Master Thesis

A Portable Project Management System for Biological Data Analysis using Web Services

Evaluator:
ao. Prof. Dr. Jürgen Dorn
Vienna University of Technology,
Institute of Software Technology and Interactive Systems

Supervisor:
Dr. Albert Kriegner
Austrian Research Centers GmbH,
Molecular Diagnostics

Erkan Dilaveroğlu
0125918 / 937
edilaver@gmail.com

Vienna, March 2008

Acknowledgements

I would like to thank all those who have contributed to the emergence of this work by technical as well as by personal support. Thanks especially go to Prof. Jürgen Dorn who has contributed valuable advice to this work.

Special thanks go to my colleagues from the Molecular Diagnostics department of the Austrian Research Centers GbmH who have made the execution of this work easier for me by creating a pleasant working environment. To Dr. Albert Kriegner I owe particular thanks for his continuous advice and support during my work.

We have come all the long way as a group of six friends known as "Fatihliler". I want to thank you all, Ahmet Yildiz, Ali Osman Kusakci, Ilhami Visne, Musa Büyükkaba and Tugrul Tarhana for your friendship during my university journey.

This work would not have been possible without the kind support of the Wonder Student Association which has first offered and then supported my education in Vienna. Particular thanks go to Ibrahim Solmaz, Yusuf Ziyaettin Sula, and Yusuf Kara. They are the men who encouraged me to go for further education and who made it possible by supporting me during my studies. Particular thanks also go to Nadire Kara for supporting all my works. Mr. and Mrs. Kara are to be considered as our family in Vienna.

Special thanks go to my family; Halil Ibrahim, Ayse, Ergün and Büsra Dilaveroglu for always being at my side. Extra special thanks go to my brother Ergün who assumed my responsibilities and made my education easier.

Finally, I particularly want to thank my wonderful wife, Inci, for her continued support, motivation and patience during this work.

Abstract

Analysis of biological data is complex and requires highly flexible analysis tools. Each of these tools generates vast amounts of biological data. In spite of recent advances in workflow based data analysis, little effort has been made in the development of new mechanism for the automated storage, integration and management of the flow of analysis data.

The reproducibility of scientific experiments is one of the most important characteristics in analysis tools. However, currently available analysis tools have only limited provenance support.

The aim of this work is to develop a data management framework (Portable Project Management System) that is optimized to manage, store, integrate and secure the data stream generated during entire analysis processes. A complete recording of the processes is of very high scientific importance.

The Portable Project Management System (PPMS©) Server was developed as a Soap based Web Service. This system supports multiple analysis tools; special focus was set on compatibility with the ARC Analysis Platform. In addition to remote usage mechanism, local storage mechanisms for offline usage are also supported.

The system contains a project model which defines the constraints to pack all analysis data. It further has an integrated provenance support which caters for the logging of all the executed processes. A special feature records parameter settings from all analysis runs, and thus enables to rerun legacy pipelines.

Finally, as the PPMS© Server is offered as Soap based Web Service, it is accessible to other systems, independent of the platform used.

Kurzfassung

Die Analyse biologischer Datensätze ist oftmals komplex und besteht aus einer Vielzahl an einzelnen Analyseschritten. Dementsprechend hoch ist der Anspruch an die verwendeten Analysewerkzeuge, wo man zunehmend auf die Flexibilität von Workflowsystemen vertraut um komplexe Analysefolgen als sogenannte Analysepipelines abzubilden. Dieser Entwicklung zum Trotz gibt es vergleichsweise wenig neue Ansätze zur Strukturierung, Integration und Verwaltung des Analysedatenflusses.

Die Nachvollziehbarkeit wissenschaftlicher Experimente ist einer der wichtigsten Charakteristiken in Analysewerkzeuge. Diese wird hingegen in derzeit existierendem Analysewerkzeuge nur beschränkt unterstützt.

Im Rahmen dieser Diplomarbeit wurde eine generisches Datenverwaltungsframework (Portable Projekt-Management-System) entwickelt, das Analysedaten mitloggen, automatisch strukturieren und verwalten kann mit dem Ziel, einer lückenlosen Erfassung des gesamten Datenstroms komplexer Analysen.

Das System wurde dafür konzipiert, verschiedene Analysesysteme zu unterstützen, wobei ein spezieller Fokus auf die Kompatibilität mit der ARC Analyse Plattform gesetzt wurde. PPMS© Server wurde als Soap basiertes Web Service entwickelt, der Client besitzt aber auch Mechanismen, die das Arbeiten auch offline ermöglichen.

Das System umfasst ein Projektmodell, um alle erforderlichen Analyse Daten zu vereinen; ein integriertes Provenance System zur Protokollierung aller erfolgten Schritte und spezielle Module die es ermöglichen, die Funktion bzw. die Pipelines des jeweiligen Analysesystems mit vorher gespeicherten Parametern erneut auszuführen.

Neben seinem Einsatz im Rahmen der ARC Analyseworkbench kann das PPMS© von verschiedenen Plattformen als ein vom Analysesystem unabhängiges, eigenständiges Soap basiertes Web Service benutzt werden.

Table of contents

Chapter 1

1 Introduction

This chapter introduces the motivation for this project. The motivation part describes the issue and mentions the key features of the solution which will be explained in detail in the subsequent chapters.

Then, the terminology including the terms throughout this document will be addressed in order to prevent possible misunderstandings already from the beginning.

1.1 Motivation

Analysis steps in the life sciences are aiming at finding out the structure, the behavior and sometimes the quantity of the analyzed samples. Biological data analysis is complex and requires highly flexible analysis tools. An analysis contains several parts that are to be executed in order to achieve the defined goal.

Research projects in the life sciences make intensive use of IT tools to manage, store, retrieve and particularly to secure the created data. In spite of recent advances in workflow based high throughput data analysis, little effort has been made in the development of new mechanism for the automated storage, integration and management of the analysis data flow. Therefore, each research assistant should find his way of securing, storing the data in a way that he can understand and repeat the analysis whenever needed. Without preserving (or preserving in an insufficient way) the input/output data and the information on how the analysis is carried out, an analysis would be irrelevant and not reproducible.

Research assistants are often still using the simplest methods to keep record of what and how they performed an analysis and they store analysis information in an unstructured and insecure way. Also analysis results are stored in similar ways which are a barrier to the quality assurance of analysis procedures.

Tools nowadays offer workbenches to execute the analysis steps but they do not truly integrate mechanisms to manage the generated data, to store intermediate results or to secure them and neither do they keep track of what has been done although the reproducibility of scientific experiments should be one of the most important characteristics in analysis tools. Some of them offer solutions to these issues but they are far too inefficient to meet the requirements of biomedical research.

The aim of this work is to develop a Portable Project Management System (PPMS©) based on web services. The system supports multiple analysis tools. Special focus is set on compatibility with the ARC Analysis Platform.

The PPMS© defines a project model to combine all the required analysis data. The model defines the frames of a project and implements barriers to ensure that the project stays uninjured.

Moreover, integrated provenance supports take care of logging all the executed steps, including intermediate results, metadata given by research assistants, calculation options selected and other possible data which could be used during the analysis process. This enables to later on recall the analysis whenever it is needed in a structured way and it ensures that the analysis stays reproducible and remains reusable.

Finally, the PPMS© will be offered as a soap based web service in order to make it accessible to other systems, independent of the platform used.

1.2 Terminology

This section gives a short overview of the key terms used in this document.

Microarray

Sets of miniaturized chemical reaction areas that are used to test DNA fragments, antibodies, or proteins, by using a chip having immobilized targets and hybridizing them with a fluorescently labeled sample. The color we get from the chip after hybridization is then scanned and the data are analyzed by a soft ware to evaluate the distinct expression levels. [1]

Function

A *function* is the simplest executable unit that can be used in a system.

Analysis

An *analysis* is the process of executing one or more *function*s in succession to reach a defined research aim.

Run

The execution of a unique *function* is defined as a *run*.

Workflow / Pipeline

A workflow is the model of the tasks or the model of successive function executions which are associated with each other by input and output information. A workflow is generally defined for a certain analysis.

Probe

One or more files representing input data or the data from a patient which are imported for the use in an analysis.

Chapter 2

2 Provenance Technologies

2.1 Introduction

The following section introduces the components that are developed by the open-source community or bioinformatics institutions and that are used in current software solutions.

The provenance is one of the most important characteristics in analysis tools because researchers should be able to retrace the legacy analysis when needed.

It is certain that the following list of components can be expanded. I have included only the most relevant components among all the currently available possibilities.

2.2 Provenance

First of all I want to give a common sense definition of the word "provenance". Its etymology is the French verb 'provenir', which means to come forth, to originate. According to the Oxford English Dictionary, provenance is defined as:

(i) *the fact of coming from some particular source or quarter; origin, derivation.*

(ii) *the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners.*

As an informatics term the meaning differs between research fields because different scientific fields require different metadata in the means of provenance. Aerospace engineering, organ transplant management, and bioinformatics are example areas where the term provenance is currently used. For this work, I prefer the definition of data provenance as the information that

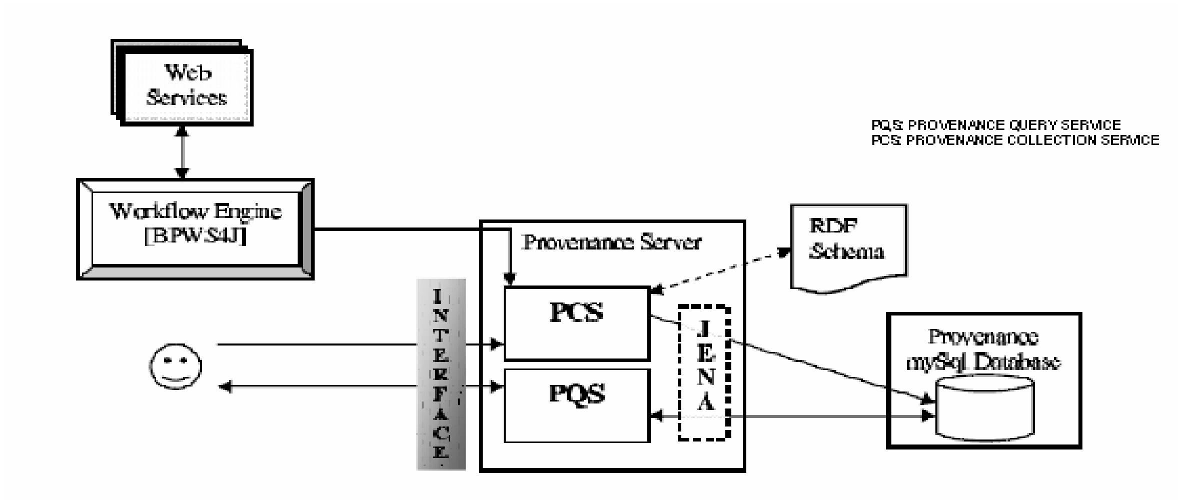

Web Services
Workflow Engine [BPWS4J]
PQS: PROVENANCE QUERY SERVICE
PCS: PROVENANCE COLLECTION SERVICE
Provenance Server
RDF Schema
PCS
PQS
INTERFACE
JENA
Provenance mySql Database

Figure 1: A possible Provenance Model for Service Oriented Architectures

helps determining the derivation history of a data product, starting from its original sources.

For experiments in the life sciences, it is vital to record the experimental process for later use such as for interpreting results, verifying that the correct process took place or tracing where data came from. For generations scientists have used basic methods for capturing provenance information [2].

Provenance systems can be used for several application purposes. Goble [3] categorizes these purposes in five groups:

1. for data quality and reliability based on the source,
2. for audit trail of data source,
3. for repetition of data derivation,
4. for establishment of the copyright and ownership of data,
5. for querying lineage metadata for data discovery.

Different application purposes require different collection levels. In a specific model called *data oriented model*, provenance information is gathered about data. This means that meta-

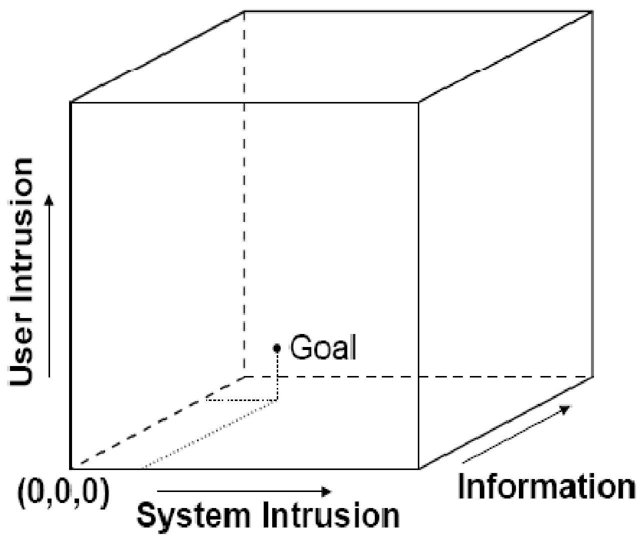


Figure 2: Provenance Trade-Offs Cube

information about a data product is collected and stored in repositories to query for information extraction when needed. Another model called *process oriented model*, concentrates on process levels and collects information about processes which are executed. The provenance can be determined then by inspecting that information. There are mainly input/output data and calculation parameters of processes [4].

One other aspect which has influence on how the data should be collected is the way of the presentation of the data. Depending on how rich the provenance information is and how large the storage resources are, the collection and presentation method changes.

There are two major approaches of representing provenance information: *eager* and *lazy*. The eager form uses pre-compiled metadata information which is collected about data source and processes as annotations. The inversion method computes the provenance data only when needed and property information collected at runtime inverts the derivations and finds out the input data supplied to functions to produce the output data [5].

It is to mention that provenance capturing presents a trade-off. Reilly [6] describes this trade-off between the amount of provenance information gathered and intrusions on the system and the user with a cube where the amount of intrusion on the system is on the x-axis, the amount of intrusion on the user is on the y-axis, and the amount of provenance information is on the z-axis. The range of each axis is 0 to 1. A system that has no provenance capabilities is located at the (0, 0, 0) point. A system located anywhere on the back face of the cube, where the z-axis is equal to 1, collects all possible provenance information. The ultimate and perhaps unachievable

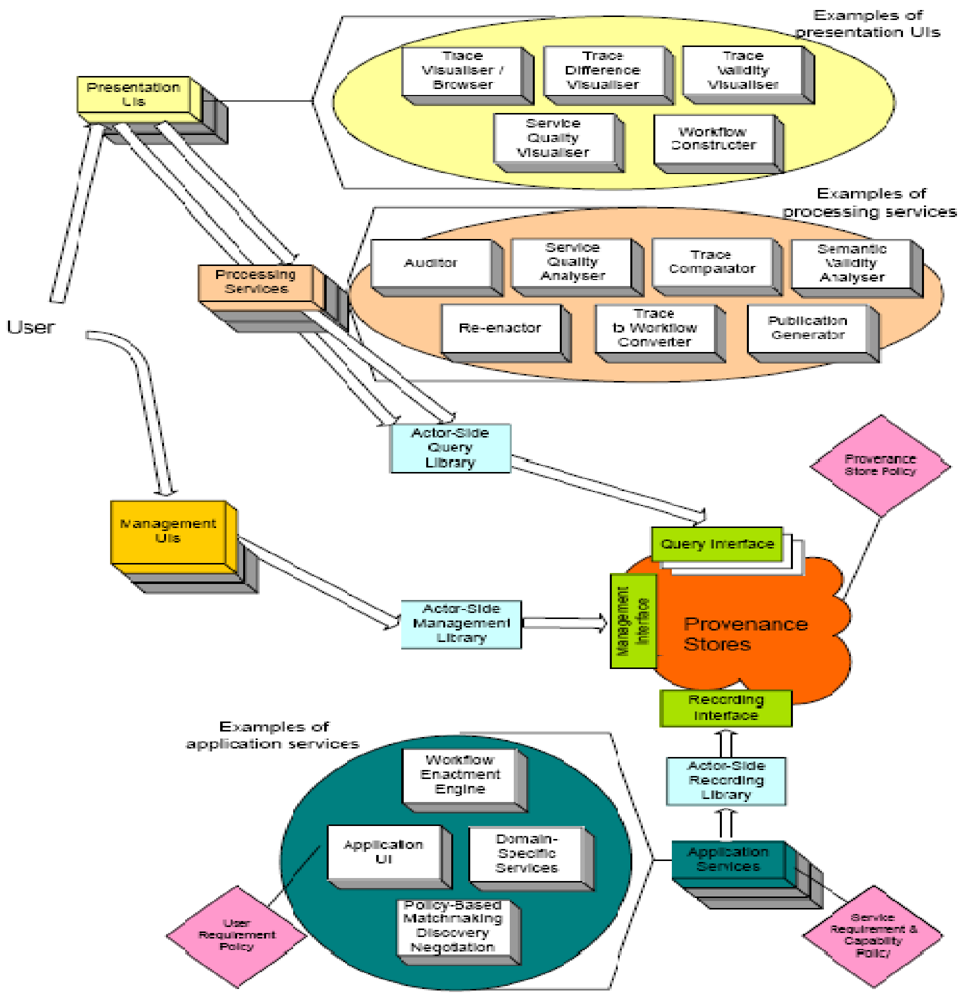

Examples of presentation UIs
Trace Visualiser / Browser
Trace Difference Visualiser
Trace Validity Visualiser
Service Quality Visualiser
Workflow Constructer
Presentation UIs
Examples of processing services
Auditor
Service Quality Analyser
Trace Comparator
Semantic Validity Analyser
Re-enactor
Trace to Workflow Converter
Publication Generator
Processing Services
User
Actor-Side Query Library
Provenance Store Policy
Management UIs
Query Interface
Management Interface
Provenance Stores
Actor-Side Management Library
Recording Interface
Actor-Side Recording Library
Examples of application services
Workflow Enactment Engine
Application UI
Domain-Specific Services
Policy-Based Matchmaking Discovery Negotiation
User Requirement Policy
Application Services
Service Requirement & Capability Policy

Figure 3: Architecture of a possible Provenance-Aware Application

goal is the (0, 0, 1) point, where all provenance information is provided with no intrusion to either the system or the user. The goal is a system that provides a large amount of provenance information while having only small intrusions on both the user and the provenance aware system. The point in Figure 2 labeled “Goal” is intended to loosely suggest a desirable location, where the cost of moving further back in the cube would require dramatic increases in the intrusion on the user and/or the system.

There have been several middleware solutions from different research fields developed with/for provenance support. Some of them are: *Scientific Annotation Middleware (SAM)* [7], *GriPhyN Virtual Data System* [8]*, myGrid* [9]*, PreServ: Provenance Recording for Services* [10]*, The Earth System Workbench* [11]*, and The Collaboratory for the Multi-scale Chemical Sciences* [12].

In the following, I am investigating some of the solutions which are relevant for this work.

2.2.1 PreServ: Provenance Recording for Services

PreServ is a software package developed by the PASOA Consortium [10]. The PASOA Consortium aims to investigate the concept of provenance and bring the provenance to e-Science. The consortium is funded by the Engineering and Physical Sciences Research Council (EPSRC).

The PASOA Consortium declares the objectives of the project as:

- to define execution and service provenance in relation to workflow enactment.
- to conceive algorithms to reason over provenance data, in order to help scientists to achieve better utilization of Grid resources for their specific tasks.
- to design a distributed cooperation protocol to generate provenance data in workflow enactment.
- to investigate value-added properties that can be deduced from provenance-based data.
- to engineer a proof of concept software architecture to support provenance generation and reasoning in Grid environments.

A provenance support system should at least support the two functions of storing provenance data and of querying provenance information. The PASOA Consortium has developed a so-called P-Assertion Recording Protocol (PReP) and has defined a recording process for the process information for Grid based applications. Because the *de facto* architecture in Grid applications is the Service Oriented Architecture (SOA), PReP was designed for SOAs in a generic manner.

The PreServ software package contains a web service for provenance storing, interfaces for recording provenance information and querying them, a set of Java libraries for accessing those interfaces, and an Axis handler for automatically recording message exchange p-assertions for Axis based web services.

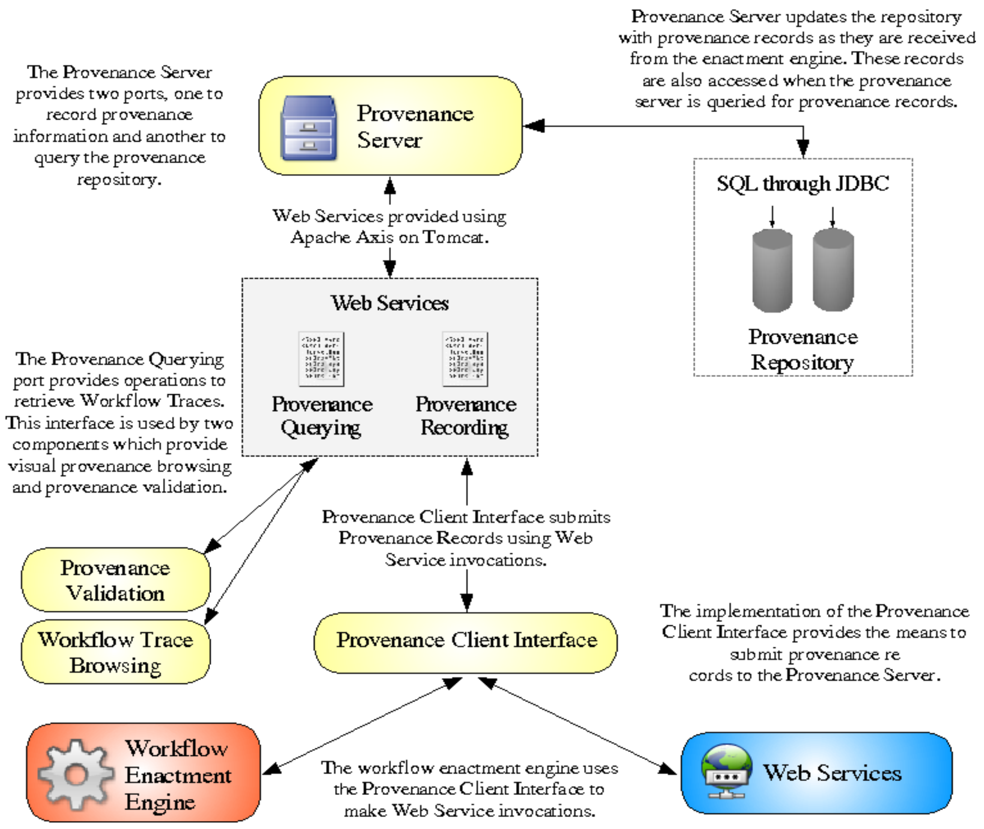

Provenance Server updates the repository with provenance records as they are received from the enactment engine. These records are also accessed when the provenance server is queried for provenance records.
The Provenance Server provides two ports, one to record provenance information and another to query the provenance repository.
Provenance Server
SQL through JDBC
Provenance Repository
Web Services provided using Apache Axis on Tomcat.
Web Services
Provenance Querying
Provenance Recording
The Provenance Querying port provides operations to retrieve Workflow Traces. This interface is used by two components which provide visual provenance browsing and provenance validation.
Provenance Client Interface submits Provenance Records using Web Service invocations.
Provenance Validation
Workflow Trace Browsing
Provenance Client Interface
The implementation of the Provenance Client Interface provides the means to submit provenance re cords to the Provenance Server.
Workflow Enactment Engine
The workflow enactment engine uses the Provenance Client Interface to make Web Service invocations.
Web Services

Figure 4: PreServ System Overview

2.2.1.1 P-Assertion Recording Protocol (PReP)

PReP introduces the protocols and defines:

- The schema for a record request message sent to provenance store
- The behavior of a provenance store when it gets such a record request
- The acknowledgement message which is sent as an answer to the requester

The XML scheme that defines the structure of incoming request and outgoing acknowledgement messages is given in Appendix A. 1. The WSDL 1.1 description of the interface is given in Appendix A. 2. It defines web services for taking and producing these messages.

PReP defines so-called *actors* which represent tools, services or any piece of software which has to store provenance information in a provenance store. PReP does not dictate any storing mechanism which means that it is independent from the underlying storing technology.

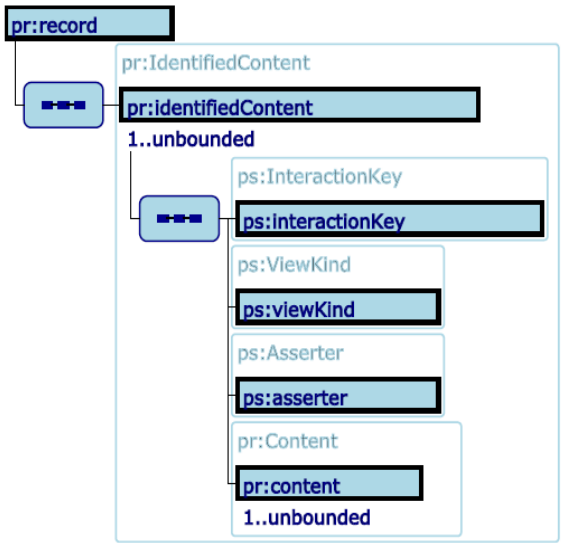


Figure 5: Process Documentation Record Request in XML Schema Diagram

A *provenance request* is a message sent by an *actor* to a provenance store which contains provenance information to be stored.

Owing to the XML scheme design (see Figure 5) it is possible to send multiple pieces of provenance information bundled at one time. This enables recording information about different interactions at a certain time which makes it possible to determine the time for transferring information. In network based distributed systems such a possibility is due to the urgent traffic problems. One can send his data at a time when network traffic is so low that provenance recording is more acceptable.

The message to be stored in the province store is bound to content attribute of the soap message. On an incoming store-request provenance the storing service has to extract the information and store it in an underlying storage device. In addition to that, a provenance storage service has to send an acknowledgement message to the actor that issued the request message. Figure 6 shows an XML scheme diagram of how such a message can be constructed. One other feature is that acknowledgement messages can be sent back either synchronously or asynchronously depending on the underlying communication mechanisms between client and server.

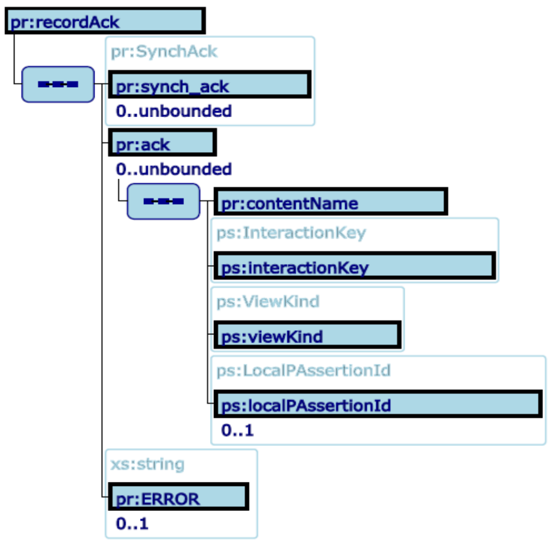


Figure 6: Process Documentation Record Acknowledgement

The acknowledgement message scheme has an element "*error*" in which the provenance storing service can put implementation specific error messages and let the using workbench give feedback to the user or react accordingly.

Not to argue that such a distributed mechanism which uses the internet network for message exchange should consider security issues. The developers of PReP suggest [13] that communication should be secured using the mechanisms described in WS-Security [14]. Alone, securing the communication is not a convenient way of securing. Knowing that, the authors suggest additional security for the data which means that the body of the soap message and all header information should be secured as well by using digital signatures. With that the integrity of the message is guaranteed. If, however a client continuously sends messages to the store, it is recommended by the authors that a secure communication channel is established using the mechanisms described in WS-Trust [15] and WS-SecureConversation [16].

One other security issue is how to secure the data in a provenance store. The authors recommend using the security architecture displayed in Figure 7. I will not go further into detail, but this architecture is described in paper [17] in detail.

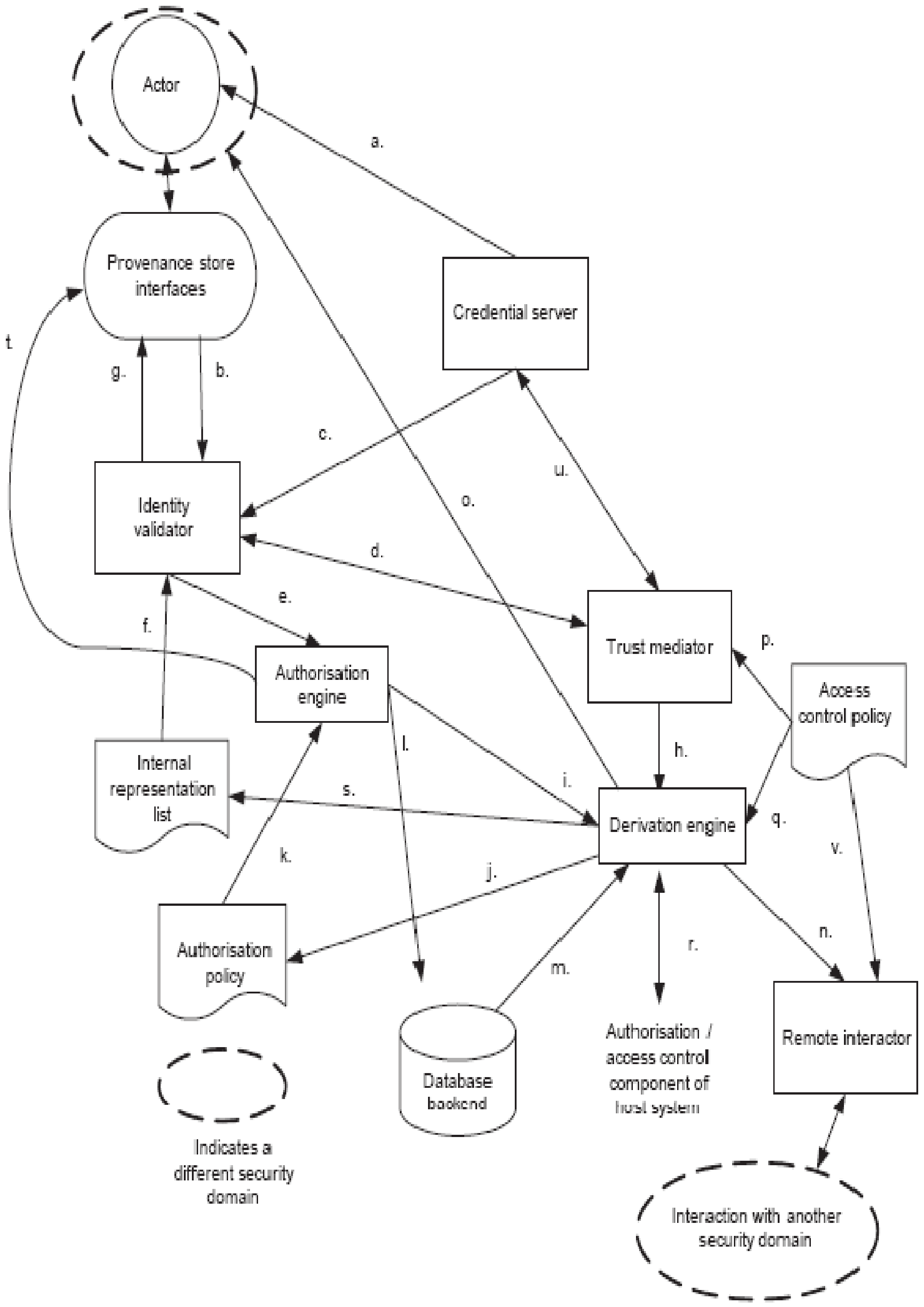


Figure 7: Provenance store security architecture

2.2.1.2 PreServ: Implementation

PreServ is an implementation of PReP which takes the form of an open source Java-based web services implementation of the protocol [18]. PreServ has been developed and used to make a number of applications provenance-aware. One of these applications used to evaluate PreServ's performance and the results are published by Groth [18]. I will focus on this later on in this chapter.

I will introduce the PreServ implementation in two aspects. One is the implementation of provenance store service which is responsible for answering incoming store-requests, extracting provenance data from soap messages and writing it to the underlying storing device. The other is the implementation of query service which is responsible for information extraction from the data kept.

PreServ has been developed unconnected to any application but is intended to stay generic and platform-independent so that all systems and services can make use of it. A special focus was put on the interoperability with Grid applications. Trying to keep scalability high, being highly expandable and meanwhile maintaining a meaningful degree of independence were decisive points of the technology determination for PreServ development. Considering these facts, the decision for Java as the implementation language is understandable since using Java PreServ runs on Windows, Solaris, Linux and Macintosh OS X platforms without modification.

The provenance store service itself has been developed as a Java servlet. PreServ uses the Apache Tomcat servlet container. One can think that a Web Service Container is the better choice but such a middleware (like Apache Axis) prevents provenance services from getting row soap messages and parsing it. The point is that PreServ needs to store soap message directly. Of course, not using web service containers has a burden: the axis can automatically bind XML data structures to Java Objects, which makes life very easier.

The provenance Store has been developed in three layers (see Figure 8): the Message Translator, which isolates provenance store's store and queries logic from the message layer; the Plug-Ins layer, which contains different plug-ins for specific functionalities that provenance service provides; and finally the Backend Storage layer where provenance information is physically stored.

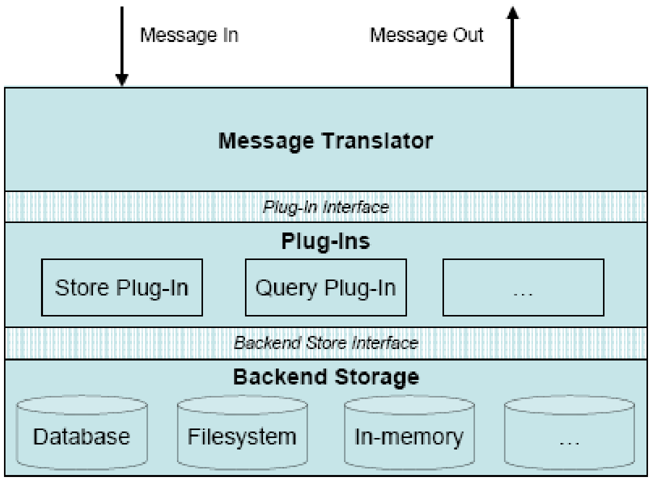


Figure 8: The Provenance Store concrete architecture

The first layer - the Message Translator - gets incoming soap messages and translates it to the format that the second layer can handle. By placing such a layer on top of the architecture it is ensured that other messaging technologies are easily supported by writing corresponding translator layer and exchanging them with the current one. One can develop the message translator layer that handles messages transferred using Java Remote Interfaces and use it without making any modification to underlying layers. In addition to translating incoming messages to the provenance store's internal format, the Message Translator is responsible for handling outgoing message formats. That means the internal format should be translated into the external format, when needed.

One other feature of the Message Translator is that it detects the appropriate plug-in to pass the message by checking the HTTP context of the incoming message. For example, if the message sent to the http://arcs.ac.at/life_sciences/provenance/record Message Translator understands that the message context is /record, it passes it to the PlugInHandler to get a reference to the appropriate plug-in for mapping.

The second layer is the plug-ins layer. As mentioned before, this layer has a set of plug-ins of which each represents a piece of functionality for the provenance store. These can have a functionality that enables storing p-assertions or deleting, and moving between different provenance store devices, or even for capturing information from already stored data. A plug-in should implement the following Java interface:

```
import org.pasoa.prep.service.BackendStore;
import org.w3c.dom.Document;
public interface PlugIn
{
public Document process (Document soapBody,   BackendStore store);
}
```

This interface means that a plug-in takes an XML document (an org.w3c.dom.Document Object) and a BackendStore reference as parameters and returns a W3C Document Object as a result. The BackendStore parameter is a reference to the data repository to which the Plug-In the data physically writes. There are currently two plug-ins implemented by the provenance store. One is for handling recording requests; the other one is for basic queries on already stored provenance information.

The third layer is the BackendStore where p-assertions are finally stored. Because a set of physical devices can be used as Backend storing device and due to the need for a common interface to interact with second layer, developers of PreServ have defined a common interface named BackendStore. BackendStore itself currently implements a set of other interfaces which represent the abilities of interfaces. These are the Record, Retrieval, Cache and Management interfaces.

```
public interface BackendStore extends Record, Retrieval, Cache, Management
{
}
public interface Record
{
    public static final String INTERACTION_PA      = "interactionPAssertion";
    public static final String ACTOR_STATE_PA      = "actorStatePAssertion";
    public static final String RELATIONSHIP_PA     = "relationshipPAssertion";
    public static final String SUBMISSION_FINISHED = "submissionFinished";
    public void record (GlobalPAssertionKey gpid, Element asserter, Element assertion, String kindOfAssertion)
throws Exception;
}
public interface Retrieval
{
    /**
```

```
     * Retrieve the message exchange at the given index
     */
    InteractionRecord getInteractionRecord (long index) throws Exception;
        /**
     * Get the number of message exchanges available.
     */
    long getNumberOfInteractionRecords () throws Exception;
}
```

Having a common interface for backend devices makes it possible to implement plug-ins without regarding the backend device actually being used. That has a drawback such as most of the common interfaces have. With such an architecture it is not possible to use additional

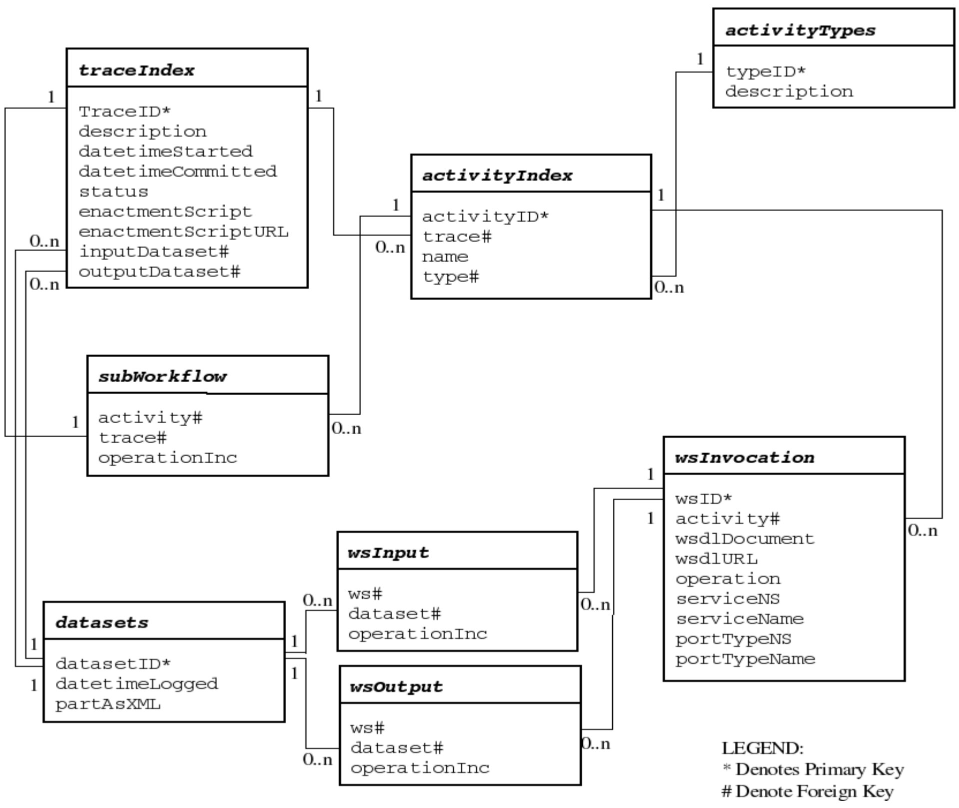


Figure 9: PreSev Data Model mapped to Backend Device

capabilities that a backend device might have. As an example, if a backend device is a relational database, the common interface does not offer a way of using sql queries.

The PreServ package has three default implementations of backend layers: in-memory, file system and database.

2.2.1.3 PreServ: Usage

There are three ways of using a Provenance Store Service through the WSDL definition (see A. 2 Sample Process Documentation Record WSDL for PASOA). A developer can call the PS web service directly or can use the Java Client Side Library to communicate. Even the Axis Handler for client side sub generation can be used.

Using through Direct Web Service Calls

This is the most basic and intuitive. However, this is the most complicated way of using PS. To call PS web services directly, one should develop all SOAP messages himself which should be compatible with WSDL interfaces of PS. This way, any implementation language can be used with PS.

Pros: Language independent, maximum flexibility

Cons: Very difficult and costly to develop

The Java Client Side Library

The PreServ package includes an API for developers who are using Java. By using the PreServ Client Side Library one can easily produce soap messages. Corresponding methods are included in API.

Pros: Easy development

Cons: Restricted to Java

The Axis Handler

This option can only be used of course if applications are implemented using the Axis Web Services libraries. The Axis Handler Jar file should have classpath added to the application. All SOAP messages produced or received by a web service client or service will be recorded in a provenance store specified in a configuration file. Essentially, the Axis Handler wraps the specified client and service and intercepts all web service communication. The Axis Handler is best used for already existing Axis based web service applications.

Pros: Can be used without modifying existing applications; easy development

Cons: Restricted to Java and dictates the use of Axis; no direct support for actor state p-assertions and recording relationships

2.2.2 GriPhyN Virtual Data System (*formerly Chimera*)

Differing from PreServ, the GriPhyN Virtual Data System manages the derivation and analysis of data objects in collaborator environments and collects provenance in the form of data derivation steps for datasets [4]. As defined in [19], the target of GriPhyN VDS developers was to be able to track how data products are derived with sufficient precision that one can create and/or re-create data products from this knowledge. Chimera is a product of this group, developed for exploring the benefits of data derivation tracking and virtual data management.

The terms *dataset*, *transformation*, *derivation*, and *invocation* have special meanings in the Chimera software platform. A *dataset* is used for a unit of data managed by VDS, and is associated with a type. A *transformation* is a typed definition for a computation procedure,

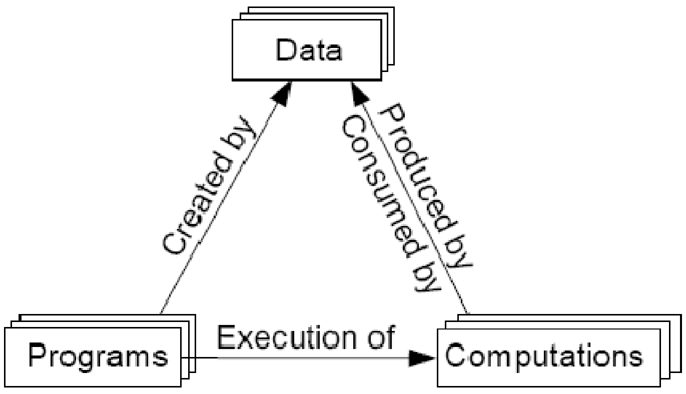


Figure 10: Interaction between Data, Programs and Computations

which takes certain types of datasets as inputs and outputs. A *derivation* initializes a transformation by giving the actual arguments and other necessary information. An *invocation* is the actual execution of a derivation [20].

Some annotation information is necessary to identify the transformations, derivations and invocations. These data are stored as attached metadata to each entity in the corresponding physical provenance store. Spoken to *transformation*, this information is information that can be useful for characterizing or locating the procedures (e.g. author, version, and cost) and information that is needed for invoking the procedure (like executable name, location, arguments, and environment).

A *derivation* actually represents an execution of a transformation and is responsible to initialize them with actual arguments if needed. In order to be able to initialize a transformation, a derivation needs to know the name of the associated transformation, the names of datasets (data objects) to which the transformation is applied and other derivation–specific information. Derivation-specific information can be values for parameters and the time that the transformation executed. Moreover, it is important to emphasize that the arguments which are possibly documented in the transformation metadata are only formal parameters, but that the parameters in the derivation metadata are the actual ones.

Datasets are the data which may be consumed by derivations. These are data objects which are already present in the physical storage. However, datasets can be generated by derivations as well. The execution of transformations (derivations) generates output data which refer to datasets in VDC. Typical dataset types are logical files, objects or relations.

These are entities of interest in the Chimera Virtual Data Schema concept. But how can the transformation information be captured?

In Chimera VDS it is possible to declare transformation or derivation-information manually by the user or to extract this information automatically from a job control language, produced by higher-level job creation interfaces such as portals, and/or created by monitoring job execution facilities and file accesses [19].

2.2.2.1 Chimera: Usage

To be able to use Chimera, applications have to use a language called Virtual Data Language. VDL interacts with an interpreter called *virtual data language interpreter* and supports both querying and populating the provenance data which is actually stored in the database as entries of the Virtual Data Catalog.

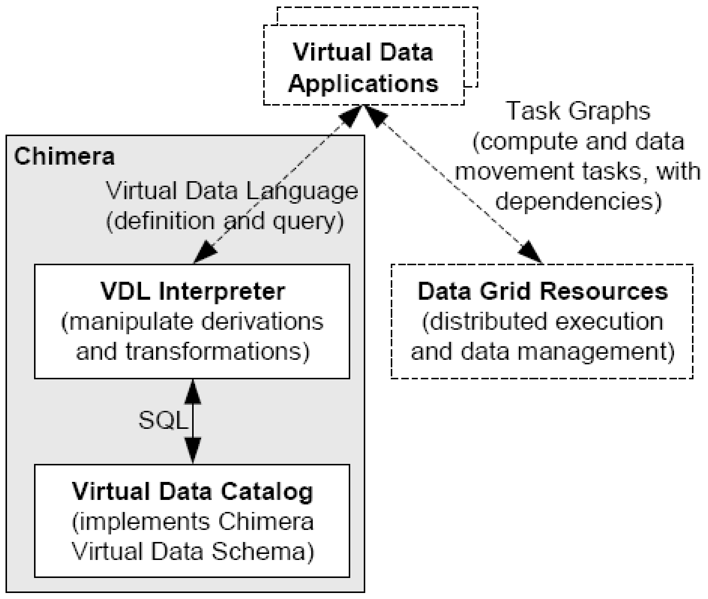

Virtual Data Applications
Task Graphs (compute and data movement tasks, with dependencies)
Chimera
Virtual Data Language (definition and query)
VDL Interpreter (manipulate derivations and transformations)
SQL
Virtual Data Catalog (implements Chimera Virtual Data Schema)
Data Grid Resources (distributed execution and data management)

Figure 11: Chimera Architecture Overview

The Virtual Data Catalog bases on a certain Virtual Data Schema (see Figure 12: UML Description of the Chimera virtual data schema). The Chimera VDS defines relations for capturing descriptions of how a program can be invoked, and to record its potential or actual invocations.

VDL supports both querying and populating the database. Querying means that the data are retrieved from the physical data store where populating can be seen as deleting and updating the present data.

Two types of statements are supported by the Chimera VDL: data definition and query statements.

There are two types of data definition statements: a transformation statement and a derivation statement. The definition statements are actually represented in the XML format in the Chimera software platform but I will give examples in a more readable syntax. The syntax for a transformation definition looks like:

```
TR <name> (
    output <parameter_name>, input <parameter_name>, none <tr_argument>
    ) {
    app <app_name> = <app_path>;
```

```
        arg <arg_name>  = <arg_value>;
        profile <environment_variable = <new_value>;
}
```

<name> in the first line is the name that is assigned to the transformation statement which is used later on in the derivation statement to identify the transformation. The TR header line consists of an input, output and argument definition. Input and output have to have parameter names and transformation arguments are name-value pairs such as:

```
none a_varible_name = “chimera”
```

TR headers may have more than one input/output definition and more than one TR argument pair.

There are three arguments defined in the body: *app*, *arg*, *profile*. *app* statements define a name for the executable and the path for it. A transformation consists of understandably only one executable application and these are potentially logical files. So *app_path* is the location where this executable (actually a LFN) is present.

arg statements describe the arguments/calculation options of the procedure to be called. There are some standard arguments which have special meanings in VDL such as *stdout* which means that standard outputs are redirected to a specified file.

The *profile* statement is a special statement to specify a default value for an environment variable for execution session.

A real-world example transformation definition statement looks like:

```
TR transformation_example (
output o1, input i1,none var1="100000",none var2="500"
) {
app blast = "/usr/bin/executable";
arg parg = "-p "${none:var1};
arg farg = "-f "${input:i1};
arg stdout = ${output:o1};
profile env.MAXMEM = ${none:var2};
}
```

The *arg* statement consists of name-value pairs. Here, in lines 5 and 6, one can see that the value part has a pre-defined default value which refers to transformation variables. These variables are to be replaced at invocation time.

The other data definition statement is a derivation (DV) statement which records the transformation invocations. The syntax for a DV statement:

```
DV <transformation_name> {
<TR_header_outputParameterName> = @{output:<actual_file_path>},
<TR_header_inputParameterName> = @{input:<actual_file_path>},
<TR_header_argParameter> = <value>
}
```

As can be seen here, derivation statements have no name to identify itself in the Chimera Data Catalog. They are to be found by searching their attributes.

The transformation name after the DV keyword defines which transformation this derivations definition invokes. The formal parameters in the transformation header and the derivation body are associated by the names used in the TR header (see Table 1).

TR <name>	DV <transformation_name>
(	{
output <parameter_name>,	<TR_header_outputParameterName> = ...
input <parameter_name>,	<TR_header_inputParameterName> = ...
none <tr_argument>	<TR_header_argParameter> = ...
)	}

Table 1 : Usage of formal parameters in TR header in DR statements

An actual derivation statement which uses transformation *transformation_example* from earlier could be:

```
DV transformation_example (
o1=@{output:analysis1.run1.result},
i1=@{input: analysis1.run1.input},
var1="20000",
var2="600"
)
```

The datasets (e.g. analysis data) and executions can be described as TR and DR statements and can be stored in Chimera Virtual Data Catalog, which enables the description of data/process lineage. The question arises of how a workflow execution should be recorded and how a data dependency chain according to this workflow should be expressed and stored in a VDC.

Chimera VDL supports the workflow definitions through compound transformations. Data dependency chains can be expressed using the feature that an output of a derivation can be the input of another one. The Foster's examples [21] for both of them can be found in Appendix B.

The second type of statements which is supported by Chimera was query statements to access the present information in Chimera VDC.

Since VDL is implemented in SQL [19] and a virtual data model is defined in relational terms [22], SQL can be used to query entities in the virtual data catalog. Being able to use SQL as query language makes querying easily extensible.

Transformations can be searched by the transformation name, application name, input LFN(s), output LFN(s), argument matches, and by other transformation metadata. Derivations can be found by searching for the associated transformation name, application name, input LFN(s), and output LFN(s).

Zhao [22] separates Chimera's query mechanism into three dimensions and discusses them from another point of view. He shows how the VDL queries can be used to gather different kind of information from VDC with examples.

2.2.2.2 Chimera: Data Schema

The Chimera Virtual Catalog bases on the Chimera Data Schema. Due to this fact I will go further into detail on this schema. See *Figure 12: UML Description of the Chimera virtual data schema* for UML representation of the Schema. This schema and its description are mostly based on the information on Foster's paper [19] which is the very first publication of the VDS.

A logical transformation is identified by its identifying name, the namespace within which the name is unique, and a version number. The signature of the transformation includes input and output parameters, which need not be files. A transformation may have an arbitrary number of formal arguments. The relationship between *Transformation* and *FormalArg* is 1:N. A transformation may have more than one derivation, each supplying different values for the parameters. A derivation may be applicable to more than one transformation. An *ActualArg* relates to a derivation. Its value is either a LFN or the value of a non-file parameter. A *FormalArg* may contain an optional default value, captured in a similar fashion by the same *Value* class. The *Value* class is an abstract base class for either a single value (*Scalar*) or a list of similar values (*List*), which are collapsed union-fashion into a single table.

The relationships between a transformation and its formal parameters, on the one hand, and a dependent derivation and its actual parameters, on the other, are not independent of each other. Each instantiation of an actual parameter maps to exactly one formal parameter describing the entry. The binding is created using the argument name, not its position in the argument list.

Scheduler and runtime environment-specific data is abstracted in the *PROFILE* table. For example, in the case of a UNIX environment variable, the namespace is "env", the key within this namespace is the environment variable name, and the value is a list of fragments, either references to bound variables or textual strings. The *FRAGMENT* table captures three child classes, a textual string, a LFN or a reference to a bound variable. The three child classes are collapsed into a single table.

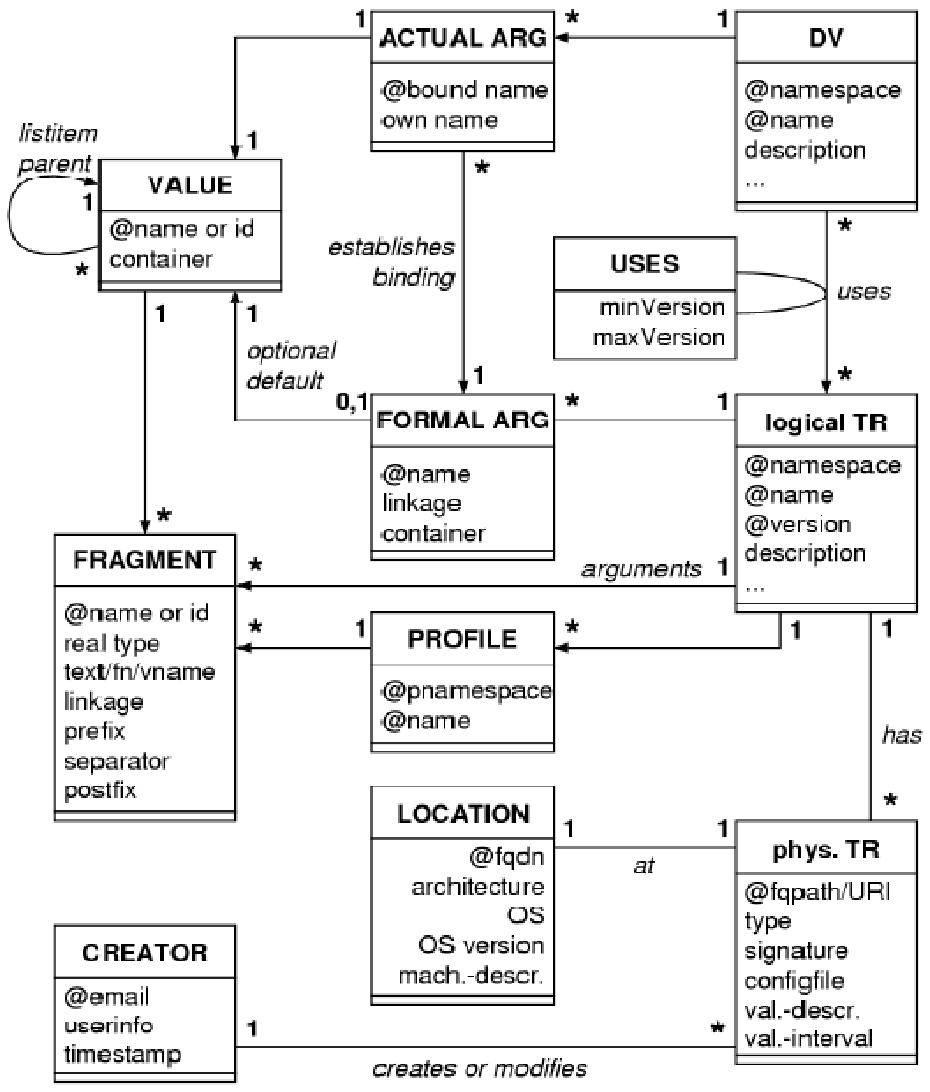


Figure 12: UML Description of the Chimera virtual data schema

Chapter 3

3 Current Workflow Based Solutions

3.1 Introduction

This section introduces two workflow based analysis software solutions which are currently being used by researchers in the life sciences.

The workflow based solutions in the first place have the aim of enabling scientists to gather information from different resources and build analysis pipelines as workflows.

3.2 Taverna

Taverna is a collaborative project led by European Bioinformatics Institute (EBI). Different universities, especially the universities from Britain participate in the project and perform development efforts with various individuals across the world.

Taverna is an open source project hosted on Sourceforge.net [23]. Sourceforge.net facilities and makes the development coordination easy for the collaborating institutions. Besides, Taverna is available under the terms of GNU Lesser General Public License (LGPL) [24] which means that one can modify the code and reuse the framework under certain circumstances.

Moreover, Taverna is funded through the Open Middleware Infrastructure Institute UK (OMII-UK) [25] and is offered in the OMII-UK software package.

Next, I will briefly introduce the Taverna software and show how to use it by executing an example workflow. Then, I will investigate the data management and provenance facilities of Taverna.

3.2.1 Working with Taverna

Taverna is a standalone software written with Java; therefore it needs at least JRE 1.5 or a higher version to run. The software version I am using for this work is v1.5.2 running with JRE 1.6 on the Windows x86 Platform.

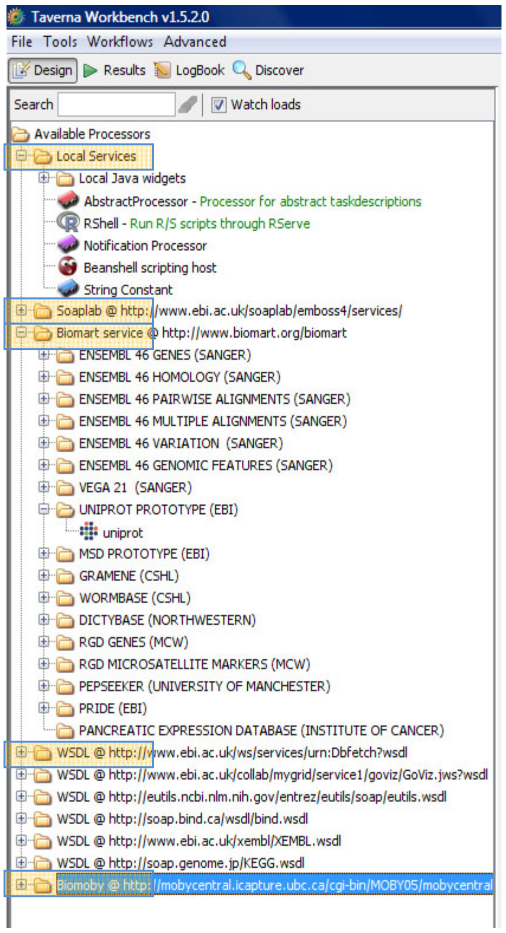
As mentioned in the introduction of this chapter, Taverna is a workflow based application which allows bioinformaticians to construct pipelines from very different research areas. Taverna offers a large set of functions to construct such pipelines. It has four types of functions (see Figure 13):

- *Local Services:* Local Services are functions written in Java and available as Java-Libraries or written in R and can be run using the embedded R-Shell caller or they can even be command-line tools written in any other programming language.
- *Soaplab tools*: Soaplab is a tool developed within the myGrid project to offer command-line tools to Taverna users from all over the planet as soap based web services. So, *Soaplab* can automatically generate and deploy web services on top of existing command-line analysis programs.
- *Biomart services:* The Biomart system [26] is a flexible data warehouse which contains complex interlinked biological data sets from different databases. Taverna's Biomart service provides full search and retrieval functionality over these data sources.
- *Biomoby services:* The Biomoby system defines an ontology-based messaging standard through which a client will be able to automatically discover and interact with task-appropriate biological data and analytical service providers, without requiring manual manipulation of data formats as data flow from one provider to the other [27].

- *Other web services:* Other tools provided by different institutions as soap based web services.

In order to construct a workflow from these functions, Taverna has a component called Advanced Model Explorer. Using this component, the user can add functions to the currently modeling workflow, add inputs to workflow and assign it to appropriate function in the list, and redirect the results of a process to another one as input. See Figure 14 for an example workflow.

On the right-hand side of the Advanced Model Explorer is a Workflow Diagram-Window where the workflow is shown as a graphic during the user's construction of the pipeline. Figure 14 shows an example workflow diagram.

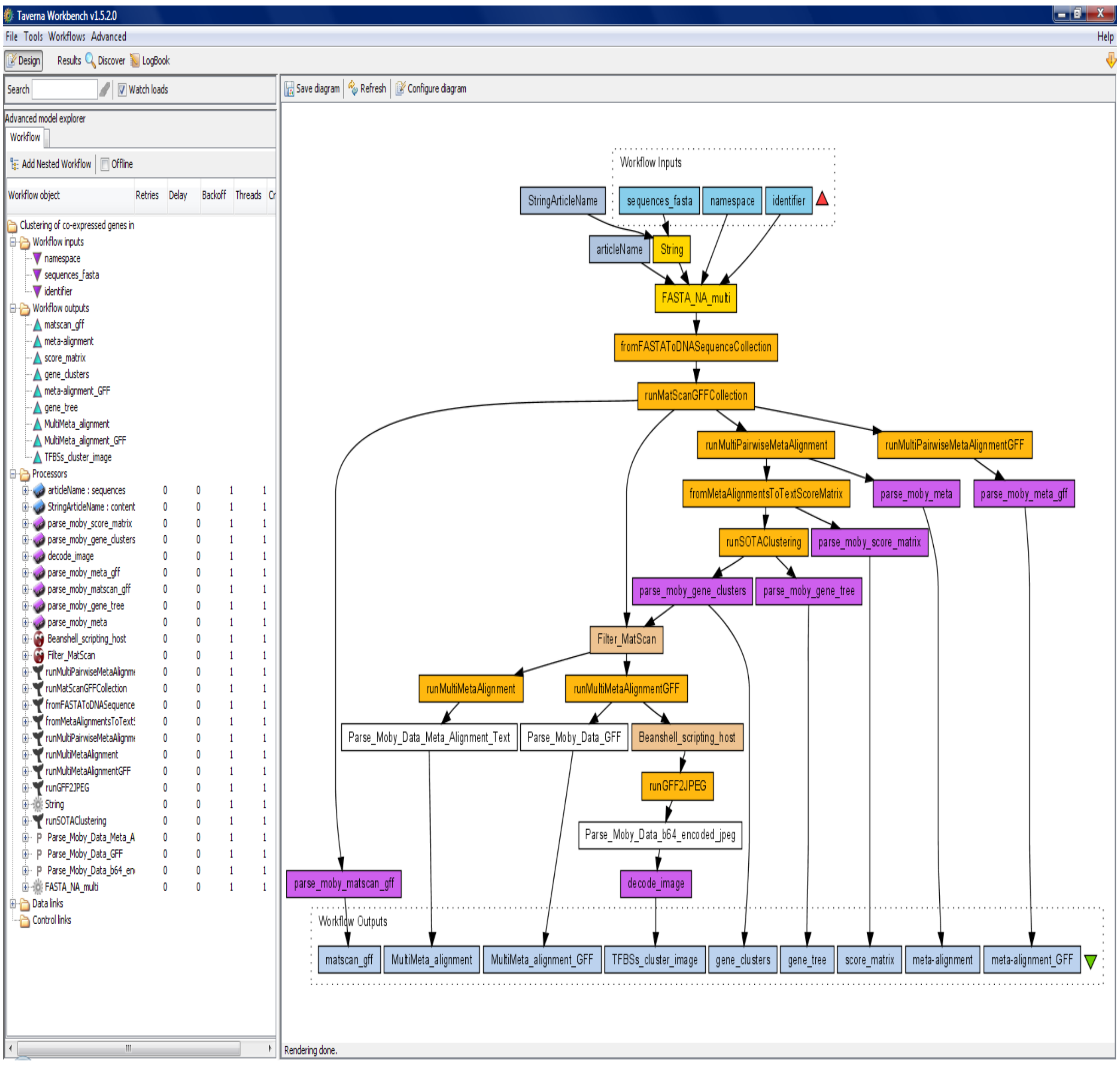

Taverna Workbench v1.5.2.0
File Tools Workflows Advanced
Design Results LogBook Discover
Search Watch loads

```
Available Processors
  Local Services
    Local Java widgets
    AbstractProcessor - Processor for abstract taskdescriptions
    RShell - Run R/S scripts through RServe
    Notification Processor
    Beanshell scripting host
    String Constant
  Soaplab @ http://www.ebi.ac.uk/soaplab/emboss4/services/
  Biomart service @ http://www.biomart.org/biomart
    ENSEMBL 46 GENES (SANGER)
    ENSEMBL 46 HOMOLOGY (SANGER)
    ENSEMBL 46 PAIRWISE ALIGNMENTS (SANGER)
    ENSEMBL 46 MULTIPLE ALIGNMENTS (SANGER)
    ENSEMBL 46 VARIATION (SANGER)
    ENSEMBL 46 GENOMIC FEATURES (SANGER)
    VEGA 21 (SANGER)
    UNIPROT PROTOTYPE (EBI)
      uniprot
    MSD PROTOTYPE (EBI)
    GRAMENE (CSHL)
    WORMBASE (CSHL)
    DICTYBASE (NORTHWESTERN)
    RGD GENES (MCW)
    RGD MICROSATELLITE MARKERS (MCW)
    PEPSEEKER (UNIVERSITY OF MANCHESTER)
    PRIDE (EBI)
    PANCREATIC EXPRESSION DATABASE (INSTITUTE OF CANCER)
  WSDL @ http://www.ebi.ac.uk/ws/services/urn:Dbfetch?wsdl
  WSDL @ http://www.ebi.ac.uk/collab/mygrid/service1/goviz/GoViz.jws?wsdl
  WSDL @ http://eutils.ncbi.nlm.nih.gov/entrez/eutils/soap/eutils.wsdl
  WSDL @ http://soap.bind.ca/wsdl/bind.wsdl
  WSDL @ http://www.ebi.ac.uk/xembl/XEMBL.wsdl
  WSDL @ http://soap.genome.jp/KEGG.wsdl
  Biomoby @ http://mobycentral.icapture.ubc.ca/cgi-bin/MOBY05/mobycentral
```

Figure 13: Taverna function list

In order to demonstrate the usage I will execute a workflow from the INB Bioinformatics and Genomics Node [28]. The workflow I will use is a workflow intended to use for Promoter analysis. It can be downloaded at [29] and the dataset at [30]. The Diagram of example Workflow is presented in Figure 14.

After downloading the workflow definition file (.xml file) written in the XScufl [31] web service workflow language and the example dataset from INB web site, the execution can be started. Pressing File > Open Workflow command and pointing the downloaded workflow definition file by using the file manager, opens prepared workflows for the execution. After that, the File > Run Workflow command is to be applied. This makes Taverna show the input definition window for entering workflow inputs (see Figure 16). Since Taverna offers to pack inputs in an XML file and the INB has prepared example datasets in that format, we load downloaded dataset files in Taverna by pressing "Load Input Doc" button and pointing the downloaded file. Finally pressing the "Run Workflow" button makes the current window disappear. Then, the main window gets focussed but with another view: the “Results perspective”. The results perspective is displayed during and after the execution. During the execution the window has two parts: a list of



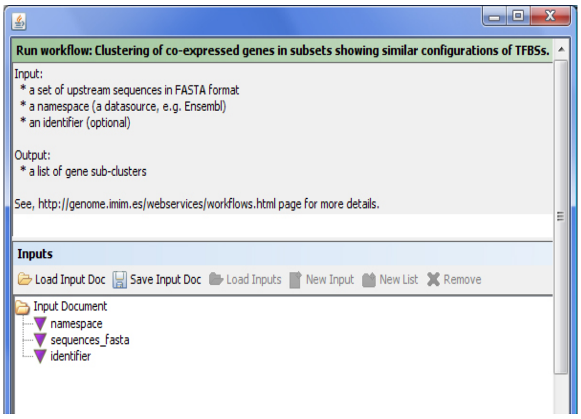
Workflow object	Retries	Delay	Backoff	Threads
Clustering of co-expressed genes in				
Workflow inputs				
namespace				
sequences_fasta				
identifier				
Workflow outputs				
matscan_gff				
meta-alignment				
score_matrix				
gene_clusters				
meta-alignment_GFF				
gene_tree				
MultiMeta_alignment				
MultiMeta_alignment_GFF				
TFBSs_cluster_image				
Processors				
articleName : sequences	0	0	1	1
StringArticleName : content	0	0	1	1
parse_moby_score_matrix	0	0	1	1
parse_moby_gene_clusters	0	0	1	1
decode_image	0	0	1	1
parse_moby_meta_gff	0	0	1	1
parse_moby_matscan_gff	0	0	1	1
parse_moby_gene_tree	0	0	1	1
parse_moby_meta	0	0	1	1
Beanshell_scripting_host	0	0	1	1
Filter_MatScan	0	0	1	1
runMultiPairwiseMetaAlignme	0	0	1	1
runMatScanGFFCollection	0	0	1	1
fromFASTAToDNASequence	0	0	1	1
fromMetaAlignmentsToText	0	0	1	1
runMultiPairwiseMetaAlignme	0	0	1	1
runMultiMetaAlignment	0	0	1	1
runMultiMetaAlignmentGFF	0	0	1	1
runGFF2JPEG	0	0	1	1
String	0	0	1	1
runSOTAClustering	0	0	1	1
Parse_Moby_Data_Meta_A	0	0	1	1
Parse_Moby_Data_GFF	0	0	1	1
Parse_Moby_Data_b64_en	0	0	1	1
FASTA_NA_multi	0	0	1	1
Data links				
Control links				

Figure 14: Taverna Workbench: On the left is the Advanced Model Explorer, on the right is an example workflow diagram

processes that are scheduled to run and an interactive workflow diagram showing which processes have been completed and which are currently being executed. One can see the intermediate inputs and intermediate outputs as well. For an example graphic showing the “Results Perspective” during execution of an example workflow, see Figure 17: Taverna Results Perspective.

After execution finishes, Taverna adds two tabs to the "Results Perspective" window: *Results,* and *Process Report.* All workflow outputs are shown in the "Results" tab (see Figure 15). It contains a Tabbed Pane where the results are shown as tabs. Finally, the "Process Report" tab has an XML file as workflow report.

For various file types there are renderers in Taverna. That means, file types are assigned to certain renderers which are responsible for displaying the according type of file or data.


Run workflow: Clustering of co-expressed genes in subsets showing similar configurations of TFBSs.

Input:
* a set of upstream sequences in FASTA format
* a namespace (a datasource, e.g. Ensembl)
* an identifier (optional)

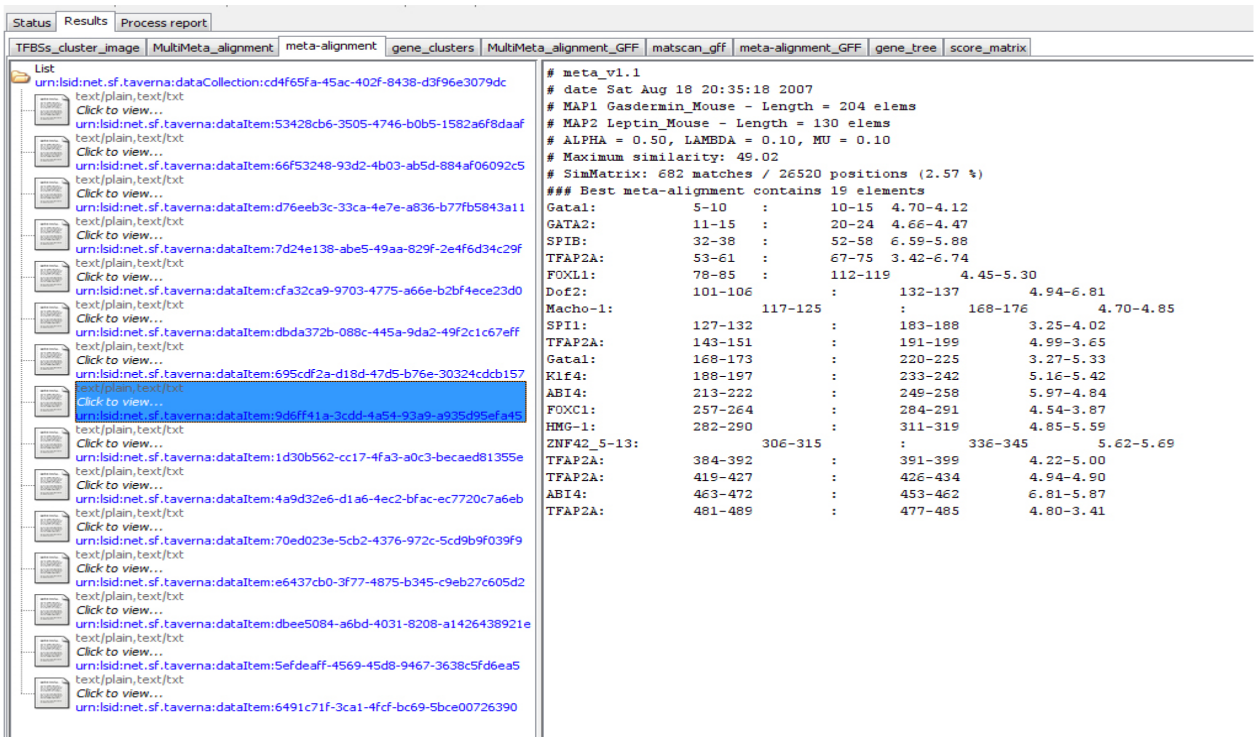
Output:
* a list of gene sub-clusters

See, http://genome.imim.es/webservices/workflows.html page for more details.

Figure 16: Input Window for defining Workflow Inputs

Finally, the produced output can be exported and saved to a physical drive as:

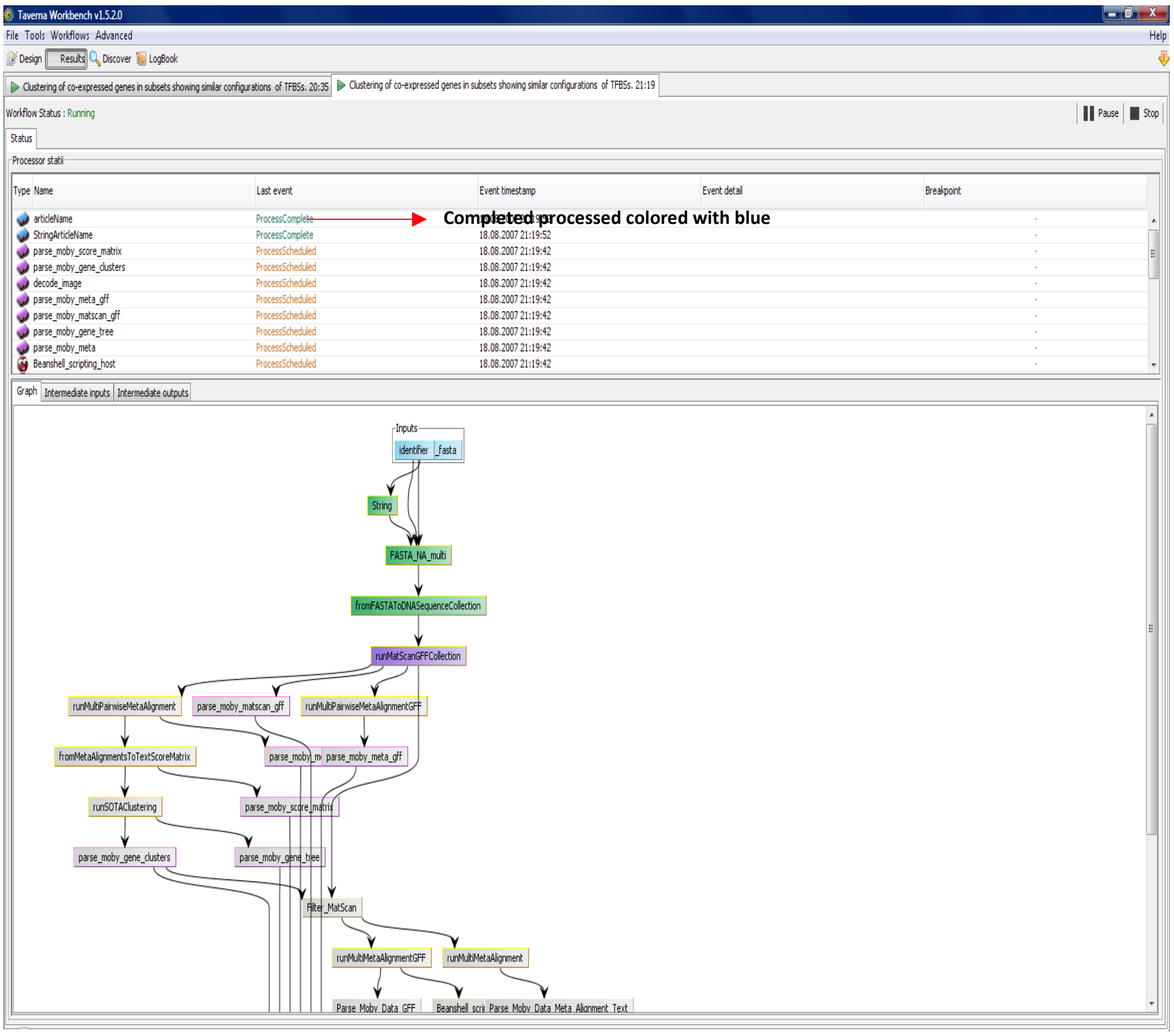
- An Excel document,
- Html files as if the results were an offline web site



```
List
urn:lsid:net.sf.taverna:dataCollection:cd4f65fa-45ac-402f-8438-d3f96e3079dc
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:53428cb6-3505-4746-b0b5-1582a6f8daaf
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:66f53248-93d2-4b03-ab5d-884af06092c5
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:d76eeb3c-33ca-4e7e-a836-b77fb5843a11
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:7d24e138-abe5-49aa-829f-2e4f6d34c29f
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:cfa32ca9-9703-4775-a66e-b2bf4ece23d0
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:dbda372b-088c-445a-9da2-49f2c1c67eff
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:695cdf2a-d18d-47d5-b76e-30324cdcb157
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:9d6ff41a-3cdd-4a54-93a9-a935d95efa45
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:1d30b562-cc17-4fa3-a0c3-becaed81355e
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:4a9d32e6-d1a6-4ec2-bfac-ec7720c7a6eb
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:70ed023e-5cb2-4376-972c-5cd9b9f039f9
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:e6437cb0-3f77-4875-b345-c9eb27c605d2
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:dbee5084-a6bd-4031-8208-a1426438921e
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:5efdeaff-4569-45d8-9467-3638c5fd6ea5
    text/plain,text/txt
    Click to view...
    urn:lsid:net.sf.taverna:dataItem:6491c71f-3ca1-4fcf-bc69-5bce00726390
```

```
# meta_v1.1
# date Sat Aug 18 20:35:18 2007
# MAP1 Gasdermin_Mouse - Length = 204 elems
# MAP2 Leptin_Mouse - Length = 130 elems
# ALPHA = 0.50, LAMBDA = 0.10, MU = 0.10
# Maximum similarity: 49.02
# SimMatrix: 682 matches / 26520 positions (2.57 %)
### Best meta-alignment contains 19 elements
Gata1:          5-10    :       10-15   4.70-4.12
GATA2:          11-15   :       20-24   4.66-4.47
SPIB:           32-38   :       52-58   6.59-5.88
TFAP2A:         53-61   :       67-75   3.42-6.74
FOXL1:          78-85   :       112-119         4.45-5.30
Dof2:           101-106         :       132-137         4.94-6.81
Macho-1:                117-125         :       168-176         4.70-4.85
SPI1:           127-132         :       183-188         3.25-4.02
TFAP2A:         143-151         :       191-199         4.99-3.65
Gata1:          168-173         :       220-225         3.27-5.33
Klf4:           188-197         :       233-242         5.16-5.42
ABI4:           213-222         :       249-258         5.97-4.84
FOXC1:          257-264         :       284-291         4.54-3.87
HMG-1:          282-290         :       311-319         4.85-5.59
ZNF42_5-13:             306-315         :       336-345         5.62-5.69
TFAP2A:         384-392         :       391-399         4.22-5.00
TFAP2A:         419-427         :       426-434         4.94-4.90
ABI4:           463-472         :       453-462         6.81-5.87
TFAP2A:         481-489         :       477-485         4.80-3.41
```

Figure 15: Results tab shown after the execution finishes


Taverna Workbench v1.5.2.0
File Tools Workflows Advanced
Help
Design
Results
Discover
LogBook
Clustering of co-expressed genes in subsets showing similar configurations of TFBSs. 20:35
Clustering of co-expressed genes in subsets showing similar configurations of TFBSs. 21:19
Pause
Stop
Status
Processor stati
Graph
Intermediate inputs
Intermediate outputs
Inputs
identifier
_fasta
String
FASTA_NA_multi
fromFASTAToDNASequenceCollection
runMatScanGFFCollection
runMultiPairwiseMetaAlignment
parse_moby_matscan_gff
runMultiPairwiseMetaAlignmentGFF
fromMetaAlignmentsToTextScoreMatrix
parse_moby_meta_gff
runSOTAClustering
parse_moby_score_matrix
parse_moby_gene_clusters
parse_moby_gene_tree
Filter_MatScan
runMultiMetaAlignmentGFF
runMultiMetaAlignment
Parse_Moby_Data_GFF
Parse_Moby_Data_Meta_Alignment_Text

Workflow Status : Running

Type	Name	Last event	Event timestamp	Event detail	Breakpoint
	articleName	ProcessComplete	[illegible]		
	StringArticleName	ProcessComplete	18.08.2007 21:19:52		
	parse_moby_score_matrix	ProcessScheduled	18.08.2007 21:19:42		
	parse_moby_gene_clusters	ProcessScheduled	18.08.2007 21:19:42		
	decode_image	ProcessScheduled	18.08.2007 21:19:42		
	parse_moby_meta_gff	ProcessScheduled	18.08.2007 21:19:42		
	parse_moby_matscan_gff	ProcessScheduled	18.08.2007 21:19:42		
	parse_moby_gene_tree	ProcessScheduled	18.08.2007 21:19:42		
	parse_moby_meta	ProcessScheduled	18.08.2007 21:19:42		
	Beanshell_scripting_host	ProcessScheduled	18.08.2007 21:19:42		

Completed processed colored with blue

Figure 17: Taverna Results Perspective

- XML file.

Next, I will explain how Taverna manages the input/output data.

3.2.2 Data Management in Taverna

Taverna has been designed to construct web service based workflows and to execute them. An important point for consideration is who is the one constructing the workflow. Preparing a workflow can be very complex with control links, loops or even parallel executions.

With the Advanced Model Explorer Taverna declares to aim for the simplest possible solutions which can be used not only by experienced bioinformaticians but also by researchers who have only limited computer knowledge. Whether this aim has been accomplished with the current version of Taverna is to be discussed at some other point.

Hence, Taverna concentrated on making workflow construction easier, offering better and more functions to the user but not yet on the data management of analyses The researchers have to save their analysis data, inputs, outputs etc. on the physical drive and have to secure these data themselves.

Next, I am introducing a plug-in of Taverna which enables storing all the information on the performed analyses: *The LogBook*. From the executed workflow to the intermediate results; everything will be stored if the user wishes so. This should not be confused with data management since *The LogBook* is just a provenance tool that saves data automatically but does not allow the user to store data in a project management manner.

3.2.3 Provenance in Taverna

Taverna has no built-in provenance support but there is a myGrid component called *LogBook* written exclusively for Taverna. Users can download it separately at [32] and install or use Taverna's Plug-In Manager for searching existing Plug-Ins which offer an easy way of installing.

After the LogBook Plug-in has been installed, Taverna adds a new perspective to the main window. There, the user will be asked for MySQL connection data because the LogBook is configured to work exclusively with the MySQL database. After entering the required information to connect to the database, the user has to decide about the level of capturing provenance information. Taverna offers four detail levels:

- *Nothing*: Noting is captured; provenance capturing is off.
- *Workflow Inputs and Outputs*: Only workflow inputs and outputs are captured and the data produced at intermediate steps are ignored.
- *All intermediate steps, except iterations*: All processing steps and the input-output data with intermediate results are captured. Only iterations are ignored.
- *Everything*: All processing steps and the input-output data with intermediate results inclusively the iterations are captured.

Right after that, Taverna begins capturing provenance data depending on the selected detail level. Taverna uses RDF graphs for metadata recording. Each RDF graph is assigned an LSID as name and stored as a named RDF graph. Developers say that this allows technical recovering and referring to the entire graphs without having to resort to the expensive RDF reification mechanism. Taverna currently uses the NG4J (Named Graphs for Jena) API to store named graphs in a MySQL database [32].

As it captures metadata about executions, LogBook captures the original workflows, too. Users should first select a workflow for which they want to see the provenance details, and then a list of processes belonging to the selected workflow is shown in a list just below the workflow-list window. Furthermore, LogBook shows all independent executions of a workflow (runs), in a tree-like list whose root is the name of a certain workflow. That means users should select the workflow, expand the list and choose which execution's (run's) details they want to see. These runs are identified with a timestamp of the execution time.

Selecting desired runs makes LogBook open the list of processes (functions for workflow) which were successfully executed or failed during workflow execution. Besides, LogBook adds three additional tabs for the workflow inputs, the workflow outputs and for the workflow diagram.

By selecting a process from the list, LogBook shows two tabs at the bottom of the window: Intermediate Inputs and Intermediate Outputs. The calculation parameter used for the execution of the marked function and the data used for it are included in an XML file shown in the Intermediate tab. The intermediate results produced by the marked process are in the Intermediate Outputs in XML file format. The tabs other than the process list – the workflow inputs, the outputs and the workflow diagram - include data as in the results perspective. See 3.2.1: *Working with Taverna* for details.

3.3 Kepler

Kepler [33] is a community driven, open source project for the analysis and modeling of scientific data. It is a platform to build and execute workflows. Kepler represents the overall workflow visually so that it is easy to understand the data flow from one component to the other.

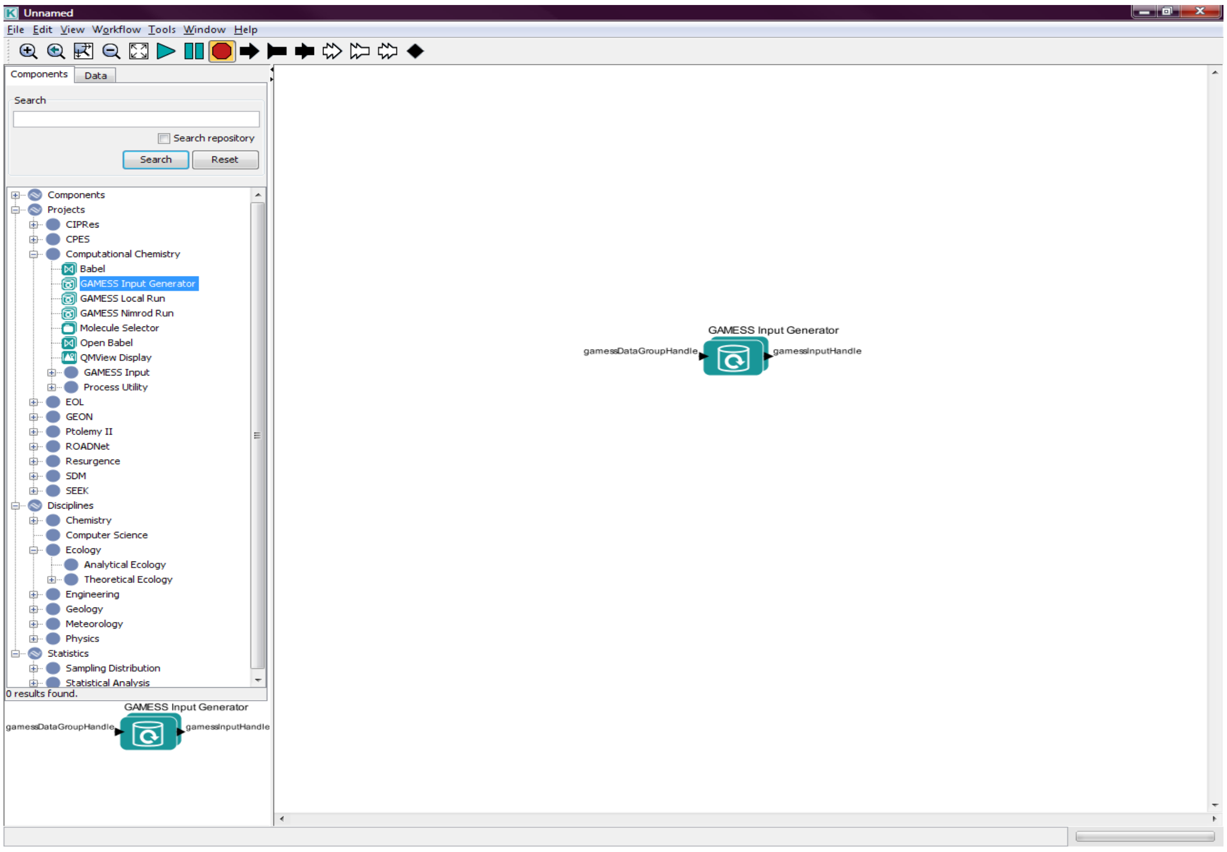


Figure 18: Kepler Main Window

Kepler allows scientists to create their own executable scientific workflows by dragging and dropping components into a workflow creation area and connecting the components to construct a specific data flow, or they can modify existing workflows to suit their needs. Quantitative analysts can use the visual interface to create and share R and other statistical analyses. Kepler is still not the optimum software for scientists with little bioinformatics background to create and modify their workflows; only advanced users can deal with such complex workflows.

Kepler is based on the Ptolemy II [34] system. The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on the assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interactions between the components. In Ptolemy, a system or model is viewed as a composition of independent components. In Kepler a model is a scientific workflow and workflows are composed of services called actors. Actors can be written in Java, web services, scripting languages like R, and database queries. Kepler supports nested workflows which are represented as composite actors.

Each actor in a workflow can contain one or more communication interfaces called ports, which are used to consume or produce data. Actors in a workflow are connected via their ports. The

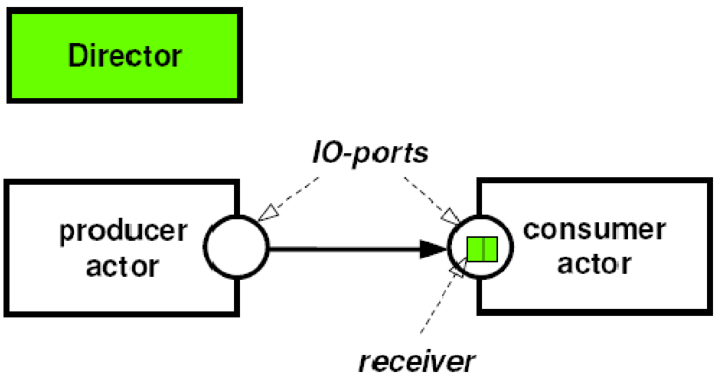


Figure 19: Components interaction in Kepler Workflow

links between two actor ports represent a dataflow and is called a channel. There are three types of Ports: input port, output port, and input/output port. Input and output ports are self-explanatory; an input/output port is needed for data both consumed and produced by the actor. Ports can be configured as “singular” or “multiple” ports. Singular port means that ports can be connected to only a single channel; multiple ports can be connected to multiple channels.

Actors can additionally have “parameters”. Parameters configure and customize the behavior of the actors, whereas inputs contain real data. For example an actor implementing a text search algorithm can have two input ports: one is the search phrase and the other one is a text in which this phrase is searched. Possible parameters for this generic search actor can be whether it is case sensitive or not.

Execution semantics in Ptolemy II are described by a director. A director controls the execution of a workflow; in other words, actors take their execution instructions from the director. Actors specify what processing occurs while the director specifies when, how it occurs and how the actors communicate with other actors in the workflow. Every workflow must have a director that controls the execution of the workflow using a particular execution model, which is called

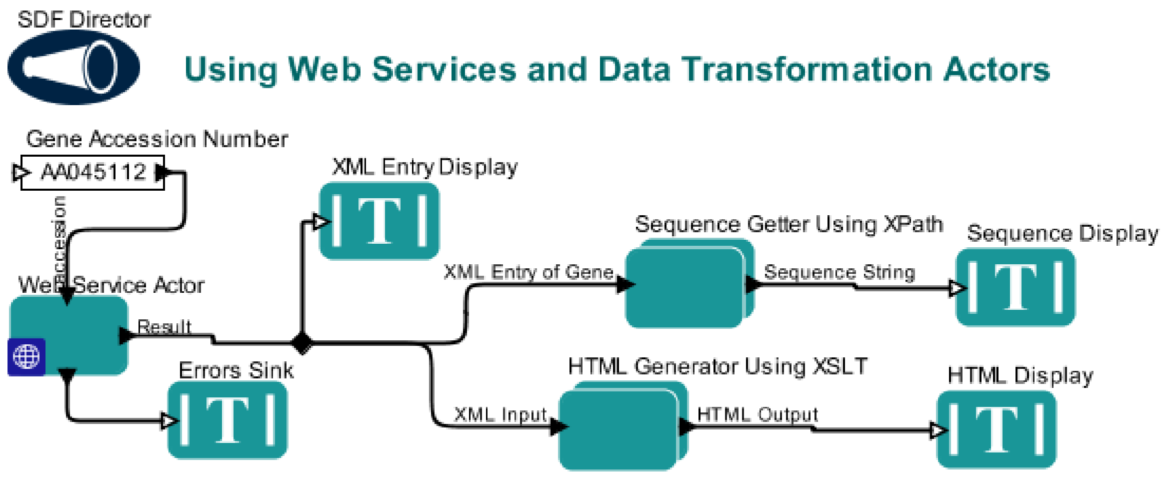


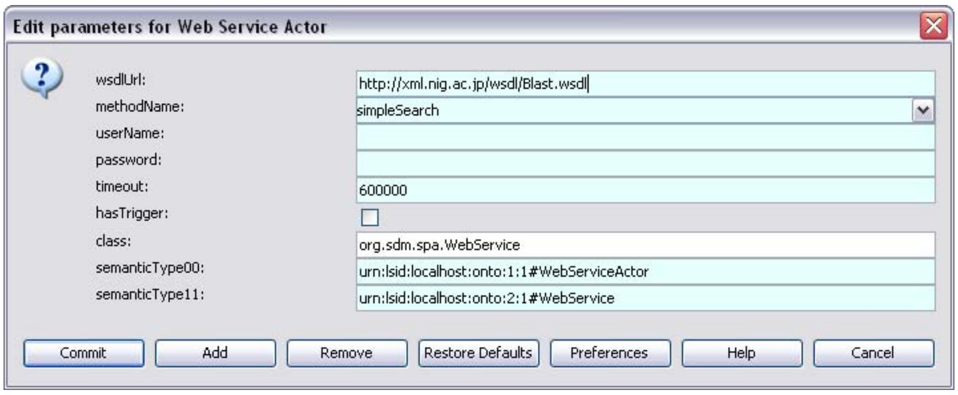
Figure 20: Sample workflow in Kepler

model of computation. Each model of computation in Kepler is represented by its own director. Workflow execution can be synchronous, with processing one component at a time or workflow components can run.

A small set of commonly used directors come pre-packed with Kepler. One of the popular directors used by workflow designers in Kepler is the Process Network (PN). In this computational model, actors are independent processes. Actors execute in parallel, each with its own thread of control. They communicate by sending tokens through unidirectional channels like UNIX pipes. Reading from unidirectional channels is blocked until input data are available; writing to a channel is non-blocking. This is similar to the execution of several processes in UNIX connected with pipes:

```
Program1 | Program2 | Program3
```

The Synchronous Data-Flow (SDF) Director enables processing of one component at a time in a pre-calculated sequence. A workflow controlled by the SDF director is a fairly simple sequence



wsdlUrl:	http://xml.nig.ac.jp/wsdl/Blast.wsdl
methodName:	simpleSearch
userName:	
password:	
timeout:	600000
hasTrigger:	☐
class:	org.sdm.spa.WebService
semanticType00:	urn:lsid:localhost:onto:1:1#WebServiceActor
semanticType11:	urn:lsid:localhost:onto:2:1#WebService

Figure 21: Kepler Web Service Actor Configuration

of operations. In such dataflow models, actors are invoked when their input data are available. The SDF director simply ensures that an actor fires after the actors whose output values it depends on. Another pre-packaged director in Kepler is the Discrete Event (DE) Director which is used for models where events occur at discrete times along a time line and one wishes to determine the average wait times or occurrence rates.

The workflow in Figure 20 demonstrates the use of the remote genomics data services to retrieve a genetic sequence. The sequence is then displayed in three different ways, first in its native format (XML), second as a sequence element that has been extracted from the XML format, and third as an HTML document that may be used for display on a web site.

Figure 21 shows a configuration window of a web service actor.

3.3.1 Workflow Definition Language

Kepler uses Ptolemy's own Modeling Markup Language (MoML). MoML is a modeling language for building models as parameterized, hierarchical components. MoML is kept very small by representing only the features of the abstract syntax. Entities with input/output-ports, connections and relations are the key elements of the MoML. The MoML does not say anything about the meaning of the components or the connections between components in the graph.

3.3.2 Web Services Support

Since many bioinformatics algorithms are exposed as web services, support for web services is a must for scientific workflow management systems. Therefore, a generic web service actor is implemented by Kepler to handle web services. The generic web service actor can be inserted into any workflow and accepts the URL of a WSDL file and the name of method defined by the WSDL (see Figure 21). Once the user has selected a WSDL and method name, the web service actor automatically configures itself by creating the necessary input and output ports. The web service actor invokes the web service and broadcasts the response through its output ports.

However, there is a certain limitation that only base types can be inputs or outputs to the web service actor. These base types are defined in the WSDL file.

3.3.3 Data Management in Kepler

Kepler has no data management features. As Taverna, Kepler focuses on creating workflows and executing them. This art of working places the workflow in the center of everything and everything is seen as a part of a workflow.

Kepler offers functionality to read from files and databases. That means that there are several local and remote tools which read resources and forward it to an input port or write data coming from an output port to some resources but it does not manage them (see Figure 22). Moreover, the term project has no meaning in Kepler's context.

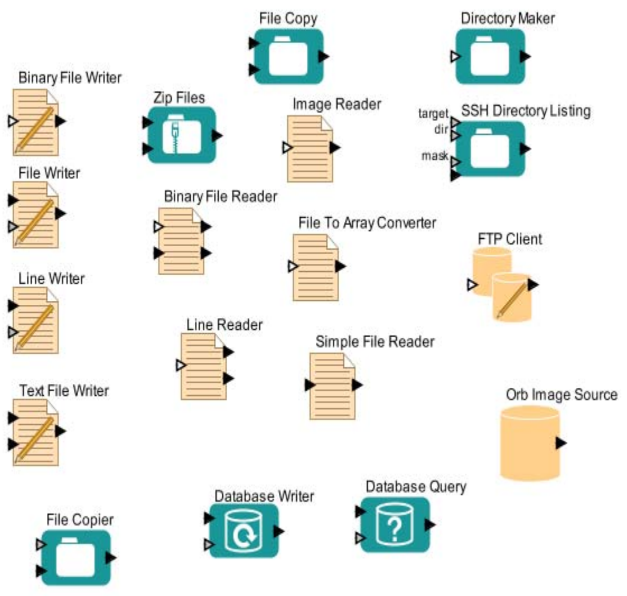


Figure 22: Different actors in Kepler for data forwarding

3.3.4 Provenance in Kepler

Just like data management, Kepler has no provenance support. Users have to take care about it themselves.

Chapter 4

4 PPMS

4.1 Introduction

The PPM system aims to answer the data management requirements of the life sciences community which is using tools for the analysis of their data.

In software engineering the term pipeline is used for a chain of processing elements arranged so that the output of each element is the input of the next. As this is often the case in terms of biomedical research, pipelines (a.k.a. workflows) based software is usually chosen in the mentioned domain.

A problem which is not mentioned so far is, that in the life sciences domain itself there are various research areas. This results in that the requirements of different research groups differ in terms of data management. Therefore, different user groups or research organizations need different ways of managing data. PPMS closes this gap by offering a flexible and extensible architecture which enables plugging in different project management styles.

The analysis tools used in the life sciences domain usually produce huge amount of data. These data are nowadays stored in local drives in a way contrived by researchers. The need for storing data in secured remote servers which are backed up continuously is enormous. Knowing that, the PPM system offers a back-end server which saves analysis data in an underlying database.

Communication between client and server is done using Soap based web services. This architecture choice was necessary since the life sciences domain uses uncountable different tools for the analysis. Beginning with HTML web sites written in Php; Perl, Shell Scripts, Ruby, JRuby, C, C++, Java and others are commonly used even if not as much as Java. Soap based web services enable the usage of PPMS services from all the tools written in different programming languages.

Users want to be able to analyze their data offline as well. While some want to be able to do this because there is no network connection at the very moment, others want this because of the overflow that network usage causes. Sometimes sending the data over the network costs time, so being able to work on a local drive is essential.

For all these requirements, PPMS offers a solution. This section introduces how and why PPMS solves these problems, as well as giving the architectural - implementation details by telling about the bottlenecks of the development.

4.2 Development Environment & Technologies

One of the essential things to know before starting to talk about the architecture details is the development environment and selected technologies in the specific fields. Since there are a lot of different implementations for a certain specification and there are very much different IDEs for developing in certain languages, it is often good to know which one is used in order to be able to guess the problems arising, the difficulties overcome and finally the parts that were possibly easy to develop.

Java has been used as the programming language. There were not a lot of choices at this point. The availability of present open source tools, the experience of the developing team in that programming language and the platform independency of the language were the key reasons of this decision.

J2EE specification version 1.5 is a major choice because of the bottlenecks which can arise due to the inexperienced team members, the community, and the lesser tool support. It still brings some convenience by the easy web service programming, EJB 3.0 with annotations, and persistence.

Because of the experience of the team members in the Netbeans IDE for Java [41] programming, the last stable version 6.0 is used. Besides that, the significant web service integration of the IDE was an essential decision point.

The other technologies are:

- *Glassfish V2*: Currently one of the best application servers that supports J2EE 1.5 specification.
- *Maven2*: For the automation used. It is necessary because of the open source tools used.
- *Toplink Essentials*: Reference implementation of the EJB 3.0 Java Persistence API
- *MySQL 5.1.11*: Database Server.
- *JAX-WS RI 2.1*: JAX-WS API's reference implementation is used because of the brilliant integration of the Netbeans IDE and the Glassfish application server.

Moreover, the server and the client are tested on a machine running with Windows Vista Home Premium x84 operating system on an Intel Core2 Duo 2.2 GHz CPU with a 2 GHz memory device.

4.3 Theoretical Background

Considering the problems described so far, it was clear that the solution had to have the two aspects of managing the data going around an analysis and of offering a mechanism for capturing provenance data to guarantee the reproducibility of the analyses.

Since the provenance data and the project data are partly the same, the two aspects of the solution should work together to prevent redundant data capture.

The term "project" is widely used in different softwares but each one has his own way of data management, his own constraints about which data it should store where. Each software has something like a project file in which the project information is stored.

However, the reviews made by me at the ARC Seibersdorf show that different research groups in different disciplines need different project constraints. Therefore, PPMS was developed in a flexible architecture which enables the easy adding of different project management styles. The project data are not stored twice but the provenance data captured behaved as if they were the project data. An exception to this one is input data imported to the project by the user.

4.3.1 Project Management Styles

As a result of the reviews, two project management styles are contrived.

The question asked within the scope of the review was “What is an analysis for you?” The most said “An analysis is the execution of a unique function a couple of times but with different parameters or with different inputs”. This is because sometimes they need to find the best parameter fits to a certain analysis and sometimes they need to analyze many different probes using a certain function. From this point of view, we contrived a Function Based Project Management.

The second most frequent answer was *“An analysis is the examination of a probe”.* This one is because in the medical research where researchers usually have a probe from a certain patient which has to be examined for a suspicion or illness. This usually requires the execution of a few different tools but sometimes only one tool is sufficient. The most important requirement here is that the researchers should be able to compile the path used for an analysis to a workflow so that they do not need to do the same thing again each time the same analysis is needed but are able to do it all with one mouse click. Another reason is that there are persons who are responsible only for use of certain devices which are used for analysis intentions and who do not know the theoretical background of the analysis. These technical persons shall be able to easily execute the pipeline, in best case only with one mouse click, without knowing the background. This led us to contrive an *Input Based Project Management.*

4.3.1.1 Function Based Project Management

Function Based Project Management (FBPM) is based on the idea that a function is used several times in order to find the best parameters suiting a certain analysis or a function are used many times for different inputs (probes) which are related to each other. Sometimes the result of the analysis has to be compared to find out medical results.

In function based management architecture the project has to have only one function. The analysis will be done using this function. As mentioned in the terminology part, while a function can be a basic executable or an R/Shell script, it can also be an already constructed complex workflow.

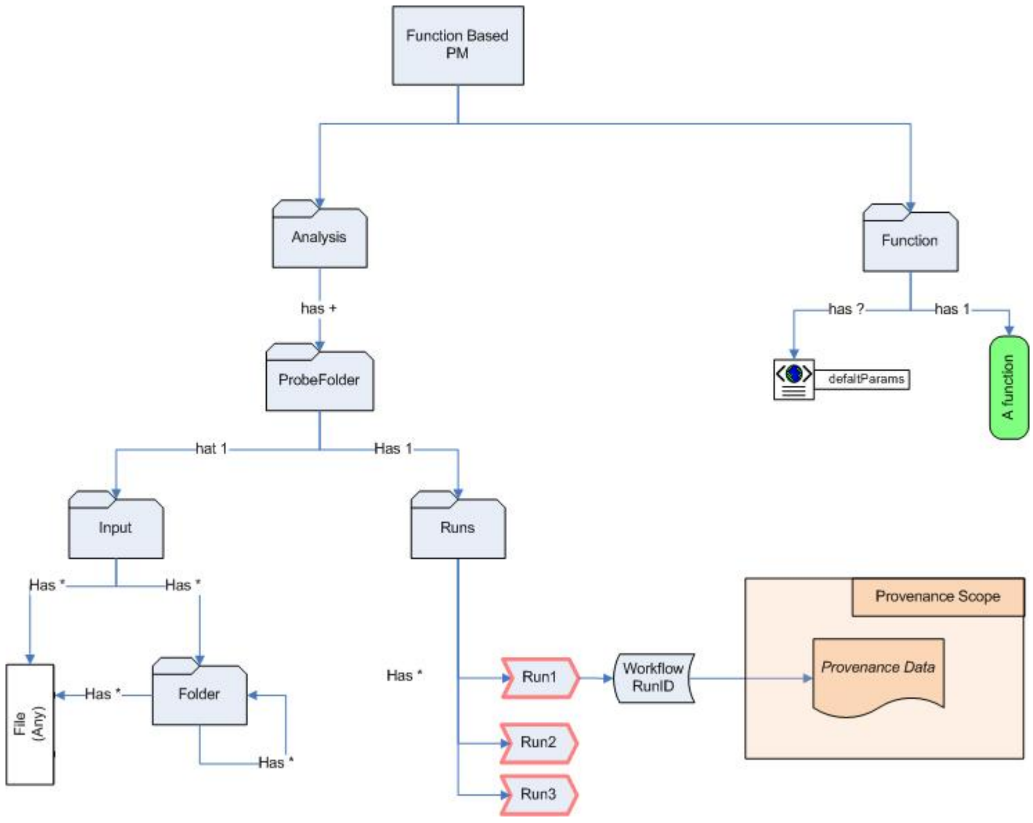


Figure 23: Function Based Project Management Structure

The function folder has a file in the XScufl [31] definition language representing the functions which can be invoked by FreeFluo [35] enactor. This can include web services, R Scripts, executables, and Java libraries. The Integrated Advanced Model Explorer lets the user construct such workflows using different tools and conditional parameters.

Function based projects can have as many probes as necessary; there are no limitations here. The container named “Analysis” is the folder in which the probes are stored. (see Figure 23) A “Probe Folder” has an input and a results container. Input containers can have folders and files in unlimited number. The file types are not restricted as well.

A Run node contains the information about the workflow run and thereby it represents a reference to the provenance information which is captured by the provenance system. Provenance data are stored remotely or locally, depending on the selected storage type which will be discussed later in this chapter.

4.3.1.2 Input Based Project Management

An analysis is the experimentation of patient data. Hence, a project should have only one specific input, optionally data from a patient that need to be analyzed using different techniques and executing different tools. This process sometimes includes parameter

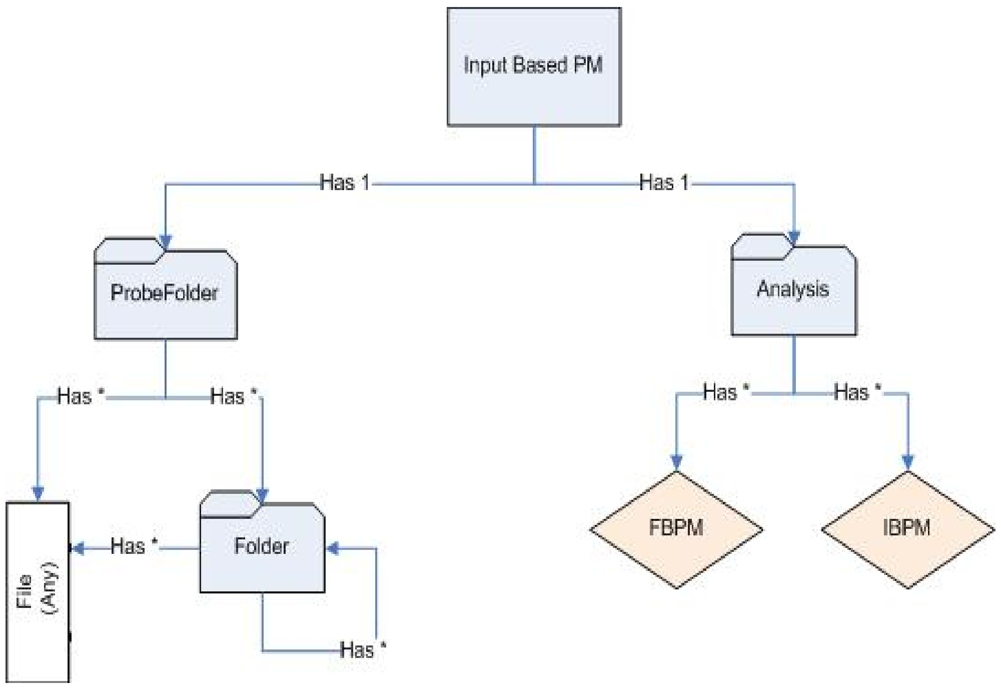


Figure 24 : Input Based Project Management Structure

optimization for specific functions which is in the boundaries of function based project management (FBPM). Therefore, input based project management is more generic than FBPM and actually it encloses it. So, FBPM projects can be embed in Input Based PMs.

Input BPM has only one input container. It can have as many files of any type as necessary. The second container in the structure is the analysis in which the experiments made on the specified input are stored.

Each experiment is itself an analysis, so they need a management structure. Input based PM allows embedding FBPM structures in it as it allows a recursive structure which means that IBPM can have IBPM within its structure as a part of the analysis. The only restriction here is that these management level's inputs can only consist of a certain sub-part of the top management level's inputs.

The most important point in IBPM is that once an analysis has been finished successfully, the user is able to compile the optimized workflow path to a new pipeline and to save it. This way, researchers wishing to find the most effective pipeline for a certain analysis can use IBPM for a certain input data set, apply very different tools on the set, optimize each part and finally they should be able to compile the best path to a workflow and use it for future analyses.

The definition of the constraints and the technique used for the insurance of project validity are discussed in point 4.4 Architecture.

4.3.2 Supported Storage Types

PPMS offers a data management system that should work online or offline.

From this point of view, PPMS offers local and remote storage mechanisms. Local storage mechanisms use the file system of the underlying operating system for the storage of the project files. Provenance data, however, are stored in a relational database.

Storing project data in the file system but not in a database of course has a reason. Researchers have so far been using file systems for storing their analysis data. So, they are indeed used to see the data in a file system by a file explorer in a structured order and they want to have it retainable. Another reason is that they need to use the data in another software when needed which has driven this type of architecture decision.

While the project data are stored in a file system, the provenance data captured are saved in a database running on a local machine.

Since the reasons why local storage providers preserve the project data in a file system are not valid in the case of a remote storage, the project data and provenance data are both entered into a database.

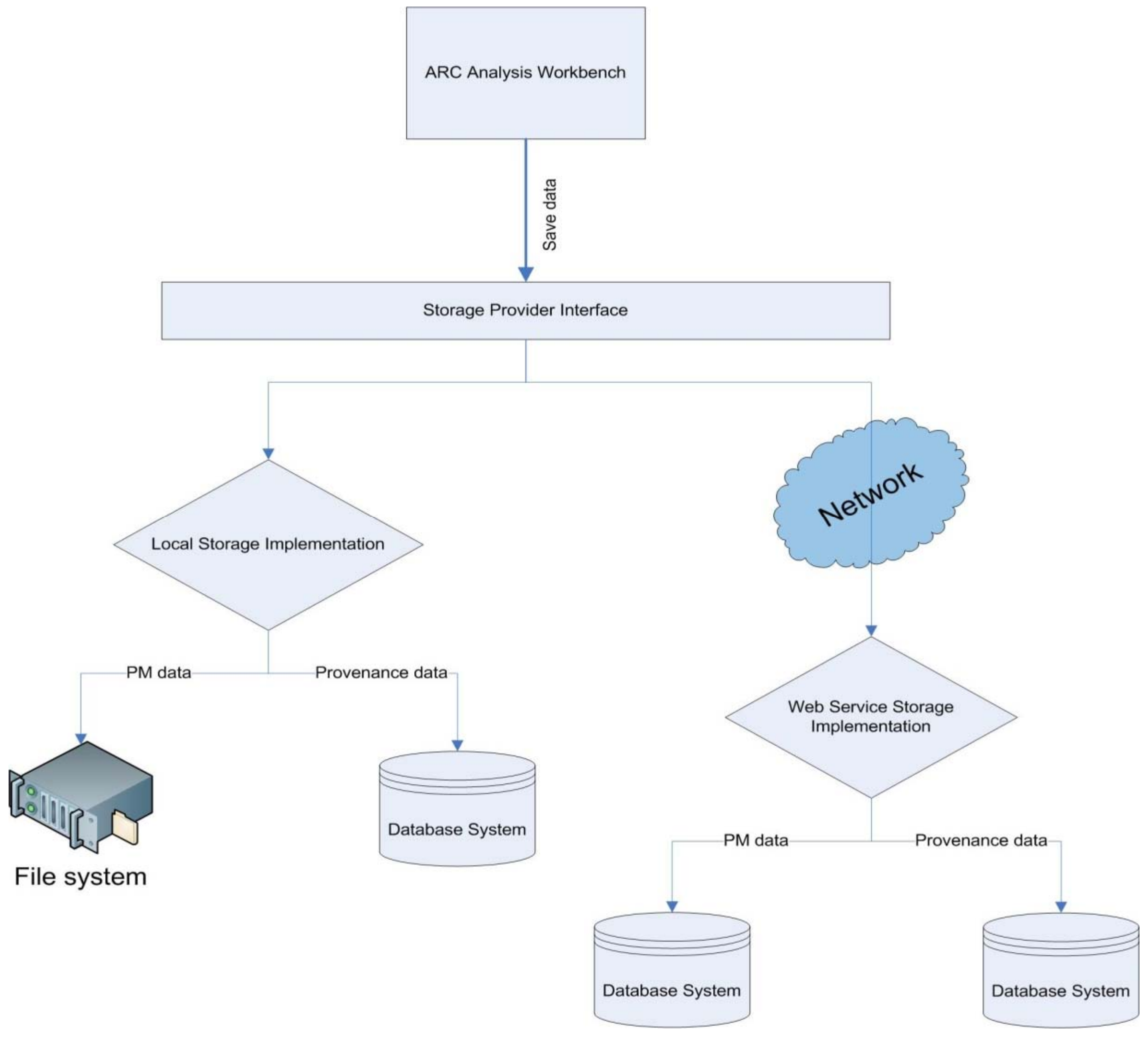


Figure 25 : Storage Type Support in PPMS

The technical details of the implementation are discussed in point 4.4 Architecture.

4.3.3 Provenance integration

Due to the fact that the ARCS Analysis Workbench is based on Taverna core version 1.6.2, the use of Logbook that is developed as a plug-in for Taverna was the perfect choice for the analysis platform. Hence, Logbook version 1.2.8 source has been hacked in order to integrate it with the project management system.

The major further implementation of the tool was the integration of the web service storage support and the browsing support of the remote data.

4.4 Architecture

The PPMS architecture is developed to be flexible and scalable enough for the easy addition of further project management styles. PPMS declares a project structure definition language (PSDL) which is used to define project styles in Xml. The PPM System ensures that the constraints are held by using a special layer. This layer and the language itself are explained in detail later on in this chapter.

PPMS is a client-server architecture. On the client side, it is implemented as a part of the ARCS Analysis Workbench which is a workflow based analysis software allowing the construction of workflows using different component technologies located on both local and remote machines. Moreover, the client library has the ability to manage the data on the current file system. While project management data are being stored in the file system, the provenance data captured are stored in a database running on a local machine.

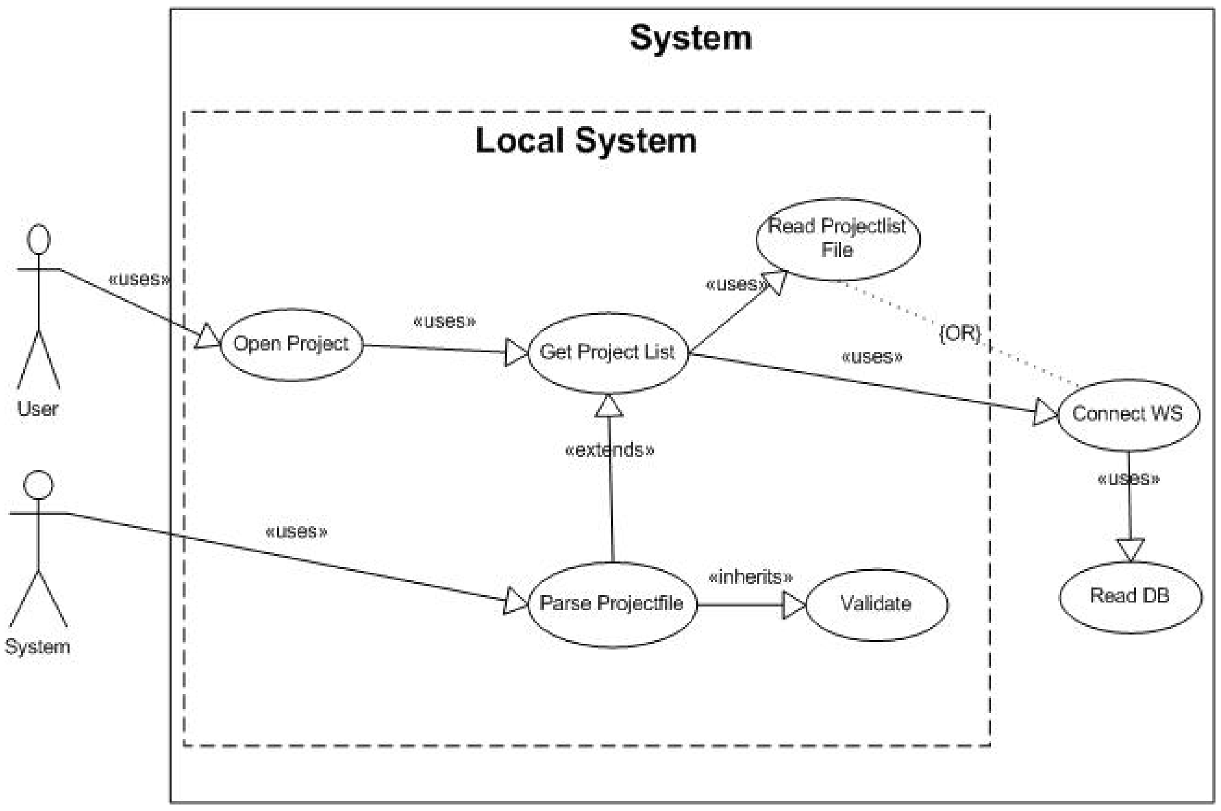
On the server side; PPMS offers two services: the Project Management Service, and the Provenance Service. The Project Management service has the functionality to manage the project data on the server side, execute CRUD functions on the project and more. The Provenance Service implements methods for storing provenance metadata and the actual provenance data.

After this short overview about PPMS architecture, I want to go into the technical details and show how it was developed. Before doing this, however, I will examine the most valuable use cases as well as functional requirements of the system.

4.4.1 Most Important Use Cases & Functional requirements

Four most important use cases are examined in detail here. Open Project, Create Project, Capture Provenance data and Run workflow. The other operations like saving, updating, removing project or provenance data are trivial and similar to these use cases, therefore they are not covered here.

4.4.1.1 Open Project

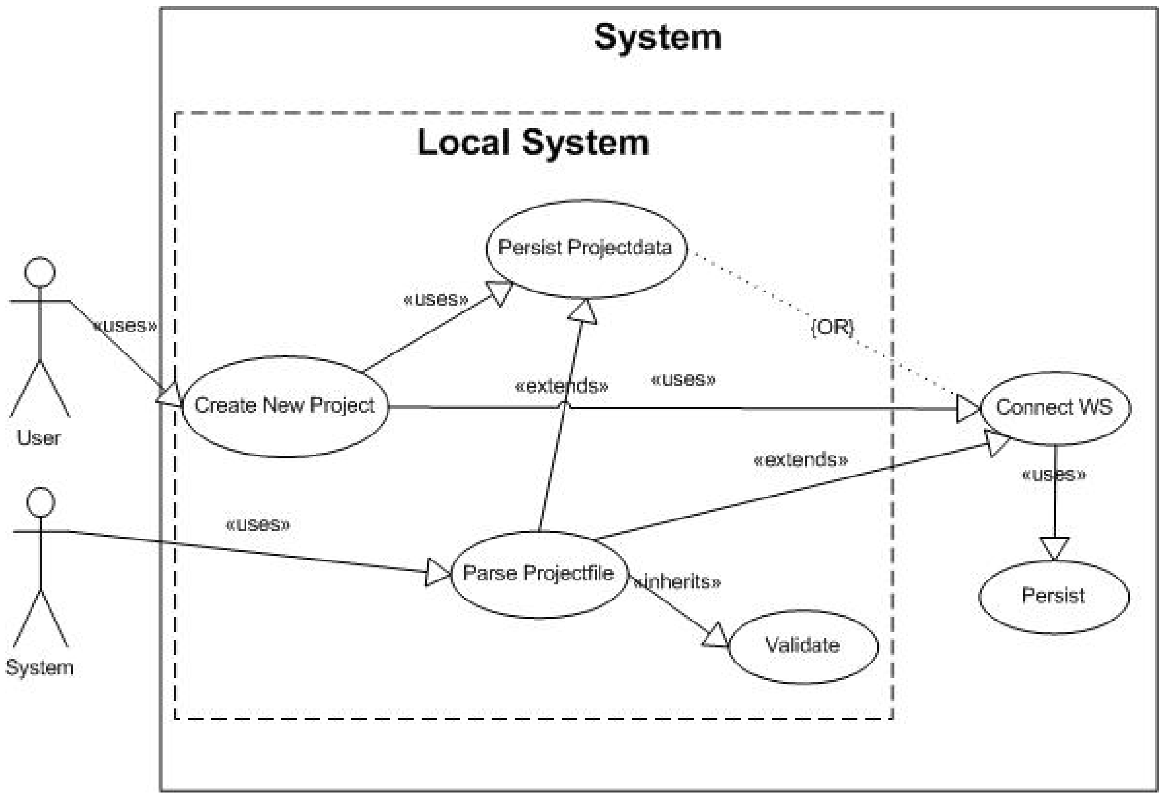


Use Case 1: Open Project

USE CASE 1	Open Project	
Goal in Context	User issues a request to open a project.	
Preconditions	The project is created earlier and in case of using a remote server the server is running. (see Use Case 2)	
Success End Condition	The project file is read and the project is opened in the client software.	
Failed End Condition	Error message displaying the reason is shown.	
Trigger	User request.	
DESCRIPTION	Step	Action
	1	User exhibited the will of opening a present project
	2	User selected the storage type to get the list of current project in selected layer
	3	System gets the list and shows it
	4	User selects a project from the list and pushes ‘open’ button

	5	System retrieves project Xml file
	6	System validates the project file against project definition file to hold the constraints.
	7	Project added to project manager and shown in user interface
EXTENSIONS	Step	Branching Action
	2a	Error connection remote server: 3a1. User will be warned not to use remote server and contact to the administrator.
	3a	Error while getting the list: 3a1. Reason is shown.
	5a	Error while getting the project file: 5a1. Reason is shown, data dropped.
	6a	Some nodes injure the project constraints: 4a1. The nodes which are not valid are ignored; others are parsed to a project tree.

4.4.1.2 Create Project

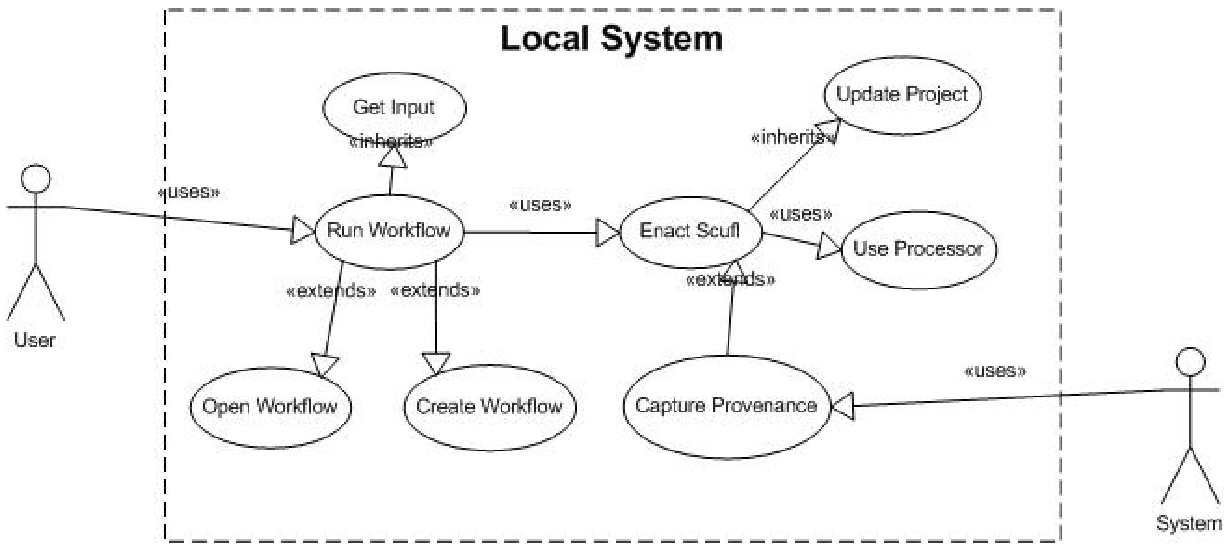


Use Case 2: Create Workflow

USE CASE 2	Create Project	
Goal in Context	User issues a request to create a new project.	
Preconditions	In case of remote storage, server is running.	
Success End Condition	The project file is saved and project is opened in the client software.	
Failed End Condition	Error message displaying the reason is shown.	
Trigger	User request.	
DESCRIPTION	Step	Action
	1	User exhibited the will of creating a new project
	2	User selected the storage type.
	3	System enters desired project name.
	4	System creates and stores default project file with default parameters.
	5	System retrieves project Xml file after creation.
	6	System validates the project file against project definition file to

		hold the constraints.
	7	Project added to project manager and shown in user interface
EXTENSIONS	Step	Branching Action
	4a	Error while creating or storing the project file: 3a1. Transaction rolled back and reason shown.
	5a	Error while getting the project file: 5a1. Reason is shown, data dropped.
	6a	Some nodes injure the project constraints: 6a1. The nodes which are not valid are ignored; others are parsed to a project tree.

4.4.1.3 Run Workflow

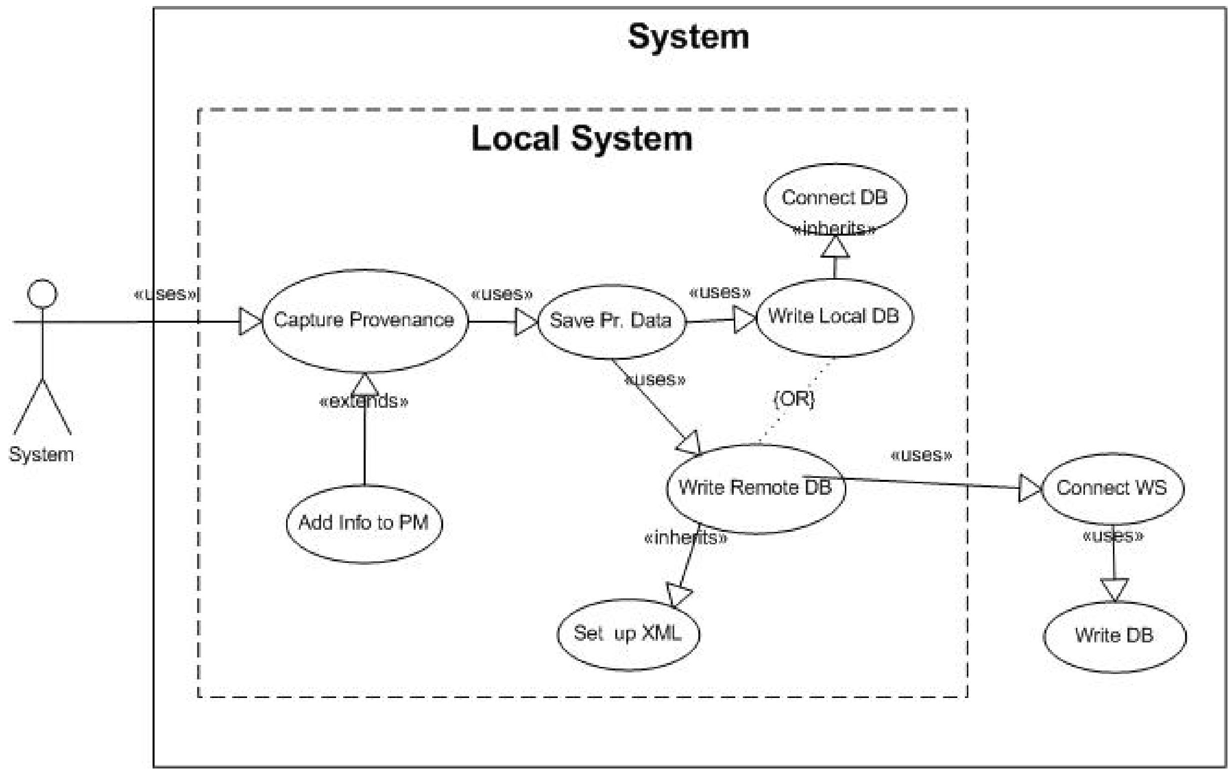


Use Case 3: Run Workflow

USE CASE 3	Run Workflow
Goal in Context	User issues a request to run a workflow.
Preconditions	Workflow is already opened, sub executables in workflow are installed in

	local machine and the remote services are alive.	
Success End Condition	Workflow run, desired results retrieved.	
Failed End Condition	Error message displaying the reason will be shown, transaction rolled back.	
Trigger	User request or a parent workflow's request in case of a nested workflow.	
DESCRIPTION	Step	Action
	1	User exhibited the will of running a workflow.
	2	Input dialog is shown for defining workflow inputs.
	3	Workflow runs with given input data and parameter.
	4	Workflow executed successfully untill the end.
	5	Execution results shown (in case of failure in any of the processes, failure is shown).
	7	Run node with information about workflow run identification is added to project management tree.
	8	Provenance data are captured and sent to storage device via desired storage type connection.
EXTENSIONS	Step	Branching Action
	2a	Workflow needs no input: 2a1. Skip to Step 3.
	4a	Error while running the workflow: 4a1. Even so, go on with Step 5.
	8a	Error while transferring the provenance data: 8a1. Error reason shown.

4.4.1.4 Capture Provenance Data



Use Case 4: Capture Provenance Data

USE CASE 4	Capture Provenance Data	
Goal in Context	System should capture the execution data	
Preconditions	Workflow is running, local db or remote service is alive	
Success End Condition	System stored the provenance information about a certain workflow run	
Failed End Condition	Workflow run information is not stored.	
Trigger	Starting of a workflow execution	
DESCRIPTION	Step	Action
	1	A workflow is executed
	2	System captures the execution information and sends them for storing via storage connection interface to desired persistence context
	3	Execution complete
	4	A Run node with identification information about just run

		workflow is added to the project tree
EXTENSIONS	Step	Branching Action
	2a	Storage type is web service: 2a1. Set up Xml file with project data for transferring over network via web services.
	2b	Cannot send data or cannot persist data in persistence context: 2b1. Show error message and possible reason
	3a	Execution failed: 3a1. Show result pane, go on with step 4.

4.4.2 Design architecture

The architectural design is explained in detail in three parts. First I will show the language used for defining constraints for function based project management. Then the server architecture is introduced including the Relaxer [36] tool which is one of the key tools for PPMS project used to generate project validation classes and Xml bean objects for manipulating the project file.

4.4.2.1 Project Structure Definition

The project types in PPMS are defined using a specific Xml scheme language called Relax NG. There are some other scheme languages like DTD, XSD and Relax Core which are candidates for the use in constructing project structure definitions and each of them has advantages over the others, but the language used for project structure definition for PPMS is Relax NG.

The reason of choosing Relax NG as scheme language is due to its user-friendly and powerful but yet simple form.

As introduced in [37]:

RELAX NG is a very powerful, yet easy to understand scheme technology and is a normalized grammar based on James Clark’s Tree Regular Expression for XML(TREX), and Makoto Murata’s Regular Language description for XML (RELAX).

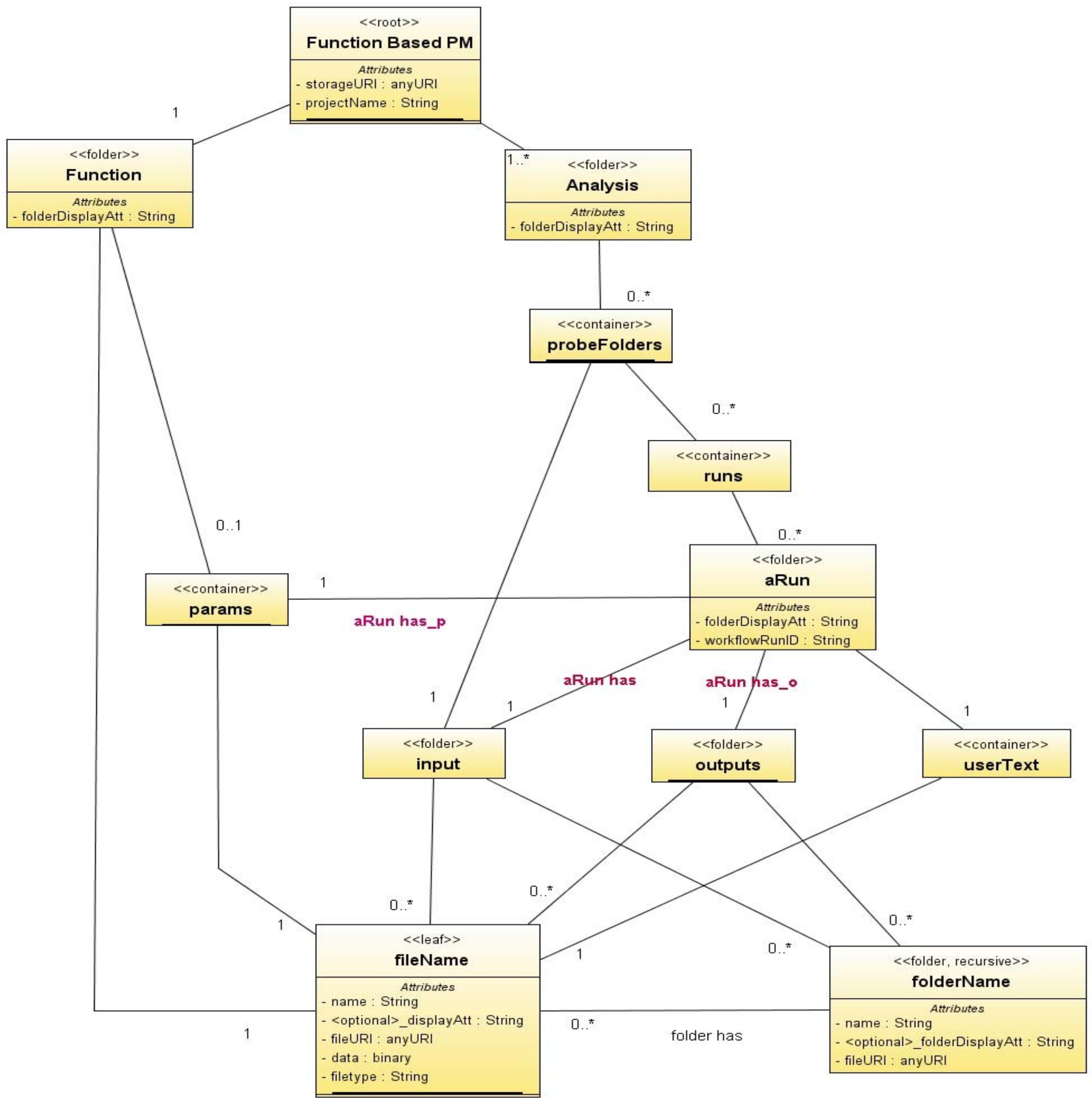


Figure 25 : Function Based Project Management Structure in UML model (red colored nodes are covered in the provenance range)

Its key features are:

(i) *It is simple and easy to learn.*

(ii) *It uses pattern-based grammar with a strong mathematical foundation.*

(iii) *It has two different syntaxes: XML syntax and compact syntax.*

(iv) *It supports XML Scheme data types.*

(v) *It supports user-defined data types.*

(vi) *It supports XML namespaces.*
(vii) *It is highly composable.*
(viii) *Elements and attributes are treated in the same way.*

I will introduce the function based project management definition file since current implementation of PPMS client software at current only support FBPM project types.

Function based projects have two top containers: Function and Analysis. Figure 25 represents a UML model of the projects node's structure where the constraints are clearly identified. This is represented in a definition file with the code:

```
<start>
     <!—project type name -->
    <element name="FBPM">
      <attribute name="projectName">
        <data type="string">
          <param name="minLength">1</param>
          <param name="maxLength">25</param>
        </data>
      </attribute>
      <attribute name="storageURI">
        <data type="anyURI" />
      </attribute>
      <!—group element declares that the sub elements should appear exactly in the order like here -->
      <group>
         <!—reference to function -->
        <ref name="mfunction" />
         <!—reference to anlaysis node -->
        <ref name="mprobe" />
      </group>
    </element>
 </start>
```

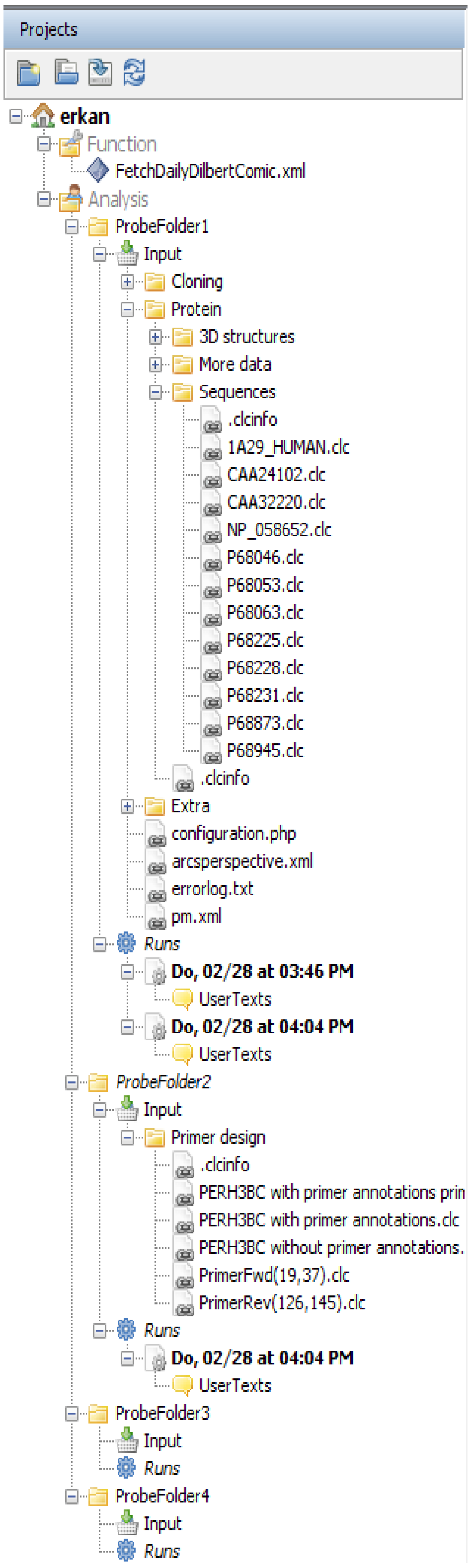


Figure 26 : FBPM Project tree

This code says exactly that the project named „FBPM" has two attributes and two sub elements. Attribute "projectname" is a string with a minimum length of 1 and a maximum character length of 25. "StorageURI" is a predefined XML Scheme data type [38] which is representing any type of URL.

The two sub elements here are references to other element definitions: *"mfunction"* and *"mprobe"*. Here are the exact definitions of these elements:

```
<define name="mfunction">
    <element name="function">
…………
        <!—function container may have one scufl definition file
representing the workflow -->
        <optional>
            <ref name="mfile" />
        </optional>
        <optional>
            <element name="params">
                <ref name="uri" />
                <ref name="mfile" />
            </element>
        </optional>
    </element>
  </define>
```

Function folders may have a file representing the workflow. The enact-able workflows are the ones in the XScufl definition language. A function may have an element for storing of parameters as well.

```
<define name="mprobe">
    <element name="probes">
```

```
……………….
            <zeroOrMore>
                <!--
                      An Analysis can have zero or unlimited number of input
                      (probe) folders. Each of them are representing a patient
                      data or a set of files that belong to each other and
                      must use in an analysis
                -->
                <element name="probeFolders">
                    …………….
                    <!-- The actual input set data comes within this folder -->
                    <element name="input">
                        ……………………
                        <interleave>
                            <zeroOrMore>
                                <ref name="mfolder" />
                            </zeroOrMore>
                            <zeroOrMore>
                                <ref name="mfile" />
                            </zeroOrMore>
                        </interleave>
                    </element>
                    <!--
                          The runs which are executed using inputs of this
                          container's inputset, stored in inside runs container
                    -->
                        <element name="runs">
                            ……………………….
                            <zeroOrMore>
                                <!-- represents one unique run -->
                                <ref name="mruns" />
                            </zeroOrMore>
                        ……………..
    </define>
```

Element *"mprobe"* represents the analysis node. An Analysis can have zero or unlimited number of input (probe) folders. Each of them is representing a patient data or a set of files that belong to each other and need to be used in an analysis together. This set of data can consist of zero or more file or folders which can again have zero or more files and folders recursively.

Probe folder's input set saved within the node named *"input".* Probe folder node has a different node with name *"runs"* which keeps track of the executions of the project function. The runs which are executed using the input set of Probe-Folder X will be stored under the *runs* container of the same probe folder node.

One last important element file element which has the following definition:

```
<define name="mfile">
    <element name="file">
      ……………………………..
      <!-- File extension - mandatory -->
      <attribute name="fileExt">
        <data type="string">
          <param name="minLength">1</param>
          <param name="maxLength">10</param>
        </data>
      </attribute>
       ……………..
      <!-- file name - mandatory -->
      <attribute name="name">
        <data type="string">
          <param name="minLength">1</param>
          <param name="maxLength">25</param>
        </data>
      </attribute>
      <!--
        file inhalt - optional
        optional because sometimes will be empty because of
        memory optimization. URL pointing where is the file actually
        stored is in such cases enough.
      -->
```

```
            <optional>
                <element name="data">
                    <data type="base64Binary" />
                </element>
            </optional>
        </element>
    </define>
```

File node has two mandatory attributes (among other trivial ones) and one optional sub element. File extension and file name attributes are mandatory. The mime type of the file will be identified according to the file extension.

The data element is optional since binary data of one file are usually very heavy and it is almost impossible to load one project file with more than for instance ten .tiff images to the memory at one time. Therefore, when working with projects, the data are not loaded to the memory but only then when they are needed they are read from the storage URL. The file element has a mandatory attribute "*storageURI*" which points to the place where the data are actually stored.

The storage mechanisms differ between the two storage types. That is totally natural as the local storage implementation stores project data on the file system, while the remote storage uses as persistence context of a database.

How a URL pointing to a file somewhere on a file system looks should be absolutely natural even if it differs between operating systems running on a particular machine. What does the URL look like in the case of a remote storage?

```
At the highest level a URI reference in string form has the syntax .
                    [scheme:]scheme-specific-part[#fragment]
where square brackets [...] delineate optional components and the characters.[39]
```

The server side of PPMS manipulates the URL structure and uses it for pointing table rows. A URL is used for addressing to a certain row in the database table as follows:

```
URI =                   [scheme:]scheme-specific-part[#fragment]
PPMS Server form =      dbname:table-name#id
```

```
Data type =            String:String#Long
```

A complete version of FBPM project structure file can be found on page 99, Appendix A. 3.

4.4.2.2 Server architecture

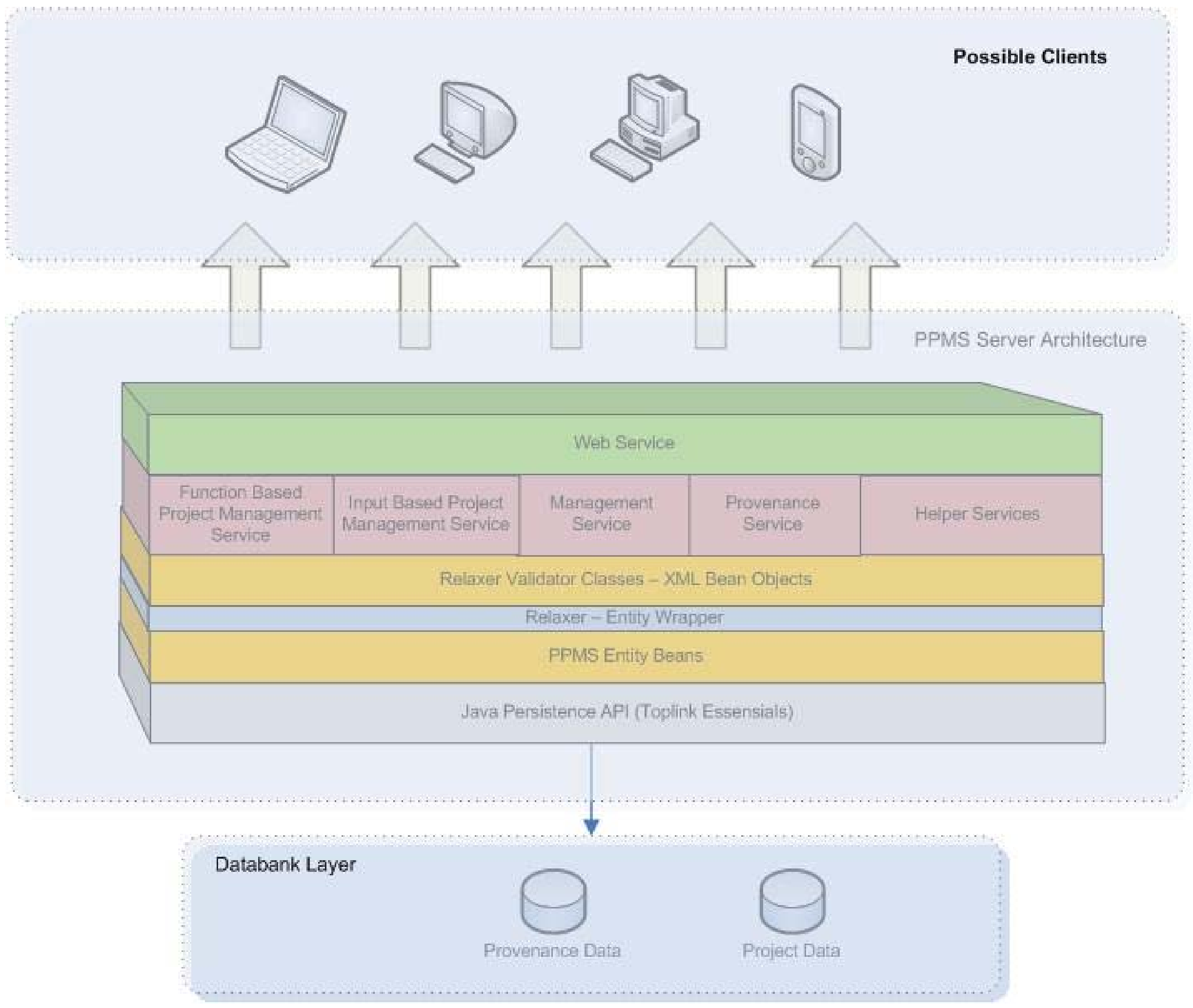


Figure 27 : PPMS Server Architecture

At the bottom of the architecture are the PPMS entity beans which are being persisted in the underlying persistence context (see Figure 27)

The entity class structure of projects is the same one as an Xml structure. Thereby, in the case of Function Based Project Management, the class structure and the cardinals are the same as presented in Figure 25.

The Glassfish application server implements the Java EE 1.5 specification which indicates how EJB 3.0 entities have to be persisted in the underlying context using JPA API. As the API implementation, Oracle's TopLink Essentials package finds usage in PPMS which is responsible for persisting project entities in the database.

Entities are generated from project Xml files delivered from the web service layer. At this point, Relaxer generated Xml classes are responsible for validating the project against defined constraints in the project structure file.

For each element the Relaxer generates an Xml scheme file, written in Relax NG, classes which manage creating, editing, and removing elements while keeping defined constraints. Among a lot of other options that the Relaxer tool has, the following options are used to generate classes for FBPM:

```
relaxer -java.verify -java.pattern.visitor -java.pattern.visitor:light -classPrefixes:FBPM_ -dir:../fbpmStubs -
java.package:arcs.resources.fbpmStubs FunctionBasedPMDef.rng
```

This indicates that the classes should have the functionality to verify the Xml file. Visitor pattern [40] should be implemented so that a visitor class can traverse over the Xml structure. Generated class names have to have "FBPM_" as prefix.

-java.verify options add two methods to each element-class:

```
/**
    * Verify an object against the schema.
    * @return boolean
    */
   public boolean isValid() {
       return (verify().isValid());
   }

/**
    * Reports a validity of an object against the schema.
    * @return RVerifyReport
    */
   public RVerifyReport verify() {
       RVerifyReport report = new RVerifyReport();
```

```
        RVerifyContext context = new RVerifyContext();
        verify(report, -1, context);
        return (report);
    }
```

Using these methods ensures that the project file is kept uninjured. Besides that, while setting up a new class from a present project file, setup methods ignores ignored parts and this way it does not allow using an invalid project file.

There is an interface to wrap the element classes to entity beans so that they are persisted in the database by the application server. The method of the FBPMtoEntityWrapper doing this is:

```
public ElementInterface wrap(Object node) {
....
}
```

where the ElementInterface is an interface implemented by all the entity classes.

The Project management style-types use the Relaxer-generated classes to manage the project objects. Project managers are stateless EJB3.0 beans, which implement remote interfaces in order to be accessed by web services.

There are four types of session beans in the PPMS Server: Project Management Service, which has for now only a method telling about which type of projects this server supports. The project types are transferred using a domain object bean called the *ProjectTypeDO*. This domain object has tree features: description of the project type, name of the project type, and a muster tree showing how the project tree will be displayed in the client’s window. This gives a better sense of the project structure:

```
public class ProjectTypeDO implements Serializable{

    private String description;
    private String name;
    private byte[] musterTree;
```

```
//setters, getters
```

The second session bean is the project manager controller. For now, only the function based project manager controller is implemented which is responsible for managing FBPM projects. The *FunctionBasedPMControllerBean* has seven features for managing or editing a project. It looks like this:

```
@javax.ejb.Remote
public interface FunctionBasedPMControllerRemote {

    /**
     * Gets an already present project from database with given id
     */
    ProjectXMLDO getFBPM(Long id) throws ResourceNotFoundException;

    /**
     * Saves given project with its elements
     * return true if it was successfully saved
     * false if not
     */
    boolean saveProject(ProjectXMLDO projectxml);

    /**
     * Returns a list of projects currently present in the database
     * as a list of ProjectXMLDO domain object
     */
    public ProjectXMLDO[] getProjectList();

    /**
     * Removes element addressed by given uri
     */
    public boolean removeElement(java.net.URI uri) throws NoSuchElementException;

    /**
     * Creates and returns a new project with given name
     * sets it up with mandatory root elements
```

```
  */
  public ProjectXMLDO createNewFBPMProject(java.lang.String name);

  /**
   * Updates elements in the database with given element.
   * uri addresses element to be updated
   */
  public boolean updateElement(java.net.URI storageURI, String element);

  /**
   * Adds a new element to database. ClassName represents a Relaxer element
   * class which is initialized over reflection and set up by passing
   * elementInXML parameter to setup method.
   */
  public URI addElement(java.lang.String className, java.lang.String elementInXML) ;
}
```

What do the methods do is explicitly explained according comments so no need to go into more detail.

Third type of sessions are the helper sessions which are *DAServiceBeans* used for getting a certain row using a URL and *FBPMXMLPersister* which is responsible for traversing over the Xml elements to persist them in a database. The special URL used for addressing rows is mentioned in chapter 4.4.2.1 Project Structure Definition.

The Provenance Service has the functionality to store meta-data information and actual data that are captured from workflow executions. The LogBook has two interfaces for storing provenance information: *MetadataService and DataService.* The functionality that these interfaces require is storing and retrieving named-graphs which are used for meta-data storing intensions and *ScuflModel* objects which are the FreeFluo objects created from the XScufl definitions.

Some of the functionality of these services is done at the PPMS client and information that is sent over the network to the PPMS server is parsed to XML representations using Taverna core functions so that they are transferable without losing any information.

Among others, the Provenance Service offers the following methods:

```
/**
 * Retrieve workflow associated with workflow id
 */
public java.lang.String getWorkflow(java.lang.String workflowLSID);

/**
 * Stores rdfGraph under graphName
 */
public void storeRDFGraph(java.lang.String rdfGraph, java.lang.String graphName);

/**
 * Removes the graph corresponding to workflowRun from the
 * repository and then does the same by recursion for all its possible
 * nested workflow runs and process runs.
 *
 */
public void removeWorkflowRun(java.lang.String workflowRun);

/**
 * Retrieves the graph corresponding to workflowRunId
 * together with all the sub graphs of processes and nested runs and returns
 * it as a String containing its RDF/XML representation.
 *
 */
public java.lang.String getWorkflowRun(java.lang.String workflowRunId);

/**
 * Gets the LSIDs of all the workflow runs in store.
 *
 */
public java.lang.String[] getAllWorkflowRuns();

/**
 * Gets all the (ids of) workflow inputs for <code>workflowRunLSID</code>
 *
 */
public java.lang.String[] getWorkflowInputs(java.lang.String lsid);

/**
 * Gets all the (ids of) workflow outputs for <code>workflowRunLSID</code>
 */
public java.lang.String[] getWorkflowOutputs(java.lang.String workflowRunLSID);

/**
 * Stores dataThing, representing a data in analysis workbench.
 */
public void storeDataThing(String dataThing, boolean silent);

/**
 * Store a workflow that is being run. LSIDs are unique to each
 * run.
 */
public void storeWorkflow(java.lang.String lsid, String model);

/**
 * Fetch a DataThing from the given LSID
 */
public String fetchDataThing(java.lang.String lsid);

/**
 * Stores under graphName the RDF graph corresponding to
 * model.
 */
public void storeModel(String model, java.lang.String graphName);
```

4.4.2.3 Client architecture

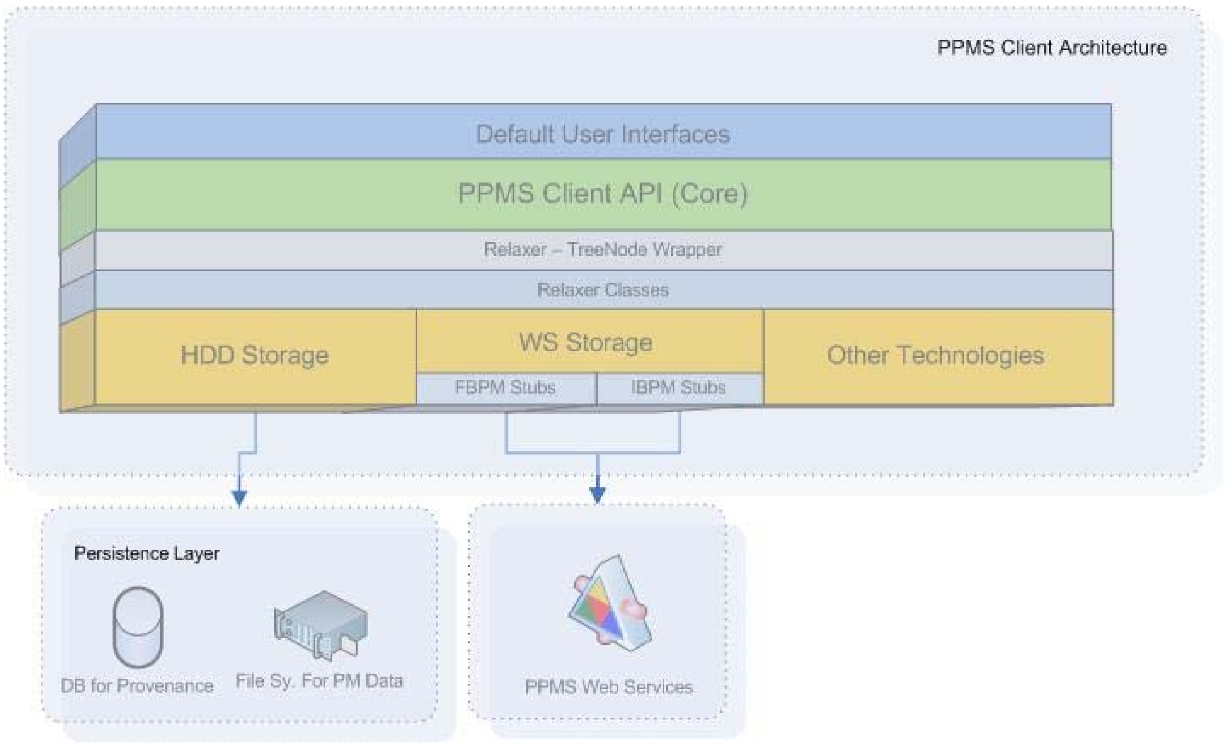


Figure 28 : PPMS Client Architecture

The client implements classes for the interaction with users in an application. The PPMS client is a built-in module for the ARCS Analysis Workbench.

The client core is developed in a flexible way so that any other applications can implement interfaces and use its own GUI components for user interaction. However, PPMS offers default user interfaces as well.

As mentioned in 4.3.2 Supported Storage Types and as can be seen in Figure 25 : Storage Type Support in PPMS, the system has two different storage type supports. PPMS defines and uses an interface called *StorageConnectionProvider* for storing data. That means any other communication protocols can easily be supported by implementing this interface.

Storage types should implement the following methods:

```
/**
 * Returns the project types which are supported by underlying
 * storage mechanism.
 */
 public ProjectTypeDO[] getSupportedProjectTypes() throws ResourceNotFoundException;

/**
 * Returns project with given id
 */
ProjectXMLDO getProject(Long id) throws ResourceNotFoundException;

/**
 * Saves project
 */
boolean saveProject(ProjectXMLDO projectxml);

/**
 * Returns project that are currently exist in underlying context
 */
public ProjectXMLDO[] getProjectList();

/**
 * Removes element which is specified with uri
 */
public boolean removeElement(java.net.URI uri) throws NoSuchElementException;

/**
 * Creates new project with given name and returns it
 */
public ProjectXMLDO createNewProject(ProjectTypeDO type, java.lang.String name);

/**
 * Changes element at storageURI with element "element"
 */
public boolean updateElement(java.net.URI storageURI, String element);

/**
 * Finds stub-class with given className and stores object in
 * underlying storage device
 */
public URI addElement(java.lang.String className, java.lang.String elementInXML) ;

/**
 * remove project with given id
 */
public boolean removeProject(Long id);
```

Comments in code explain functionality, as always done in this document.

There are two types of storage connection provider implementations in PPMS. Using Web Services and using File System of the current operating system.

The *HDDStorageProvider* implements file system storage support. The data are stored in the file system using default java.io classes. The folder structure is the same structure as the tree structure showing the project (Figure 26 : FBPM Project tree).

In the case of local management of the data, provenance data are stored in the database by the LogBook tool.

WSStorageProvider class is responsible for storing data on a remote server. For doing this, the WSStorageProvider uses the stub classes generated by jaxws's wsimport tool for using the web services offered on the ARCS's application servers (see Figure 28).

Provenance data will be captured by LogBook's own capturing ability and sent to the server by using the web service connection interface. The methods implemented for this are described in chapter 4.4.2.2 .

The storage provider interface requires that the project are transferred using the *ProjectXMLDO* domain object. This class is a serializable POJO which has:

```
private Long id;
private String xml;
private ProjectTypeDO projectType;
```

The string Xml attribute contains an Xml representation of the project. The Id identifies the project in the context of storage mechanisms and the ProjectType is the type of the project.

The client core sets a new Relaxer project object using the Xml attribute of the ProjectXMLDO class. As mentioned earlier, during the setup the Relaxer checks the compatibility of the project

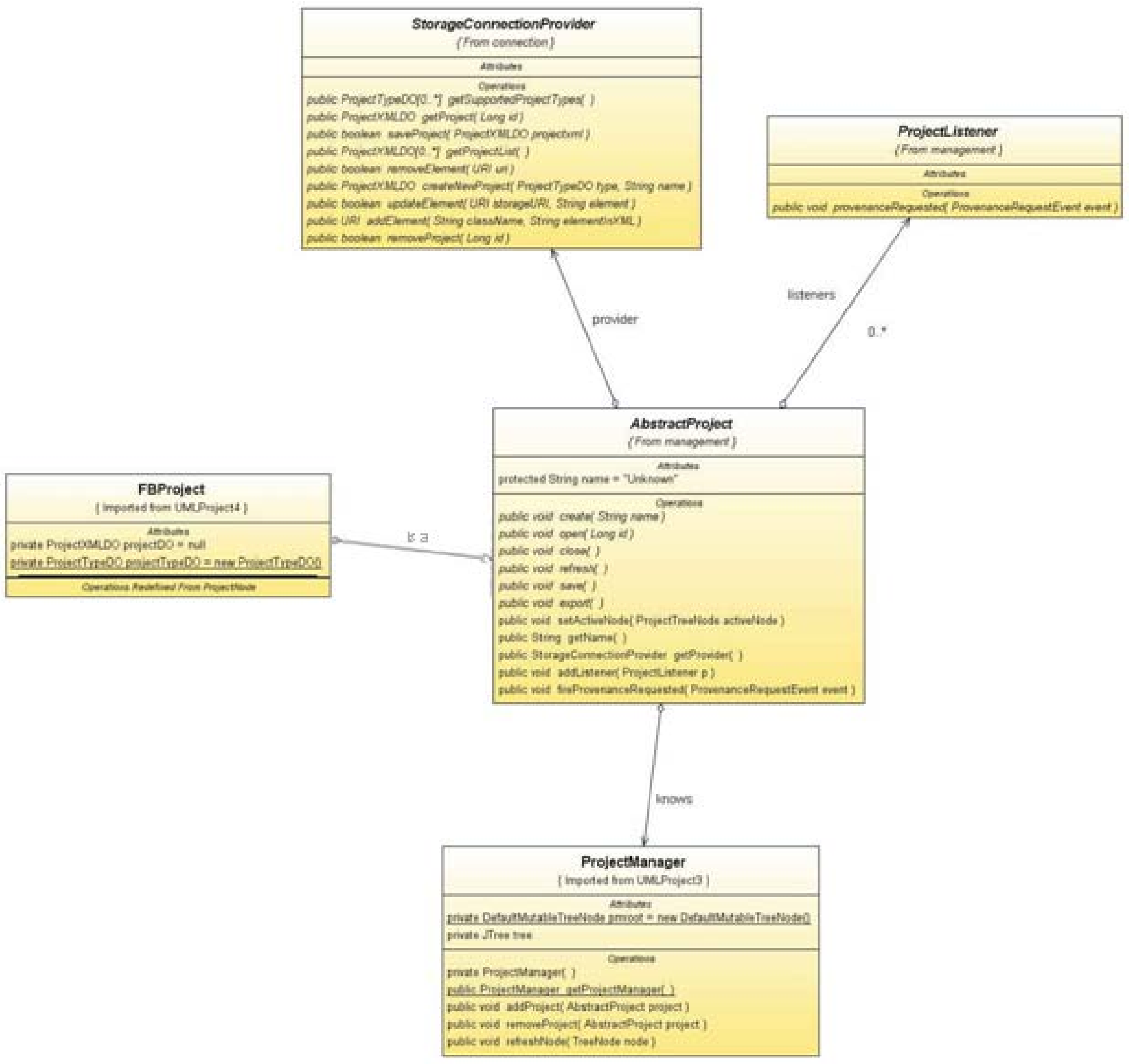

StorageConnectionProvider
(From connection)
Attributes
Operations
public ProjectTypeDO[0..*] getSupportedProjectTypes( )
public ProjectXMLDO getProject(Long id)
public boolean saveProject(ProjectXMLDO projectxml)
public ProjectXMLDO[0..*] getProjectList( )
public boolean removeElement(URI uri)
public ProjectXMLDO createNewProject(ProjectTypeDO type, String name)
public boolean updateElement(URI storageURI, String element)
public URI addElement(String className, String elementInXML)
public boolean removeProject(Long id)
ProjectListener
(From management)
Attributes
Operations
public void provenanceRequested(ProvenanceRequestEvent event)
provider
listeners
0..*
AbstractProject
(From management)
Attributes
protected String name = "Unknown"
Operations
public void create(String name)
public void open(Long id)
public void close( )
public void refresh( )
public void save( )
public void export( )
public void setActiveNode(ProjectTreeNode activeNode)
public String getName( )
public StorageConnectionProvider getProvider( )
public void addListener(ProjectListener p)
public void fireProvenanceRequested(ProvenanceRequestEvent event)
FBProject
(Imported from UMLProject4)
Attributes
private ProjectXMLDO projectDO = null
private ProjectTypeDO projectTypeDO = new ProjectTypeDO()
Operations Redefined From ProjectNode
is a
knows
ProjectManager
(Imported from UMLProject3)
Attributes
private DefaultMutableTreeNode prmroot = new DefaultMutableTreeNode()
private JTree tree
Operations
private ProjectManager( )
public ProjectManager getProjectManager( )
public void addProject(AbstractProject project)
public void removeProject(AbstractProject project)
public void refreshNode(TreeNode node)

Figure 29: Project Manager Interaction graph

against the project structure scheme file and it ignores the elements that are injured.

The *XMLtoTreeParser* takes care of parsing the Relaxer object and of generating a tree for displaying it in the project manager window.

ProjectManager class manages opened projects of the type *AbstractProject*. Figure 29: Project Manager Interaction graph shows the interaction of the project manager class. Each project type should implement the *AbstractProject* in order to be used in the client. For function based projects, PPMS has a default implementation, the *FBProject*.

Moreover, the project manager lets projects define their actions. All the nodes which are displayed in manager tree are types of the *ProjectTreeNode. ProjectTreeNode* interface requires that the method

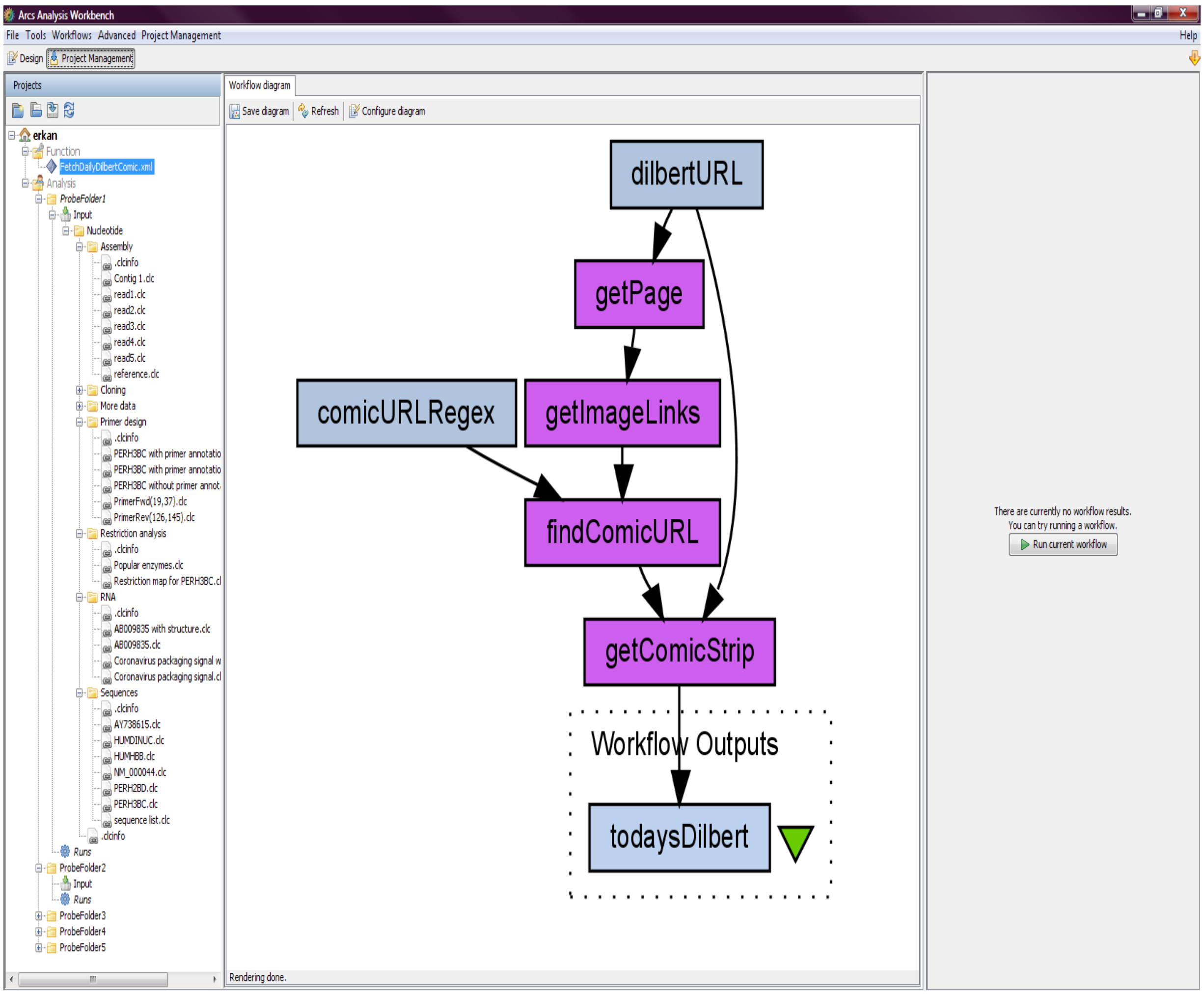
```
JPopupMenu getPopup();
```

is implemented. The Project manager uses this method to show a popup menu when the user right-clicks over the corresponding tree node. This way, project implementations are free to define actions that they want.

Finally, the client uses the LogBook Browser tool for displaying provenance information. Each run node in the function based project management has an action for displaying the appropriate provenance information in detail using the LogBook's browser.

4.4.3 Usage

First of all, the ARCS workbench with project management support has to be installed. There are no other tools necessary, assuming that java 1.6 is installed on the machine on which analysis workbench is running.

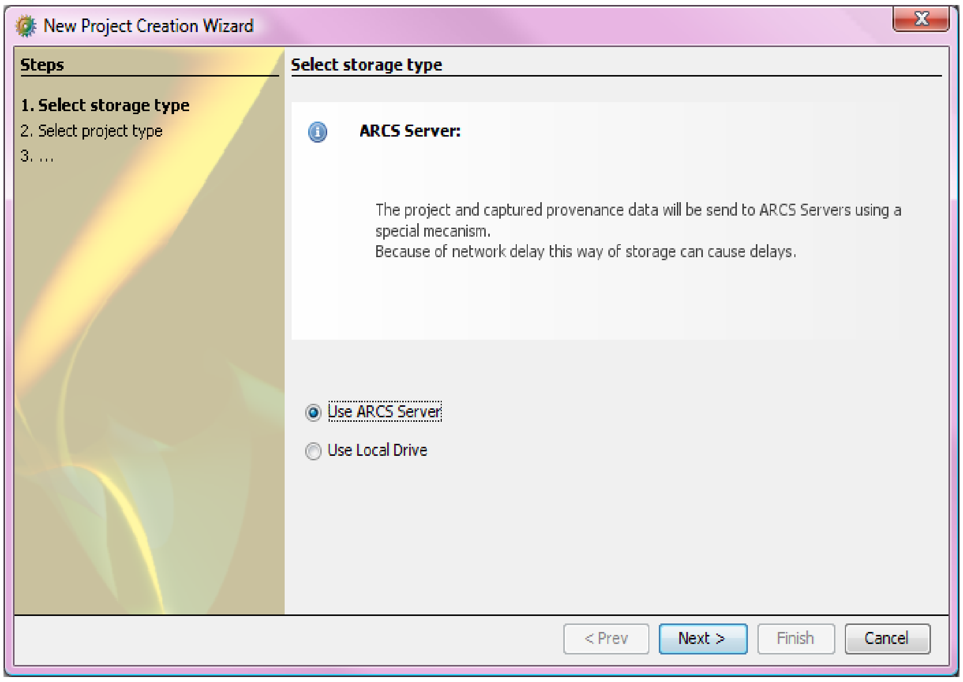


Rendering done.

There are currently no workflow results.
You can try running a workflow.

Figure 30 : ARCS Analysis Workbench with Project Management Support

After starting the workbench, there are two perspectives: design and project management. Select the project management perspective to work with the project management system. By default, there are no projects opened. One can open or create projects by selecting the open or create project actions from the file menu or by clicking on the appropriate button in the projects toolbar (see Figure 30).



The project and captured provenance data will be send to ARCS Servers using a special mecanism.
Because of network delay this way of storage can cause delays.

(● Use ARCS Server) ○ Use Local Drive

Figure 31 : New Project Wizard

Performing one of these actions opens a wizard to create or open a project. First, the user should select the storage type. There are two options: using the ARCS Remote Server or a local drive (see Figure 31).

In the case of the new project wizard, the user in the second step enters the type of project he wishes to create. Options here are “Function Based Project Management” and “Input Based Project Management” of which the second one is for now disabled since the PPMS does not support IBPM yet. The third step then is to name of the project. The user should select a workflow for the project since function based projects must own a function.

In the case of the open project wizard, the user is shown a list of present projects saved earlier. By selecting one of the projects from the list, the wizard shows the meta-information about the project (see Figure 33).

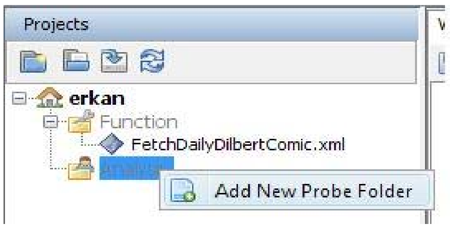
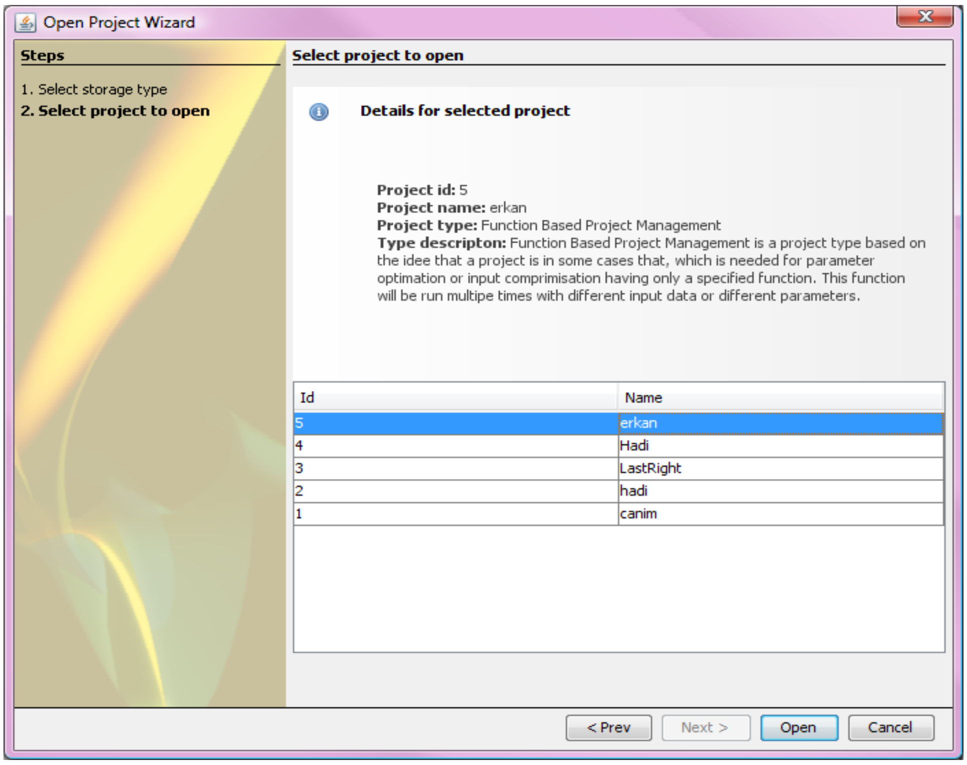


Figure 32 : Right click to add new probe folder node

Finishing the wizard adds the project to the project tree standing on the left of the window. If the project has no probe folder node, right-clicking on the analysis node shows a popup menu with an action to add a new probe



Open Project Wizard

Steps

1. Select storage type
2. **Select project to open**

Select project to open

Details for selected project

Project id: 5
Project name: erkan
Project type: Function Based Project Management
Type descripton: Function Based Project Management is a project type based on the idee that a project is in some cases that, which is needed for parameter optimation or input comprimisation having only a specified function. This function will be run multipe times with different input data or different parameters.

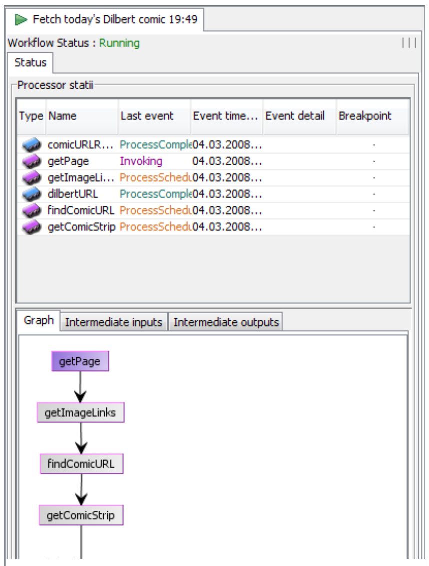
Id	Name
5	erkan
4	Hadi
3	LastRight
2	hadi
1	canim

Figure 33 : Open Project Wizard shows meta-data about the selected project

folder (see Figure 32).

After this, the user can import data to probe the folder by clicking the import data action from the desired probe folder's popup menu. All the imported leaf nodes (files) are dragable and carry contents of the corresponding files.

Before starting to run though, the user should open the workflow by clicking 'open action' from the function's popup menu. This opens the workflow and displays it in the workflow diagram panel in the middle of the workbenches main window. Clicking the "run current workflow" button standing at the right part of the window makes FreeFluo enactor run the workflow.



Fetch today's Dilbert comic 19:49

Workflow Status : Running

Status

Processor statii

Type	Name	Last event	Event time...	Event detail	Breakpoint
	comicURLR...	ProcessComple	04.03.2008...		·
	getPage	Invoking	04.03.2008...		·
	getImageLi...	ProcessSchedu	04.03.2008...		·
	dilbertURL	ProcessComple	04.03.2008...		·
	findComicURL	ProcessSchedu	04.03.2008...		·
	getComicStrip	ProcessSchedu	04.03.2008...		·

Figure 34: Running pane

At this moment, the running pane is replaced with a button panel. The running pane shows the list of processes scheduled (see Figure 34) to run and each one's current status. The user can follow the proceedings from this process list.

After finishing of the execution results, the pane shows the workflow results as well as intermediate inputs and outputs.

The client's provenance tool captures all the information and sends it to the storage mechanism using the projects desired storage provider implementation. After sending finishes, a run node will be added to the project folder *Runs* node showing the execution time. The users can right-click on it to display the popup menu which has an action for browsing the provenance information (see Figure 35).

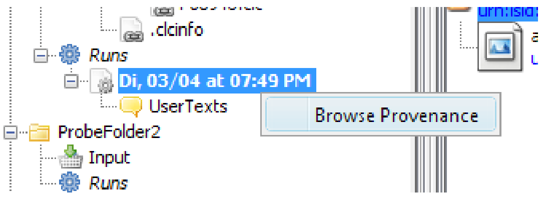
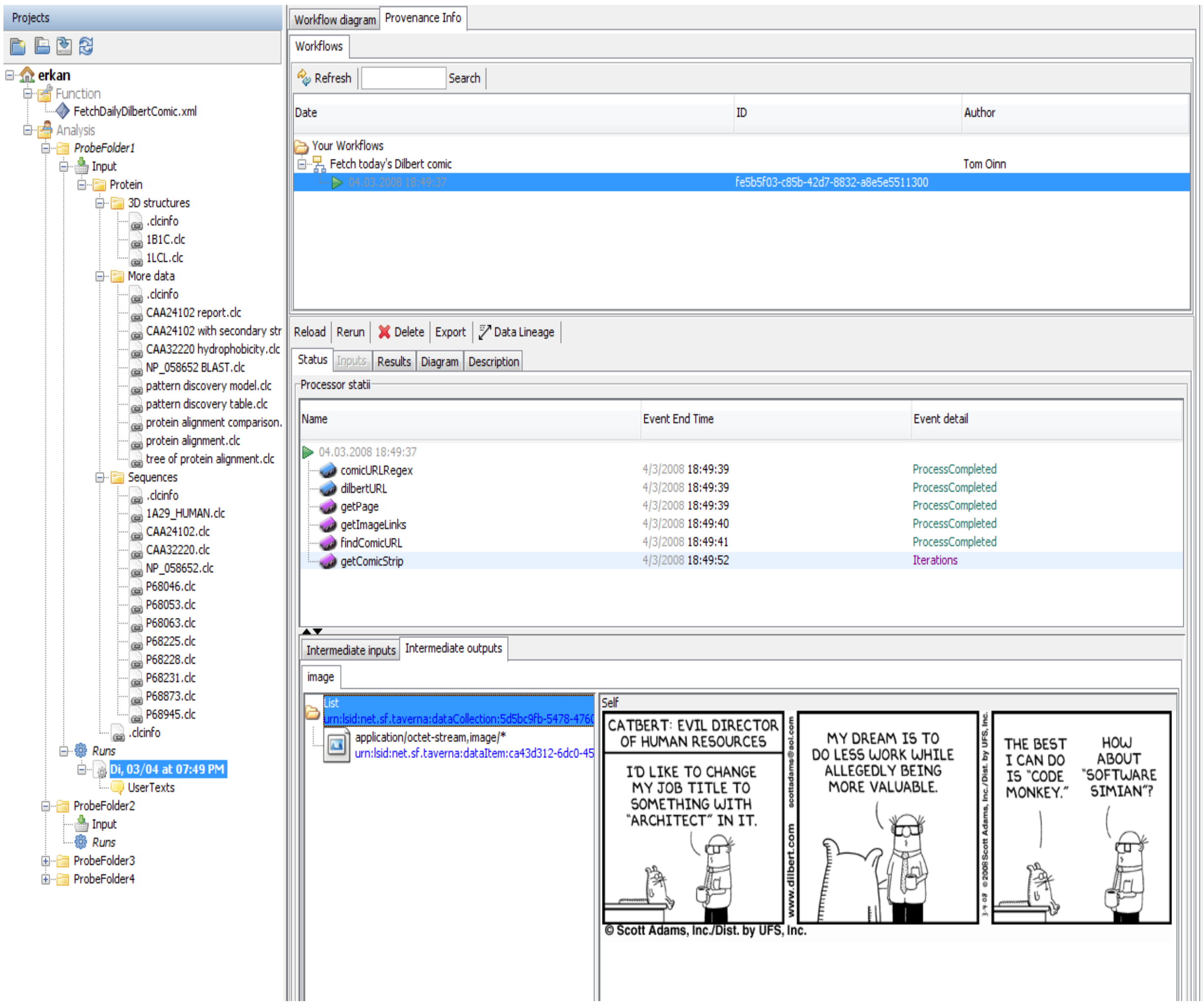


Figure 35: Browse Provenance Action

Clicking the "Browse Provenance" action adds a middle pane in a new tab showing the provenance browser. The user can see the workflow diagram, process list and input or outputs of projects from appropriate windows.

Moreover, the workflow can be re-run by clicking the "rerun" button or reloading it to the workflow browser by clicking the "reload" button.

Selecting one of the processes from the list shows intermediate results and inputs (see Figure 36).



Projects

- erkan
  - Function
    - FetchDailyDilbertComic.xml
  - Analysis
    - ProbeFolder1
      - Input
        - Protein
          - 3D structures
            - .clcinfo
            - 1B1C.clc
            - 1LCL.clc
          - More data
            - .clcinfo
            - CAA24102 report.clc
            - CAA24102 with secondary str
            - CAA32220 hydrophobicity.clc
            - NP_058652 BLAST.clc
            - pattern discovery model.clc
            - pattern discovery table.clc
            - protein alignment comparison.
            - protein alignment.clc
            - tree of protein alignment.clc
          - Sequences
            - .clcinfo
            - 1A29_HUMAN.clc
            - CAA24102.clc
            - CAA32220.clc
            - NP_058652.clc
            - P68046.clc
            - P68053.clc
            - P68063.clc
            - P68225.clc
            - P68228.clc
            - P68231.clc
            - P68873.clc
            - P68945.clc
          - .clcinfo
      - Runs
        - Di, 03/04 at 07:49 PM
          - UserTexts
    - ProbeFolder2
      - Input
      - Runs
    - ProbeFolder3
    - ProbeFolder4

Workflow diagram | Provenance Info

Workflows

Refresh | Search

Date	ID	Author
Your Workflows		
Fetch today's Dilbert comic		Tom Oinn
04.03.2008 18:49:37	fe5b5f03-c85b-42d7-8832-a8e5e5511300	

Reload | Rerun | Delete | Export | Data Lineage

Status | Inputs | Results | Diagram | Description

Processor statii

Name	Event End Time	Event detail
04.03.2008 18:49:37		
comicURLRegex	4/3/2008 18:49:39	ProcessCompleted
dilbertURL	4/3/2008 18:49:39	ProcessCompleted
getPage	4/3/2008 18:49:39	ProcessCompleted
getImageLinks	4/3/2008 18:49:40	ProcessCompleted
findComicURL	4/3/2008 18:49:41	ProcessCompleted
getComicStrip	4/3/2008 18:49:52	Iterations

Intermediate inputs | Intermediate outputs

image

List
urn:lsid:net.sf.taverna:dataCollection:5d5bc9fb-5478-476
application/octet-stream,image/*
urn:lsid:net.sf.taverna:dataItem:ca43d312-6dc0-45

Self

CATBERT: EVIL DIRECTOR OF HUMAN RESOURCES
I'D LIKE TO CHANGE MY JOB TITLE TO SOMETHING WITH "ARCHITECT" IN IT.
MY DREAM IS TO DO LESS WORK WHILE ALLEGEDLY BEING MORE VALUABLE.
THE BEST I CAN DO IS "CODE MONKEY."
HOW ABOUT "SOFTWARE SIMIAN"?
www.dilbert.com
© Scott Adams, Inc./Dist. by UFS, Inc.

Figure 36: Provenance Browser

Chapter 5

5 Evaluation & Discussion

In this chapter I will evaluate the present work by considering advantages and disadvantages of using project management tools with provenance support and comparing the achieved results with alternative solutions currently available.

5.1 Evaluation of the Work

Workflow based analysis software uses various kinds of tools. While some tools are computationally intensive, some are not. While some tools use huge amount of data and accordingly generate huge data outputs, some produce only meaningful numbers. Conversely, some use text files as an input and produce tiff images requiring a few hundred mega-bytes of storage space.

Thus, the performance of project management systems with web services entirely depends on the current workflow used, since the time needed to transfer the data via the network is usually not more than the time required for computer processing the data.

Considering these facts, two different workflows were used for evaluating PPMS. One of them requires significant computational power and time while the other one requires minimum computational power.

5.1.1 Related Work

The two workflows used here are from myExperiment.org [42] a Virtual Research Environment that enables sharing and executing scientific workflows.

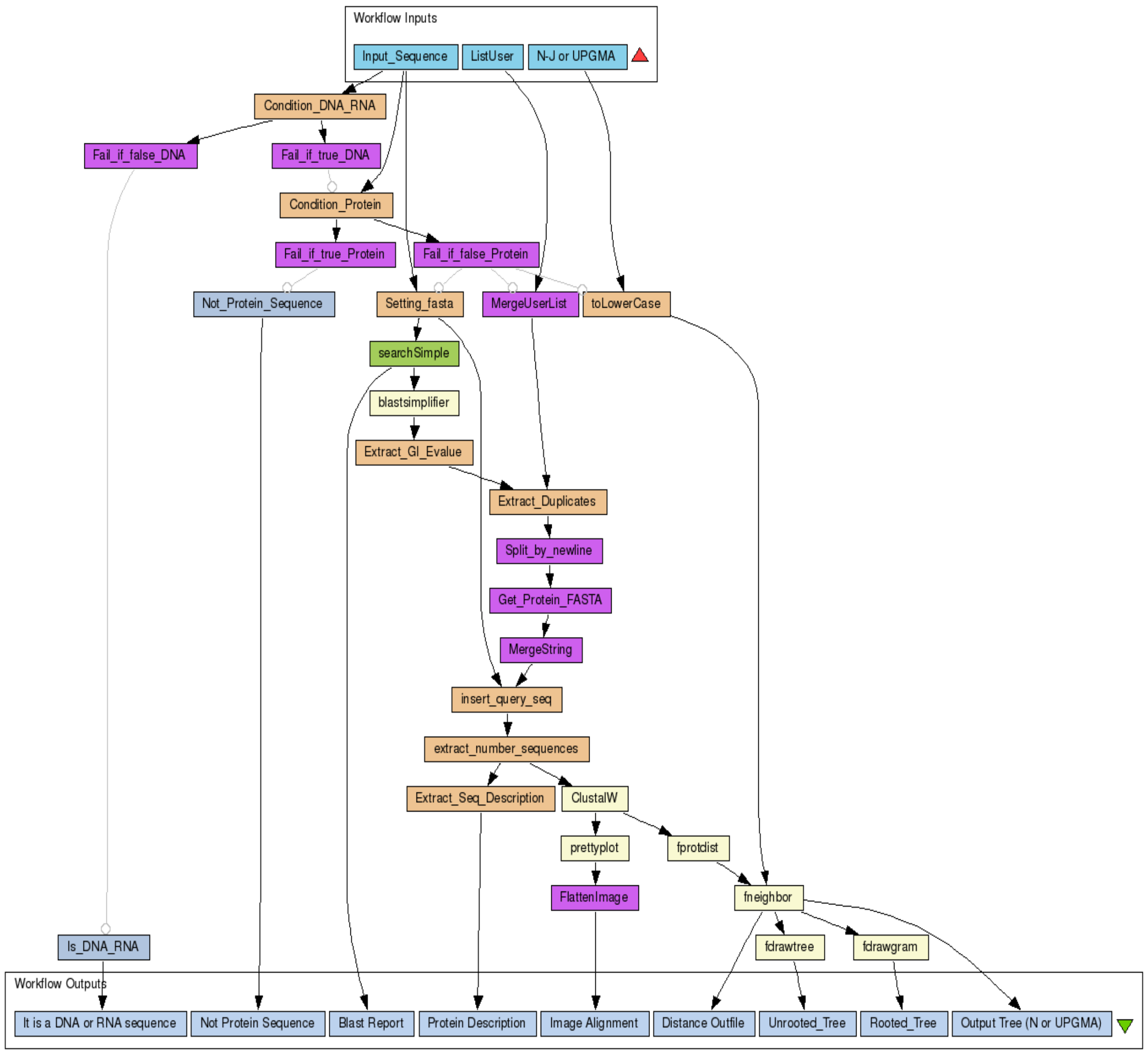


Figure 37 : Workflow for Protein Sequence Analysis

The first workflow used is “Workflow for Protein Sequence Analysis” (see Figure 37) Here is the description about what actually this workflow does:

This workflow performs a generic protein sequence analysis. In order to do that a novel protein sequence enters into the software along with a list of known protein identifiers chosen by the biologist to perform a homology search, followed by a multiple sequence alignment and finally a phylogenetic analysis. [43].

The second workflow is the simplest one, needs no computation but it simply retrieves an image by connecting to a remote site (see Figure 38).

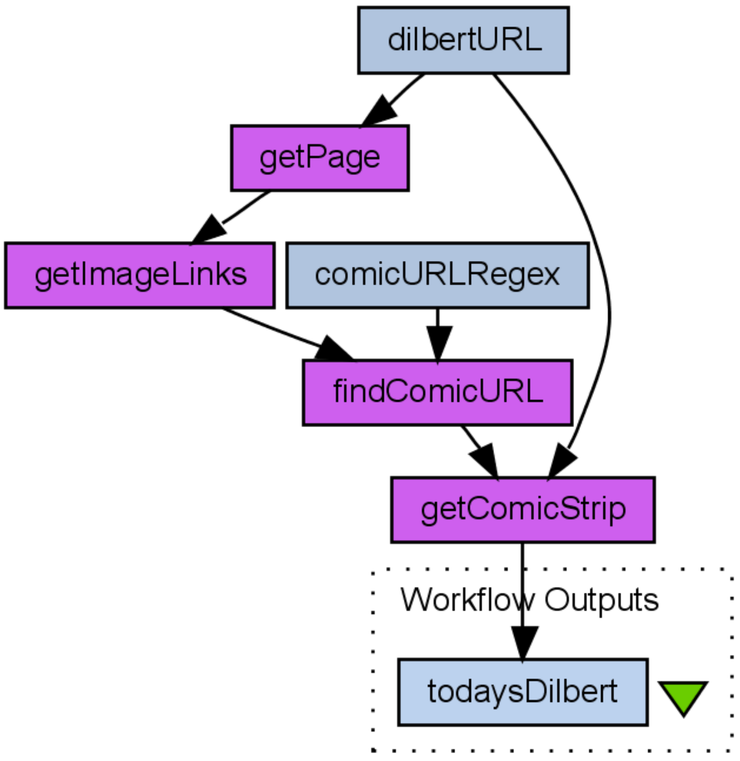


Figure 38 : Fetch daily Dilbert comic

The workflows run using the ARC Analysis Workbench version 1.0 which is based on Taverna core 1.6.2 that, as an enactor, again has the engine FreeFluo version 1.6.2.0.

The client software is running on a notebook with 2.2 Intel Core2 Duo central processor units, 2 Gb memory capacity and it is using a broad-band internet connection.

The PPMS server is running on a Glassfish v2 application server. The computer itself has two Intel® 64 Bit Xeon® Processors with 2.0 GHz clock speed, 8 Gb memory capacity and is finally connected to the internet using a broad-band connection.

5.1.2 Performance Evaluation

5.1.2.1 Performance facts of the first workflow

Input	*Input_sequence*	*List_User*	*Cluster Method*
List 1	MSGEPELIELRELAPAGRAGKGRTRLERANALRIARGTA CNPTRQLVPGRGHRFQPAGPATHTWCDLCGDFIWGVV RKGLQCAHCKFTCHYRCRALVCLDCCGPRDLGWEPAVE RDTNVDEPVEWETPDLSQAEIEQKIKEYNAQINSNLFMS LNKDGSYTGFIKVQLKLVRPVSVPSSKKPPSLQDARRGP GRGTSVRRRTSFYLPKDAVKHLHVLSRTRAREVIEALLR KFLVVDDPRKFALFERAERHGQVYLRKLLDDEQPLRLRL LAGPSDKALSFVLKENDSGEVNWDAFSMPELHNFLRILQ REEEEHLRQILQKYSYCRQKIQEALHACPLG	Q96TS5 Q12575	Neighbor- Joining algorithm
List 2	AITRRVACLDGVNTATNAACCALFAVRDDIQQNL FDGGECGEEVHESLRLTFHDAIGISPSLAATGKFGG GGADGSIMIFDDIEPNFHANNGVDEIINAQKPFVAK HNMTAGDFIQFAGAVGVSNCPGAPQLSFFLGRPA	AAA33739 BAD89745	UPGMA algorithm

Table 2 : Input used for Protein Sequence Analysis Workflow

The protein sequence analysis workflow is run using the input values from Table 2. This workflow has 28 processes including conditional checks.

This workflow approximately generates **10 Mb** of data; **14415 rows** are written in **7 tables** in **2 different** databases.

A run with local provenance took (Taverna with Provenance plug-in):

Run 1 on 09.03.08 at 22:30h:

Time for execution	**740.96 Seconds**
Time for provenance Storage	753.485 Seconds

Run 2 on 10.03.08 14:23h:

Time for execution	**297.773 Seconds**
Time for provenance Storage	310.050 Seconds

A run with web service based remote provenance support took:

Run 1 on 09.03.08 at 22:50h:

Time for execution	**747.12 Seconds**
Time for provenance Storage	778.31 Seconds

Run 2 on 10.03.08 15.03h:

Time for execution	**282.142 Seconds**
Time for provenance Storage	597.480 Seconds

5.1.2.2 Performance facts of the second workflow

This workflow needs no input and has 6 processors including conditional checks.

This workflow generated approximately **400 kB** of data, **712 rows** written in **7 tables** in **2 different** databases.

A run with local provenance took (Taverna with Provenance Plug-In):

Time for execution	**3.947 Seconds**
Time for provenance Storage	16.178 Seconds

Run with web service based remote provenance support took:

Time for execution	**3.307 Seconds**
Time for provenance Storage	31.481 Seconds

Further runs have approximately taken same times for both execution and storage.

5.1.3 Results

It is certainly clear that there are far more criterions for evaluating the usability of the PPMS than just the performance. The ability to work on an analysis in a team, which is enabled by PPMS due to the remote storage facility, is often indispensable.

The project context is essential either. With project management the data produces by workflows will be stored in a structured way the researcher wishes to have without any further concentration. ARC Analysis Workbench with PPMS is the only software by now in the Life Sciences domain offering project management facilities o both local and remote storage mechanisms.

Also the project context is essential. With project management the data produced by the workflows are stored in a structured way according to the researchers' wish. The ARC Analysis Workbench with PPMS is the only software by now in the life sciences domain offering project management facilities on both local and remote storage mechanisms.

The first workflow used for this evaluation is more complex than the second workflow, therefore it naturally requires more time to accomplish its work and it generates noticeably more data than the “Fetch daily Dilbert comic” workflow.

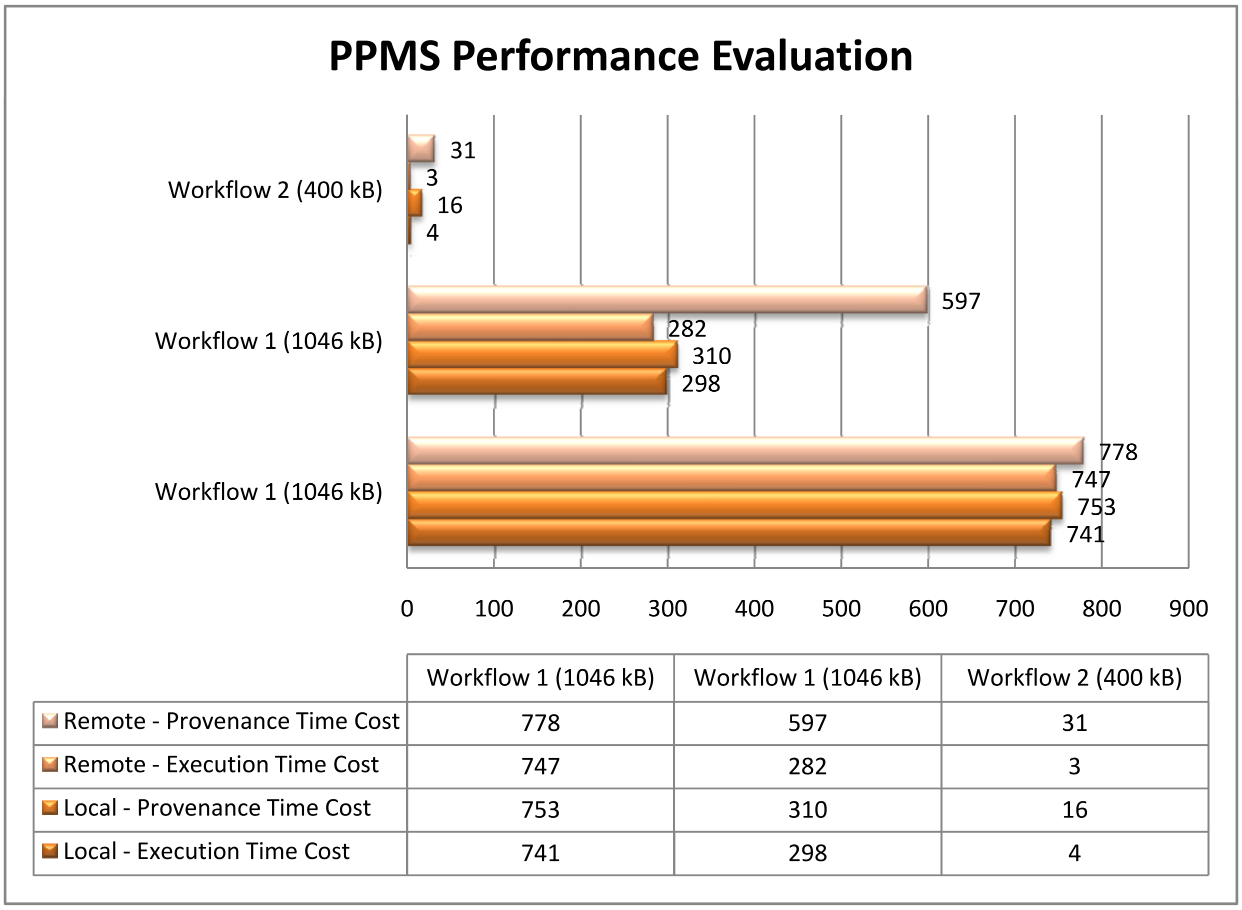
Workflow 1 is executed a few times on two days. Because workflow 1 has some components which are remotely executed, the execution time depends on the free resources available.

Figure 40 shows that two different runs of the same workflow at different times can give entirely different performance results.

It is also clear that the longer an execution process takes, the lesser is the difference between a web services based project management and an analysis software without PPMS support. Since execution and provenance capturing are running on different threads, if an execution thread is blocked somewhere by some demanding process, the provenance thread meanwhile sends the captured data in queue. That means that, in the best case, the difference between a PPMS and a local provenance is the time required for the storage of the output of the last processor in the workflow on a local storage device and on the server. In the worst case, the execution takes only little time but it puts a huge amount of data to the provenance queue for storing.

5.2 Discussion with future views

Workflow based analysis solutions are very popular in biomedical research and are becoming even more accepted than ever due to the great initiatives' support for such software. The research community is aware of the solutions but the complexity of these tools is still a



	Workflow 1 (1046 kB)	Workflow 1 (1046 kB)	Workflow 2 (400 kB)
■ Remote - Provenance Time Cost	778	597	31
■ Remote - Execution Time Cost	747	282	3
■ Local - Provenance Time Cost	753	310	16
■ Local - Execution Time Cost	741	298	4

Figure 40 : PPMS performance evaluation

problem. Even though the lack of data and project management support becomes less important because of other even bigger challenges, the management of data is a significant improvement if it is implemented.

As discussed in this work, the two (actually there are far more tools without PM support than two, of course) workflow based analysis platforms have no data management facility but they put the workflow execution in the center of their solutions perspective. One should at first create a workflow consisting of different components and then he must show the tool where the input data are.

One other important requirement in the life sciences domain is to be able to repeat past analyses. Researchers need to understand and re-evaluate their own or other's analyses with minimum data loss.

There are some studies conducted on that topic in different working areas but the best one for the life sciences domain is LogBook which is developed in the myGrid component suite. Since it is not built-in in Taverna, it is not supported by a management system and therefore it displays all the workflow runs without having any analysis or project context, with its actual capabilities remaining widely un-used.

PPMS brings all the analysis data together in a project context. That alone brings a significant facilitation. Researchers can execute their analyses without concern about where and in which structure the data will be stored. They can simply concentrate on working and in the end the data are hierarchically stored in a chosen file system or on a remote server, where data can be uploaded and used from every terminal without having to carry the data on an e.g. portable storage device.

A research team can then decide if an analysis is something different from the project types supported by PPMS. In this case, they can easily define the project structure, implement the necessary interfaces and use it within the PPM system. The PPMS is easily extendable.

Other types of storage mechanisms can easily be integrated in PPMS as well. RMI support, local database support, RESTful Web Services support and others can easily be developed and integrated in the system.

The provenance support in PPMS sets provenance in an analysis context where it becomes perfectly clear within which analysis project the workflow was run, so that the purpose and the context of its execution is clearly comprehensible.

However, some improvements in PPMS would also be advantageous. IBM's Boca framework for metadata storage would offer different features in a larger scale for provenance support.

Moreover, PPMS client could have a caching feature which caches the provenance data before sending them via the network in order to minimize the disadvantages of using a remote storage mechanism. Besides that, a workflow runs' provenance data can be sent in threads and these threads can be displayed in a list so that the data get transferred quicker and the user has a sense of what happens at the moment.

Finally an authentication of users on the server would be appreciated enabling a user-based data management.

The growth-speed of software solutions with many different features in the life sciences domain is really enormous. This gives us hope that research will more and more be alleviated.

Appendix A

XML Documents

A. 1 Process Documentation Record Schema

Below is the full schema for process documentation record and record acknowledgement messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xmlns:pr="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
           xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.pasoa.org/schemas/version023s1/PStruct.xsd ..\PStruct.xsd ">
   <xs:annotation>
      <xs:documentation>
         The Provenance Store Record schema
```

```
        Author: Paul Groth Last
        Modified: 30 August 2005
        Copyright (c) 2006 University of Southampton
        See pasoalicense.txt for license information.
        http://www.opensource.org/licenses/mit-license.php
    </xs:documentation>
</xs:annotation>
<xs:import
    namespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
    schemaLocation="../PStruct.xsd" />
<xs:element name="record" type="pr:Record" />
<xs:element name="recordAck" type="pr:RecordAck"/>
<xs:element name="content" type="pr:Content" />
<xs:complexType name="RecordAck">
    <xs:sequence>
        <xs:element name="synch_ack" minOccurs="0"
            maxOccurs="unbounded" type="pr:SynchAck">
            <xs:annotation>
                <xs:documentation>
                    If the provenance store is being accessed under
                    a synchronous connection, (i.e. remote procedure
                    call style) the provenance store may return a
                    synch_ack instead of an ack element. Under such
                    a situation, the provenance store does not need
                    to parse the incoming message in order to send
                    an acknowledgment.
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="ack" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="contentName">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="interactionPAssertion"/>
```

```
                          <xs:enumeration value="actorStatePAssertion"/>
                        <xs:enumeration value="relationshipPAssertion"/>
                          <xs:enumeration value="exposedInteractionMetaData"/>
                          <xs:enumeration value="submissionFinished"/>
                        </xs:restriction>
                    </xs:simpleType>
                  </xs:element>
                  <xs:element ref="ps:interactionKey"/>
                  <xs:element ref="ps:viewKind"/>
                  <xs:element ref="ps:localPAssertionId" minOccurs="0"/>
              </xs:sequence>
          </xs:complexType>
      </xs:element>
      <xs:element name="ERROR" minOccurs="0" maxOccurs="1" type="xs:string">
          <xs:annotation>
              <xs:documentation>
                  This field should be pressent if the messageName
                  element's value is ERROR. As of yet we do not
                  define the type of the error message that can be
                  returned by the provenance store.
              </xs:documentation>
          </xs:annotation>
      </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Record">
  <xs:sequence>
      <xs:element name="identifiedContent" type="pr:IdentifiedContent" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="IdentifiedContent">
  <xs:sequence>
      <xs:element ref="ps:interactionKey"/>
      <xs:element ref="ps:viewKind"/>
      <xs:element ref="ps:asserter"/>
      <xs:element ref="pr:content" maxOccurs="unbounded"/>
```

```
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Content">
        <xs:choice>
            <xs:element ref="ps:interactionPAssertion"/>
            <xs:element ref="ps:actorStatePAssertion" />
            <xs:element ref="ps:relationshipPAssertion"/>
            <xs:element ref="ps:exposedInteractionMetaData"/>
            <xs:element name="submissionFinished" type="xs:int" />
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="SynchAck"></xs:complexType>
</xs:schema>
```

A. 2 Sample Process Documentation Record WSDL for PASOA

WSDL document for process documentation record and acknowledgement messages.

```
<?xml version="1.0"?>
<definitions name="PRecord"
            targetNamespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.wsdl"
            xmlns:tns="http://www.pasoa.org/schemas/version023s1/record/PRecord.wsdl"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:pr="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd">
    <documentation>
        The Provenance Store recording port type and messages
        Author: Paul Groth
        Last Modified: Feb 1 2005
        Copyright (c) 2006 University of Southampton
        See pasoalicense.txt for license information.
        http://www.opensource.org/licenses/mit-license.php
    </documentation>
    <import namespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
            location="./PRecord.xsd" />
    <message name="Record">
        <part name="body" element="pr:record"/>
    </message>
    <message name="RecordAck">
        <part name="body" element="pr:recordAck"/>
    </message>
    <portType name="RecordPortType">
        <operation name="Record">
            <input message="tns:Record"/>
            <output message="tns:RecordAck"/>
        </operation>
    </portType>
```

A. 3 Function Based Project Management Project Structure Definition File

```
</definitions>
<?xml version="1.0" encoding="utf-8"?>
<!--
    Document    : FunctionBasedPMDef.xml
    Author      :   Erkan Dilaveroglu
-->
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:java="http://www.relaxer.org/xmlns/relaxer/java"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
    <start>
        <element name="FBPM">
            <attribute name="projectName">
                <data type="string">
                    <param name="minLength">1</param>
                    <param name="maxLength">25</param>
                </data>
            </attribute>
            <attribute name="storageURI">
                <data type="anyURI" />
            </attribute>
            <group>
                <ref name="mfunction" />
                <ref name="mprobe" />
            </group>
        </element>
    </start>
    <define name="mfunction">
        <element name="function">
            <!-- function folder display name -->
            <ref name="folderDisplayAtt" />
            <ref name="uri" />
            <optional>
                <ref name="mfile" />
            </optional>
            <optional>
                <element name="params">
                    <ref name="uri" />
                    <ref name="mfile" />
                </element>
            </optional>
        </element>
    </define>
    <define name="mprobe">
        <element name="probes">
            <ref name="folderDisplayAtt" />
            <ref name="uri" />
            <zeroOrMore>
                <element name="probeFolders">
                    <ref name="folderDisplayAtt" />
                    <ref name="uri" />
                    <element name="input">
                        <ref name="folderDisplayAtt" />
                        <ref name="uri" />
                        <interleave>
                            <zeroOrMore>
                                <ref name="mfolder" />
                            </zeroOrMore>
                            <zeroOrMore>
                                <ref name="mfile" />
                            </zeroOrMore>
                        </interleave>
                    </element
                    <element name="runs">
                        <ref name="folderDisplayAtt" />
                        <ref name="uri" />
                        <zeroOrMore>
                            <ref name="mruns" />
                        </zeroOrMore>
                    </element>
                </element>
            </zeroOrMore>
        </element>
    </define>
    <define name="mcategorie">
        <attribute name="categorie">
            <choice>
                <value>FUNCTION</value>
                <value>PARAMETER</value>
                <value>INPUT</value>
                <value>OUTPUT</value>
                <value>USERTEXT</value>
            </choice>
        </attribute>
    </define>
    <define name="folderDisplayAtt">
```

```
    <attribute name="displayName">
        <data type="string">
            <param name="minLength">1</param>
            <param name="maxLength">25</param>
        </data>
    </attribute>
</define>
<define name="uri">
    <attribute name="storageURI">
        <data type="anyURI" />
    </attribute>
</define>
<define name="mfile">
    <element name="file">
        <ref name="uri" />
        <ref name="mcategorie" />
        <attribute name="fileExt">
            <data type="string">
                <param name="minLength">1</param>
                <param name="maxLength">10</param>
            </data>
        </attribute>
        <optional>
            <ref name="folderDisplayAtt" />
        </optional>
        <attribute name="name">
            <data type="string">
                <param name="minLength">1</param>
                <param name="maxLength">25</param>
            </data>
        </attribute>
        <optional>
            <element name="data">
                <data type="base64Binary" />
            </element>
        </optional>
    </element>
</define>
<define name="mfolder">
    <element name="folder">
        <ref name="uri" />
        <ref name="mcategorie" />
        <attribute name="name">
            <data type="string">
                <param name="minLength">1</param>
                <param name="maxLength">25</param>
            </data>
        </attribute>
        <optional>
            <ref name="folderDisplayAtt" />
        </optional>
        <interleave>
            <zeroOrMore>
                <ref name="mfolder" />
            </zeroOrMore>
            <zeroOrMore>
                <ref name="mfile" />
            </zeroOrMore>
        </interleave>
    </element>
</define>
<define name="mruns">
    <element name="aRun" java:className="WorkflowRun">
        <ref name="uri" />
        <attribute name="workflowRunId"/>
        <attribute name="date">
            <data type="dateTime"/>
        </attribute>
        <element name="userText" >
            <ref name="folderDisplayAtt" />
            <ref name="uri" />
            <oneOrMore>
```

```
                    <ref name="mfile" />
                </oneOrMore>
            </element>
        </element>
    </define>
</grammar>
```

Appendix B

VDL Examples

B.1. Dependency Chain

In the following example, file2, the output of trans1 produced by derivation usetrans1, is used as the input to trans2 in derivation usetrans2. This is the essence of data provenance tracking in Chimera [21].

```
TR trans1( output a2, input a1 )
{
argument stdin = ${input:a1};
argument stdout = ${output:a2};
exec = "/usr/bin/app1";
}
TR trans2( output a2, input a1 )
{
argument stdin = ${input:a1};
argument stdout = ${output:a2};
exec = "/usr/bin/app2";
}
DV trans1
(
 a2=@{output:"file2"},
a1=@{input:"file1"}
);
DV trans2
(
 a2=@{output:"file3"},
a1=@{input:"file2"}
);
```

B.2. Compound Transformation

Three simple transformations, and the fourth transformation, trans4, which is a compound transformation composed of calls to trans1, 2, and 3: [21]

```
TR trans1( output a2, input a1 )
{
argument = "...";
argument stdin = ${input:a1};
argument stdout = ${output:a2};
profile hints.pfnHint = "/usr/bin/app1";
}
TR trans2( output a2, input a1 )
{
argument = "...";
argument stdin = ${input:a1};
argument stdout = ${output:a2};
exec = "/usr/bin/app2";
}
TR trans3( input a2, input a1, output a3 )
{
```

```
argument parg = "-p foo";
argument farg = "-f "${input:a1};
argument xarg = "-x -y -o "${output:a3};
argument stdin = ${input:a2};
exec = "/usr/bin/app3";
}
TR trans4( input a2, input a1, inout
a5=@{inout:"anywhere":""}, inout
a4=@{inout:"somewhere":""}, output a3 )
{
trans1( a2=${output:a4}, a1=${a1} );
trans2( a2=${output:a5}, a1=${a2} );
trans3(
 a2=${input:a5}, a1=${input:a4},
a3=${output:a3}
);
```

Appendix C

Abbreviations

PPMS	:	Portable Project Management System
SOA	:	Service Oriented Architecture
OECD	:	Organization for Economic Co-operation and Development
DNA	:	Deoxyribonucleic acid
WSDL	:	Web Service Definition Language
PASOA	:	Provenance Aware Service Oriented Architecture
PReP	:	P-Assertion Recording Protocol
PreServ	:	Provenance Recording for Services
WS	:	Web Service (Soap Based)
HTTP	:	Hypertext Transfer Protocol
PS	:	Provenance Store

VDS	:	Virtual Data System
VDC	:	Virtual Data Catalog
VDL	:	Virtual Data Language
XML	:	eXtendable Markup Language
TR	:	Transformation
DV	:	Derivation
SQL	:	Structured Query Language
LFN	:	Logical File Name
UML	:	Unified Modeling Language
IDE	:	Integrated Development Environment
IBPM	:	Input Based Project Management
FBPM	:	Function Based Project Management
CRUD	:	Create-Read-Update-Delete
URL	:	Uniform Resource Locator
URI	:	Uniform Resource Identifier
POJO	:	Plain Old Java Object

Bibliography

[1]. Microarray. [Online] [Cited: 08 14, 2007.] http://www.everythingbio.com/glos/definition.php?word=Microarray.

[2]. *P-assertion Recording for Services.* [Online] 5 20, 2007. http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/SoftWare.

[3]. *Position Statement: Musing on Provenance, Workflow and (Semantic Web) Anno-tation for Bioinformatics.* **c.Goble.** Chicago : Workshop on Data Derivation and Provenance, 2002.

[4]. *A Survey of Data Provenance in e-Science.* **Yogesh L.Simmhan, Beth Plale, and Dennis Gannon.** s.l. : SIGMOD Record, 11.5.2005. Vols. 34, No. 3.

[5]. *Research Problems in Data Provenance.* **Tan, Wang-Chiew.** 2004, IEEE Data Engineering Bulletin, pp. 27(4):42-52.

[6]. *Exploring Provenance in a Distributed Job Execu-tion System.* **F.Naughton, Christine F.Reilly and Jeffrey.** Chicago : International Provenance and Annotation Workshop, 2006.

[7]. *Scientific Annotation Middleware.* [Online] 5 30, 2007. http://collaboratory.emsl.pnl.gov/docs/collab/sam/.

[8]. *VDS - The GriPhyN Virtual Data System.* [Online] 5 30, 2007. http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain.

[9]. *myGrid.* [Online] 6 5, 2007. http://www.mygrid.org.uk/.

[10]. *Provenance Aware Service Oriented Architecture (PASOA).* [Online] 6 6, 2007. http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome.

[11]. *The Earth System Service Workbench.* [Online] 6 10, 2007. http://essw.bren.ucsb.edu/projects/proj-essw.htm.

[12]. *Collaboratory for Multi-scale Chemical Science.* [Online] 6 12, 2007. http://cmcs.org/.

[13]. *P-Assertion Recording Protocol.* **Paul Groth, Victor Tan, Simon Miles, John Ibbotson, and Luc Moreau.** Southampton : Technical Report, ECS, University of Southampton, 8 24, 2006.

[14]. *Web Services Security.* [Online] 6 13, 2007. http://www-128.ibm.com/developerworks/library/specification/ws-secure/.

[15]. *Web Services Trust Language.* [Online] 6 13, 2007. http://www-128.ibm.com/developerworks/library/specification/ws-trust.

[16]. *Web Services Secure Conversation Language.* [Online] 6 13, 2007. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secon/.

[17]. *An Architectur for Provenance Systems.* **Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau.** Southampton : Technical report, Electronics and Computer Science, University of Southampton, 2006.

[18]. *PReServ: Provenance Recording for Services.* **Paul Groth, Simon Miles, and Luc Moreau.** Nottingham,UK : s.n., 2005. In Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05).

[19]. *Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation.* **Ian Foster, Jens Vöckler, Michael Wilde, and Yong Zhao.** s.l. : 4th International Conference on Scientific and Statistical Database Management (SSDBM'02), 2002.

[20]. *Virtual Data Grid Middleware Services for Data-Intensive Science.* **Yong Zhao, Michael Wilde, Ian Foster, Jens Vöckler, E.Glibert, T.Jordan, and E.Quigg.** Toronto, Ontario, Canada : Concurrency, Practice and Experience, 2004.

[21]. *The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration.* **Ian Foster, Jens Vöckler, Michael Wilde, and Yong Zhao.** s.l. : Conference on Innovative Data Research CIDR, 2003.

[22]. **Yong Zhao, Michael Wilde, and Ian Foster.** Applying the Virtual Data Provenance Model. *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW2006), Lecture Notes in Computer Science.* Springer, 2006.

[23]. Taverna. [Online] [Cited: 08 15, 2007.] http://taverna.sourceforge.net/index.php.

[24]. GNU LGPL . [Online] [Cited: 08 15, 2007.] http://www.opensource.org/licenses/lgpl-license.php.

[25]. OMII-UK. [Online] [Cited: 08 16, 2007.] http://www.omii.ac.uk/.

[26]. Biomart. [Online] [Cited: 08 17, 2007.] www.biomart.org.

[27]. Biomoby. [Online] [Cited: 08 17, 2007.] www.biomoby.org.

[28]. INB Bioinformatics and Genomics Node. [Online] [Cited: 08 16, 2007.] http://genome.imim.es/webservices/index.html.

[29]. INB Example Workflow. [Online] [Cited: 08 17, 2007.] http://genome.imim.es/webservices/workflows/PromoterAnalysisWorkflow_coexpressedGenesMode_Fasta_Version.xml.

[30]. INB Example Workflow Dataset. [Online] [Cited: 08 17, 2007.] http://genome.imim.es/webservices/examples/coexpressed_genes_FastaSeqs_Document.xml.

[31]. Scufl Web Service Workflow Language. [Online] [Cited: 08 17, 2007.] http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html.

[32]. Taverna LabBook Plug-In. [Online] [Cited: 08 19, 2007.] http://www.mygrid.org.uk/wiki/Mygrid/TavernaProvenancePluginOneOne.

[33]. *Kepler Home.* [Online] [Cited: 2 20, 2008.] http://kepler-project.org/.

[34]. *Ptolemy II Home.* [Online] 2 20, 2008. http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[35]. FreeFluo Workflow Enactor. [Online] 2 10, 2008. http://freefluo.sourceforge.net/.

[36]. *Relaxer Home.* [Online] [Cited: 2 15, 2008.] http://www.relaxer.jp/index.html.

[37]. Beginning XML 4th Edition. [book auth.] Jeff Rafter, Joe Fawcett, Eric van der Vlist, Danny Ayers, Jon Duckett, Andrew Watt, Linda McKinnon David Hunter. s.l. : Wrox, May 2007.

[38]. *XML Schema Datatypes.* [Online] 2 11, 2008. http://www.w3.org/2001/XMLSchema-datatypes.

[39]. *Java 6 API.* [Online] [Cited: 2 25, 2008.] http://java.sun.com/javase/6/docs/api/.

[40]. **Stephen Stelting, Olav Maassen.** *Applied Java™ Patterns.* s.l. : Prince Hall, 2001.

[41]. Netbeans IDE Home. [Online] www.netbeans.org.

[42]. myExperiment.org. [Online] [Cited: 03 09, 2008.] www.myexperiment.org.

[43]. Workflow for Protein Sequence Analysis. [Online] [Cited: 03 09, 2008.] http://www.myexperiment.org/workflows/124.