" minOccurs="0" />
        <element ref="wf:fileWriter_1599921089" minOccurs="0" />
        <element ref="wf:fileReader_458790670" minOccurs="0" />
        <element ref="wf:fileWriter_1464339253" minOccurs="0" />
        <element ref="wf:sffToFasta_81081428" minOccurs="0" />
        <element ref="wf:patternMatch_1587433265" minOccurs="0" />
        <element ref="wf:blastall_368139581" minOccurs="0" />
      </all>
      <attribute name="tasktype" use="required" type="string" fixed="ROOT" />
      <attribute name="starttime" type="string" />
      <attribute name="endtime" type="string" />
    </complexType>
  </element>
  <group name="parameter">
    <sequence>
      <element name="initialValue" minOccurs="0">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="time" type="string" />
            </extension>
          </simpleContent>
        </complexType>
      </element>
      <element name="parameterChanged" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="time" type="string" />
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </group>
  <group name="stdText">
    <sequence>
      <element name="text" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="time" type="string" />
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </group>
  <group name="portStart">
    <all>
      <element name="start" minOccurs="0">
        <complexType>
          <all>
            <element name="id" type="string" />
            <element name="portName" type="string" />
          </all>
          <attribute name="time" type="string" use="required" />
        </complexType>
      </element>
```

```
      <element name="port" type="string" />
      <element name="cluster" type="string" />
    </all>
  </group>
  <group name="portEnd">
    <all>
      <element name="port" type="string" />
      <element name="cluster" type="string" />
      <element name="end">
        <complexType>
          <all>
            <element name="id" type="string" />
            <element name="portName" type="string" />
          </all>
          <attribute name="time" type="string" use="required" />
        </complexType>
      </element>
    </all>
  </group>
  <group name="events">
    <all>
      <element name="stdoutReady" minOccurs="0">
        <complexType>
          <group ref="wf:stdText" minOccurs="0" />
        </complexType>
      </element>
      <element name="stdoutClosed" minOccurs="0">
        <complexType>
          <attribute name="time" type="string" use="required" />
        </complexType>
      </element>
      <element name="stderrReady" minOccurs="0">
        <complexType>
          <group ref="wf:stdText" minOccurs="0" />
        </complexType>
      </element>
      <element name="stderrClosed" minOccurs="0">
        <complexType>
          <attribute name="time" type="string" use="required" />
        </complexType>
      </element>
    </all>
  </group>
  <element name="fileReader_967155551">
    <complexType>
      <all>
        <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                              minOccurs="0" />
        <element name="information" minOccurs="0">
          <complexType>
            <all>
              <element name="events" minOccurs="0">
                <complexType>
                  <group ref="wf:events" minOccurs="0" />
                </complexType>
              </element>
              <element name="parameters" minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="filename" minOccurs="0">
                      <complexType>
                        <group ref="wf:parameter" minOccurs="0" />
                      </complexType>
                    </element>
                  </sequence>
                </complexType>
              </element>
              <element name="portStart" minOccurs="0" maxOccurs="1">
                <complexType>
                  <group ref="wf:portStart" />
                </complexType>
              </element>
            </all>
          </complexType>
        </element>
      </all>
      <attribute name="starttime" type="string" />
      <attribute name="endtime" type="string" />
```

```
      <attribute name="executefinish" type="string" />
      <attribute name="id" use="required" type="string" fixed="967155551" />
      <attribute name="name" use="required" type="string" fixed="fileReader" />
      <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
    </complexType>
  </element>
  <element name="fileWriter_916452136">
    <complexType>
      <all>
        <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                              minOccurs="0" />
        <element name="information" minOccurs="0">
          <complexType>
            <all>
              <element name="events" minOccurs="0">
                <complexType>
                  <group ref="wf:events" minOccurs="0" />
                </complexType>
              </element>
              <element name="parameters" minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="filename" minOccurs="0">
                      <complexType>
                        <group ref="wf:parameter" minOccurs="0" />
                      </complexType>
                    </element>
                    <element name="farmed" minOccurs="0">
                      <complexType>
                        <group ref="wf:parameter" minOccurs="0" />
                      </complexType>
                    </element>
                  </sequence>
                </complexType>
              </element>
              <element name="portEnd" minOccurs="0" maxOccurs="1">
                <complexType>
                  <group ref="wf:portEnd" />
                </complexType>
              </element>
            </all>
          </complexType>
        </element>
        <element ref="wf:sffToFasta_81081428" minOccurs="0" />
      </all>
      <attribute name="starttime" type="string" />
      <attribute name="endtime" type="string" />
      <attribute name="executefinish" type="string" />
      <attribute name="id" use="required" type="string" fixed="916452136" />
      <attribute name="name" use="required" type="string" fixed="fileWriter" />
      <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
    </complexType>
  </element>
  <element name="fileReader_741287972">
    <complexType>
      <all>
        <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                              minOccurs="0" />
        <element name="information" minOccurs="0">
          <complexType>
            <all>
              <element name="events" minOccurs="0">
                <complexType>
                  <group ref="wf:events" minOccurs="0" />
                </complexType>
              </element>
              <element name="parameters" minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="filename" minOccurs="0">
                      <complexType>
                        <group ref="wf:parameter" minOccurs="0" />
                      </complexType>
                    </element>
                  </sequence>
                </complexType>
              </element>
              <element name="portStart" minOccurs="0" maxOccurs="1">
```

```
              <complexType>
                <group ref="wf:portStart" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="741287972" />
    <attribute name="name" use="required" type="string" fixed="fileReader" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
</element>
<element name="fileWriter_1599921089">
  <complexType>
    <all>
      <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                          minOccurs="0" />
      <element name="information" minOccurs="0">
        <complexType>
          <all>
            <element name="events" minOccurs="0">
              <complexType>
                <group ref="wf:events" minOccurs="0" />
              </complexType>
            </element>
            <element name="parameters" minOccurs="0">
              <complexType>
                <sequence>
                  <element name="filename" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                  <element name="farmed" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                </sequence>
              </complexType>
            </element>
            <element name="portEnd" minOccurs="0" maxOccurs="1">
              <complexType>
                <group ref="wf:portEnd" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
      <element ref="wf:patternMatch_1587433265" minOccurs="0" />
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="1599921089" />
    <attribute name="name" use="required" type="string" fixed="fileWriter" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
</element>
<element name="fileReader_458790670">
  <complexType>
    <all>
      <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                          minOccurs="0" />
      <element name="information" minOccurs="0">
        <complexType>
          <all>
            <element name="events" minOccurs="0">
              <complexType>
                <group ref="wf:events" minOccurs="0" />
              </complexType>
            </element>
            <element name="parameters" minOccurs="0">
              <complexType>
```

```
                <sequence>
                  <element name="filename" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                </sequence>
              </complexType>
            </element>
            <element name="portStart" minOccurs="0" maxOccurs="1">
              <complexType>
                <group ref="wf:portStart" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="458790670" />
    <attribute name="name" use="required" type="string" fixed="fileReader" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
</element>
<element name="fileWriter_1464339253">
  <complexType>
    <all>
      <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                          minOccurs="0" />
      <element name="information" minOccurs="0">
        <complexType>
          <all>
            <element name="events" minOccurs="0">
              <complexType>
                <group ref="wf:events" minOccurs="0" />
              </complexType>
            </element>
            <element name="parameters" minOccurs="0">
              <complexType>
                <sequence>
                  <element name="filename" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                  <element name="farmed" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                </sequence>
              </complexType>
            </element>
            <element name="portEnd" minOccurs="0" maxOccurs="1">
              <complexType>
                <group ref="wf:portEnd" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
      <element ref="wf:blastall_368139581" minOccurs="0" />
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="1464339253" />
    <attribute name="name" use="required" type="string" fixed="fileWriter" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
</element>
<element name="sffToFasta_81081428">
  <complexType>
    <all>
      <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                          minOccurs="0" />
```

```
      <element name="information" minOccurs="0">
        <complexType>
          <all>
            <element name="events" minOccurs="0">
              <complexType>
                <group ref="wf:events" minOccurs="0" />
              </complexType>
            </element>
            <element name="portEnd" minOccurs="0" maxOccurs="1">
              <complexType>
                <group ref="wf:portEnd" />
              </complexType>
            </element>
            <element name="portStart" minOccurs="0" maxOccurs="2">
              <complexType>
                <group ref="wf:portStart" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
      <element ref="wf:fileReader_967155551" minOccurs="0" />
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="81081428" />
    <attribute name="name" use="required" type="string" fixed="sffToFasta" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
</element>
<element name="patternMatch_1587433265">
  <complexType>
    <all>
      <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
               minOccurs="0" />
      <element name="information" minOccurs="0">
        <complexType>
          <all>
            <element name="events" minOccurs="0">
              <complexType>
                <group ref="wf:events" minOccurs="0" />
              </complexType>
            </element>
            <element name="parameters" minOccurs="0">
              <complexType>
                <sequence>
                  <element name="resultPatternMatch" minOccurs="0">
                    <complexType>
                      <group ref="wf:parameter" minOccurs="0" />
                    </complexType>
                  </element>
                </sequence>
              </complexType>
            </element>
            <element name="portEnd" minOccurs="0" maxOccurs="2">
              <complexType>
                <group ref="wf:portEnd" />
              </complexType>
            </element>
            <element name="portStart" minOccurs="0" maxOccurs="2">
              <complexType>
                <group ref="wf:portStart" />
              </complexType>
            </element>
          </all>
        </complexType>
      </element>
      <element ref="wf:sffToFasta_81081428" minOccurs="0" />
      <element ref="wf:fileReader_741287972" minOccurs="0" />
    </all>
    <attribute name="starttime" type="string" />
    <attribute name="endtime" type="string" />
    <attribute name="executefinish" type="string" />
    <attribute name="id" use="required" type="string" fixed="1587433265" />
    <attribute name="name" use="required" type="string" fixed="patternMatch" />
    <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
  </complexType>
```

```
  </element>
  <element name="blastall_368139581">
    <complexType>
      <all>
        <element xmlns:q1="http://www.w3.org/2000/09/xmldsig#" ref="ds:Signature"
                              minOccurs="0" />
        <element name="information" minOccurs="0">
          <complexType>
            <all>
              <element name="events" minOccurs="0">
                <complexType>
                  <group ref="wf:events" minOccurs="0" />
                </complexType>
              </element>
              <element name="parameters" minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="tmp_path" minOccurs="0">
                      <complexType>
                        <group ref="wf:parameter" minOccurs="0" />
                      </complexType>
                    </element>
                  </sequence>
                </complexType>
              </element>
              <element name="portEnd" minOccurs="0" maxOccurs="2">
                <complexType>
                  <group ref="wf:portEnd" />
                </complexType>
              </element>
              <element name="portStart" minOccurs="0" maxOccurs="1">
                <complexType>
                  <group ref="wf:portStart" />
                </complexType>
              </element>
            </all>
          </complexType>
        </element>
        <element ref="wf:patternMatch_1587433265" minOccurs="0" />
        <element ref="wf:fileReader_458790670" minOccurs="0" />
      </all>
      <attribute name="starttime" type="string" />
      <attribute name="endtime" type="string" />
      <attribute name="executefinish" type="string" />
      <attribute name="id" use="required" type="string" fixed="368139581" />
      <attribute name="name" use="required" type="string" fixed="blastall" />
      <attribute name="node" use="required" type="string" fixed="fs0.das3.cs.vu.nl" />
    </complexType>
  </element>
</schema>
```

10.5 XML provenance file file for BLAST workflow with the history tracing system

The XML file contains the XML instance generated by the history tracing system of the BLAST workflow executed with WS-VLAM. The standard output was deleted.

```
<?xml version="1.0" encoding="utf-8"?>
<inst:BLAST xmlns:inst="http://blast.wsvlam.xml/" p1:schemaLocation="http://blast.wsvlam.xml/
        BLAST.xsd" tasktype="ROOT" starttime="Thu Feb 24 17:39:01 CET 2011" endtime="Thu Feb
        24 17:44:36 CET 2011" xmlns:p1="http://www.w3.org/2001/XMLSchema-instance">
  <inst:fileWriter_1599921089 starttime="Thu Feb 24 17:39:21 CET 2011"
        node="fs0.das3.cs.vu.nl" id="1599921089" name="fileWriter" executefinish="Thu Feb 24
        17:42:34 CET 2011" endtime="Thu Feb 24 17:42:41 CET 2011">
    <inst:information>
      <inst:events>
        <inst:stdoutReady>
          <inst:text time="Thu Feb 24 17:39:57 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:39:58 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:42:37 CET 2011">...</inst:text>
        </inst:stdoutReady>
```

```
      <inst:stdoutClosed time="Thu Feb 24 17:42:41 CET 2011" />
      <inst:stderrClosed time="Thu Feb 24 17:42:41 CET 2011" />
    </inst:events>
    <inst:parameters>
      <inst:filename>
        <inst:initialValue time="Thu Feb 24 17:42:22 CET 2011">
                file:///home/matzerat/results/result_txt</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:22 CET 2011">
                file:///home/matzerat/results/result_txt</inst:parameterChanged>
      </inst:filename>
      <inst:farmed>
        <inst:initialValue time="Thu Feb 24 17:42:23 CET 2011">false</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:23 CET 2011">
                false</inst:parameterChanged>
      </inst:farmed>
    </inst:parameters>
  </inst:information>
  <inst:patternMatch_1587433265 node="fs0.das3.cs.vu.nl"id="1587433265"name="patternMatch"/>
</inst:fileWriter_1599921089>
<inst:fileWriter_916452136 starttime="Thu Feb 24 17:39:48 CET 2011" node="fs0.das3.cs.vu.nl"
            id="916452136" name="fileWriter" executefinish="Thu Feb 24 17:42:29 CET 2011"
            endtime="Thu Feb 24 17:42:37 CET 2011">
  <inst:information>
    <inst:events>
      <inst:stdoutReady>
        <inst:text time="Thu Feb 24 17:39:53 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:39:54 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:42:33 CET 2011">...</inst:text>
      </inst:stdoutReady>
      <inst:stdoutClosed time="Thu Feb 24 17:42:37 CET 2011" />
      <inst:stderrClosed time="Thu Feb 24 17:42:37 CET 2011" />
    </inst:events>
    <inst:parameters>
      <inst:filename>
        <inst:initialValue time="Thu Feb 24 17:42:14 CET 2011">
                file:///home/matzerat/results/fastaOutput</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:14 CET 2011">
                file:///home/matzerat/results/fastaOutput</inst:parameterChanged>
      </inst:filename>
      <inst:farmed>
        <inst:initialValue time="Thu Feb 24 17:42:22 CET 2011">false</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:22 CET 2011">
                false</inst:parameterChanged>
      </inst:farmed>
    </inst:parameters>
  </inst:information>
  <inst:sffToFasta_81081428 node="fs0.das3.cs.vu.nl" id="81081428" name="sffToFasta" />
</inst:fileWriter_916452136>
<inst:sffToFasta_81081428 starttime="Thu Feb 24 17:39:51 CET 2011" node="fs0.das3.cs.vu.nl"
                id="81081428" name="sffToFasta" executefinish="Thu Feb 24 17:42:28 CET 2011"
                endtime="Thu Feb 24 17:42:40 CET 2011">
  <inst:information>
    <inst:events>
      <inst:stdoutReady>
        <inst:text time="Thu Feb 24 17:39:56 CET 2011">... </inst:text>
        <inst:text time="Thu Feb 24 17:39:56 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:39:57 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:40:26 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:42:36 CET 2011">...</inst:text>
      </inst:stdoutReady>
      <inst:stdoutClosed time="Thu Feb 24 17:42:40 CET 2011" />
      <inst:stderrClosed time="Thu Feb 24 17:42:40 CET 2011" />
    </inst:events>
    <inst:portStart>
      <inst:start time="Thu Feb 24 17:41:52 CET 2011">
        <inst:id>81081428</inst:id>
        <inst:portName></inst:portName>
      </inst:start>
      <inst:cluster>inputSff</inst:cluster>
      <inst:port></inst:port>
    </inst:portStart>
    <inst:portEnd>
      <inst:end time="Thu Feb 24 17:42:07 CET 2011">
        <inst:id>81081428</inst:id>
        <inst:portName></inst:portName>
      </inst:end>
      <inst:cluster>sffOutput</inst:cluster>
      <inst:port></inst:port>
```

```
      </inst:portEnd>
      <inst:portEnd>
        <inst:end time="Thu Feb 24 17:42:08 CET 2011">
          <inst:id>81081428</inst:id>
          <inst:portName></inst:portName>
        </inst:end>
        <inst:cluster>sffOutput</inst:cluster>
        <inst:port></inst:port>
      </inst:portEnd>
    </inst:information>
    <inst:fileReader_967155551 starttime="Thu Feb 24 17:40:18 CET 2011"
              node="fs0.das3.cs.vu.nl" id="967155551" name="fileReader" executefinish="Thu
              Feb 24 17:42:24 CET 2011" endtime="Thu Feb 24 17:42:28 CET 2011">
      <inst:information>
        <inst:events>
          <inst:stdoutReady>
            <inst:text time="Thu Feb 24 17:40:54 CET 2011">...</inst:text>
            <inst:text time="Thu Feb 24 17:42:28 CET 2011">...</inst:text>
          </inst:stdoutReady>
          <inst:stderrClosed time="Thu Feb 24 17:42:28 CET 2011" />
          <inst:stdoutClosed time="Thu Feb 24 17:42:28 CET 2011" />
        </inst:events>
        <inst:parameters>
          <inst:filename>
            <inst:initialValue time="Thu Feb 24 17:42:09 CET 2011">
                   file:///home/matzerat/BLAST-package4torsten/input-
                   data/Input_sffinfo_Component.tar
            </inst:initialValue>
            <inst:parameterChanged time="Thu Feb 24 17:42:09 CET 2011">
                  file:///home/matzerat/BLAST-package4torsten/input-
                  data/Input_sffinfo_Component.tar
            </inst:parameterChanged>
          </inst:filename>
        </inst:parameters>
      </inst:information>
    </inst:fileReader_967155551>
  </inst:sffToFasta_81081428>
  <inst:patternMatch_1587433265 starttime="Thu Feb 24 17:40:51 CET 2011"
              node="fs0.das3.cs.vu.nl" id="1587433265" name="patternMatch" executefinish="Thu
              Feb 24 17:42:34 CET 2011" endtime="Thu Feb 24 17:42:39 CET 2011">
    <inst:information>
      <inst:events>
        <inst:stdoutReady>
          <inst:text time="Thu Feb 24 17:41:25 CET 2011">...<inst:text>
          <inst:text time="Thu Feb 24 17:42:35 CET 2011">...</inst:text>
        </inst:stdoutReady>
        <inst:stdoutClosed time="Thu Feb 24 17:42:39 CET 2011" />
        <inst:stderrClosed time="Thu Feb 24 17:42:39 CET 2011" />
      </inst:events>
      <inst:portStart>
        <inst:start time="Thu Feb 24 17:42:08 CET 2011">
          <inst:id>1587433265</inst:id>
          <inst:portName></inst:portName>
        </inst:start>
        <inst:cluster>sffOutputFile</inst:cluster>
        <inst:port></inst:port>
      </inst:portStart>
      <inst:portStart>
        <inst:start time="Thu Feb 24 17:42:08 CET 2011">
          <inst:id>1587433265</inst:id>
          <inst:portName></inst:portName>
        </inst:start>
        <inst:cluster>patternFile</inst:cluster>
        <inst:port></inst:port>
      </inst:portStart>
      <inst:portEnd>
        <inst:end time="Thu Feb 24 17:42:08 CET 2011">
          <inst:id>1587433265</inst:id>
          <inst:portName></inst:portName>
        </inst:end>
        <inst:cluster>result_fasta</inst:cluster>
        <inst:port></inst:port>
      </inst:portEnd>
      <inst:portEnd>
        <inst:end time="Thu Feb 24 17:42:09 CET 2011">
          <inst:id>1587433265</inst:id>
          <inst:portName></inst:portName>
        </inst:end>
```

```
      <inst:cluster>result_txt</inst:cluster>
      <inst:port></inst:port>
    </inst:portEnd>
    <inst:parameters>
      <inst:resultPatternMatch>
        <inst:initialValue time="Thu Feb 24 17:42:24 CET 2011">
                  /home/matzerat/results/resultPatternMatch</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:24 CET 2011">
                  /home/matzerat/results/resultPatternMatch</inst:parameterChanged>
      </inst:resultPatternMatch>
    </inst:parameters>
  </inst:information>
  <inst:sffToFasta_81081428 node="fs0.das3.cs.vu.nl" id="81081428" name="sffToFasta" />
  <inst:fileReader_741287972 starttime="Thu Feb 24 17:41:52 CET 2011"
              node="fs0.das3.cs.vu.nl" id="741287972" name="fileReader" executefinish="Thu
              Feb 24 17:42:24 CET 2011" endtime="Thu Feb 24 17:42:32 CET 2011">
    <inst:information>
      <inst:events>
        <inst:stdoutReady>
          <inst:text time="Thu Feb 24 17:42:28 CET 2011">...</inst:text>
        </inst:stdoutReady>
        <inst:stdoutClosed time="Thu Feb 24 17:42:32 CET 2011" />
        <inst:stderrClosed time="Thu Feb 24 17:42:32 CET 2011" />
      </inst:events>
      <inst:parameters>
        <inst:filename>
          <inst:initialValue time="Thu Feb 24 17:42:22 CET 2011">
                file:///home/matzerat/BLAST-package4torsten/input-data/tagMichel20080707.txt
          </inst:initialValue>
          <inst:parameterChanged time="Thu Feb 24 17:42:22 CET 2011">
                file:///home/matzerat/BLAST-package4torsten/input-data/tagMichel20080707.txt
          </inst:parameterChanged>
        </inst:filename>
      </inst:parameters>
    </inst:information>
  </inst:fileReader_741287972>
</inst:patternMatch_1587433265>
<inst:fileWriter_1464339253 starttime="Thu Feb 24 17:41:21 CET 2011"
            node="fs0.das3.cs.vu.nl" id="1464339253" name="fileWriter" executefinish="Thu
            Feb 24 17:44:21 CET 2011" endtime="Thu Feb 24 17:44:33 CET 2011">
  <inst:information>
    <inst:events>
      <inst:stdoutReady>
        <inst:text time="Thu Feb 24 17:41:56 CET 2011">...</inst:text>
        <inst:text time="Thu Feb 24 17:42:46 CET 2011...</inst:text>
        <inst:text time="Thu Feb 24 17:44:26 CET 2011">...</inst:text>
      </inst:stdoutReady>
      <inst:stderrClosed time="Thu Feb 24 17:44:33 CET 2011" />
      <inst:stdoutClosed time="Thu Feb 24 17:44:33 CET 2011" />
    </inst:events>
    <inst:parameters>
      <inst:filename>
        <inst:initialValue time="Thu Feb 24 17:42:23 CET 2011">
                file:///home/matzerat/results/out_blast_tar</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:23 CET 2011">
                file:///home/matzerat/results/out_blast_tar</inst:parameterChanged>
      </inst:filename>
      <inst:farmed>
        <inst:initialValue time="Thu Feb 24 17:42:23 CET 2011">false</inst:initialValue>
        <inst:parameterChanged time="Thu Feb 24 17:42:23 CET 2011">
                false</inst:parameterChanged>
      </inst:farmed>
    </inst:parameters>
  </inst:information>
  <inst:blastall_368139581 starttime="Thu Feb 24 17:39:18 CET 2011" node="fs0.das3.cs.vu.nl"
              id="368139581" name="blastall" executefinish="Thu Feb 24 17:44:20 CET
              2011" endtime="Thu Feb 24 17:44:30 CET 2011">
    <inst:information>
      <inst:events>
        <inst:stdoutReady>
          <inst:text time="Thu Feb 24 17:39:54 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:39:55 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:42:44 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:44:14 CET 2011">...</inst:text>
          <inst:text time="Thu Feb 24 17:44:24 CET 2011">...</inst:text>
        </inst:stdoutReady>
        <inst:stdoutClosed time="Thu Feb 24 17:44:30 CET 2011" />
        <inst:stderrClosed time="Thu Feb 24 17:44:30 CET 2011" />
```

```
        </inst:events>
        <inst:portStart>
          <inst:start time="Thu Feb 24 17:42:08 CET 2011">
            <inst:id>368139581</inst:id>
            <inst:portName></inst:portName>
          </inst:start>
          <inst:cluster>result_fasta</inst:cluster>
          <inst:port></inst:port>
        </inst:portStart>
        <inst:portStart>
          <inst:start time="Thu Feb 24 17:42:08 CET 2011">
            <inst:id>368139581</inst:id>
            <inst:portName></inst:portName>
          </inst:start>
          <inst:cluster>Ref_File_gz</inst:cluster>
          <inst:port></inst:port>
        </inst:portStart>
        <inst:portEnd>
          <inst:end time="Thu Feb 24 17:42:09 CET 2011">
            <inst:id>368139581</inst:id>
            <inst:portName></inst:portName>
          </inst:end>
          <inst:cluster>out_blast_tar</inst:cluster>
          <inst:port></inst:port>
        </inst:portEnd>
        <inst:parameters>
          <inst:tmp_path>
            <inst:initialValue time="Thu Feb 24 17:42:24 CET 2011">
                        /var/scratch/matzerat/result</inst:initialValue>
            <inst:parameterChanged time="Thu Feb 24 17:42:24 CET 2011">
                        /var/scratch/matzerat/result</inst:parameterChanged>
          </inst:tmp_path>
        </inst:parameters>
      </inst:information>
      <inst:fileReader_458790670 starttime="Thu Feb 24 17:40:21 CET 2011"
              node="fs0.das3.cs.vu.nl" id="458790670" name="fileReader" executefinish="Thu Feb
              24 17:42:24 CET 2011" endtime="Thu Feb 24 17:42:31 CET 2011">
        <inst:information>
          <inst:events>
            <inst:stdoutReady>
              <inst:text time="Thu Feb 24 17:40:57 CET 2011">...</inst:text>
              <inst:text time="Thu Feb 24 17:42:27 CET 2011">...</inst:text>
            </inst:stdoutReady>
            <inst:stdoutClosed time="Thu Feb 24 17:42:31 CET 2011" />
            <inst:stderrClosed time="Thu Feb 24 17:42:31 CET 2011" />
          </inst:events>
          <inst:parameters>
            <inst:filename>
              <inst:initialValue time="Thu Feb 24 17:42:23 CET 2011">
                     file:///home/matzerat/BLAST-package4torsten/input-data/Ribosomal_Human.gz
              </inst:initialValue>
              <inst:parameterChanged time="Thu Feb 24 17:42:23 CET 2011">
                     file:///home/matzerat/BLAST-package4torsten/input-data/Ribosomal_Human.gz
              </inst:parameterChanged>
            </inst:filename>
          </inst:parameters>
        </inst:information>
      </inst:fileReader_458790670>
      <inst:patternMatch_1587433265 node="fs0.das3.cs.vu.nl" id="1587433265"
               name="patternMatch" />
    </inst:blastall_368139581>
  </inst:fileWriter_1464339253>
</inst:BLAST>
```