D9.5 Report on the Final ACGT Workflow Environment

D9.6 Report on the Final specifications of meta-data for the ACGT data, tools, services and workflows

Project Number:	FP6-2005-IST-026996
Deliverable id:	D 9.5 & D 9.6
Deliverable name:	Report on the final ACGT Workflow Environment and specifications of meta-data for the ACGT data, tools, services and workflows
Date:	September 1, 2010

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	ACGT
Project Full Name:	Advancing Clinico-Genomic Clinical Trials on Cancer: Open Grid Services for improving Medical Knowledge Discovery
Document id:	D 9.5 & D9.6
Document name:	Report on the final ACGT Workflow Environment and specifications of meta-data for the ACGT data, tools, services and workflows
Document type (PU, INT, RE)	PUB
Version:	1.0
Date:	01/09/2010
Editors: Organisation: Address:	Stelios Sfakianakis FORTH-ICS Foundation for Research and Technology-Hellas (FORTH) Institute of Computer Science N. Plastira 100, Vassilika Vouton, GR-700 13 Heraklion, Crete, Greece

Document type PU = public, INT = internal, RE = restricted

ABSTRACT:

The present document is a merger of the deliverables 9.5 and 9.6. It offers a comprehensive description of the final ACGT workflow environment and the metadata specifications for the data, tools, services, and workflows. The design considerations, architecture, and implementation of the workflow management system are provided in sufficient detail. In the second part of the document the use of metadata annotations in ACGT is described in a unified view by consolidating input from previous deliverables.

KEYWORD LIST: workflows, metadata, services, workflows, semantic interoperability, provenance, orchestration

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	23/03/2010	Draft	Stelios Sfakianakis
0.5	25/08/2010	Draft	Stelios Sfakianakis
0.9	01/09/2010	Draft	Stelios Sfakianakis
1.0	01/09/2010	Final	Stelios Sfakianakis

List of Contributors

- Stelios Sfakianakis, FORTH-ICS
- Lefteris Koumakis, FORTH-ICS
- George Zacharioudakis, FORTH-ICS
- Manolis Tsiknakis, FORTH-ICS
- Dennis Wegener, FhG
- Johan Karlsson, UMA
- Max Garcia, UMA

Contents

Table of Figures

Acronyms

API	Application Programming Interface
BPEL	Business Process Executable Language
CA	Certification Authority
DMS	Data Management Service
DN	X.509 Distinguished Name
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
N3	Notation 3 serialization format for RDF
OPM	Open Provenance Model
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SAWSDL	Semantic Annotations for WSDL
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
WS	Web Services
WSDL	Web Service Description Language
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language

Executive Summary

The ACGT computational environment aims to provide a semantics enabled platform for the implementation of state of the art research experiments in the context of clinical trials for cancer. The *ACGT Workflow Environment* in particular is a set of end user interfaces and components of the grid infrastructure for the design, archiving, execution, and management in general of the ACGT scientific workflows. The aim of this environment is to assist the users in their scientific research by supporting the composition of different data access and knowledge extraction and analytical services into complex workflows. This way the users can extend and enrich the functionality of the ACGT system by reusing existing ACGT compliant services and producing "added value" composite services. This reuse and composition of services is in some sense a programming task where the user actually writes a "program" to realize a scenario or to test a scientific hypothesis. In this dynamic environment describing the informational and computational resources in a machine readable way through the use of *Metadata* makes possible a whole array of advanced yet necessary scenarios like resource identification and discovery, interoperability, dynamic decision making, provenance, and so forth. In essence metadata is the crucial component of every large and complex system that abstracts and captures the essential information of the underlying data independent of their representational details.

In this document we present the final design of the ACGT Workflow Environment and also the approach that has been followed in ACGT in order to support semantics based interoperability and integration through the means of metadata. This report builds upon a number of technical deliverables, extends and complements them with the most recent developments just before the conclusion of the project.

1 Introduction

The significance of the workflow technologies in the bioinformatics and in scientific research in general has been acknowledged and actually proven in practice in the recent years. It is then no surprise that the ACGT consortium in the project's description of work included the provision for delivering a workflow management tool for the end users to design their custom data analyses. Nevertheless during the course of the project we opted to define our own infrastructure for the design and execution of the scientific workflows instead of using some of the existing solutions such as the Taverna Workbench or Triana and Kepler. This was not an easy decision and it was driven by the following forces:

- The security framework of ACGT is very strict and based on the recommendations of the Ethical Board the technologies used need to provide a certain set of security functionality. In particular the need to know the principal user's identity in every step down the pipeline of the data access and service invocation required the "delegation of user's credentials", something that was not provided "out of the box" in any of the well known scientific workflows systems.
- The ACGT architecture that is based on Grid technologies and Service Orientation hinted the provision of a workflow management system that is virtually centralized and accessible remotely by open standards and state of the art technologies. At that time none of the candidates/workflow tools offered a robust "service oriented" deployment solution with central user management supported by the Grid Virtual Organization (VO) mechanisms.
- Better control on the appearance, functionality, metadata integration, roadmap, etc. was an additional objective. It may be argued that such requirements are excuses that hide a "not invented here" syndrome but it is frequently the case that customizing an existing software application is more difficult than building a new one from scratch with the specific requirements driving the design and implementation[1].

So instead of using existing solutions in this area (e.g. the Taverna Workbench which was and still is a strong "competitor" scientific workflow management system) we have decided to do the following:

- Design and implement a new Workflow Editor from scratch. Emphasis was given to the user friendliness, the graphical user interface, and the use of ubiquitous web technologies and standards
- Support the execution of the scientific workflows by expressing them in the standard Business Process Execution Language (BPEL) and then employing an open source production quality BPEL enactor.
- Comply with the security and ethical constraints imposed due to the sensitivity of the data in terms of the patient's anonymity and the privacy and authorization

The outcome is what is collectively called the ACGT workflow environment. The ACGT Workflow Environment and its features and scope have been nicely summarized in the deliverable D9.2 where there's the following passage:

The term "ACGT Workflow Environment" collectively describes the end user interfaces and the grid infrastructure for the design, archiving, execution, and management in general of the ACGT scientific workflows. The aim of this environment is to assist the users in their scientific research by supporting the composition of different data access and knowledge extraction and analytical services into complex workflows. This way the users

[1] As it is frequently said "reinventing the wheel" is necessary if the existing wheels are square.

can extend and enrich the functionality of the ACGT system by reusing existing ACGT compliant services and producing "added value" composite services. This reuse and composition of services is in some sense a programming task where the user actually writes a "program" to realize a scenario or to test a scientific hypothesis.

This document aims to be an update to deliverable 9.2. We present updated information on the design of the ACGT workflow environment and its rationale and report on its implementation. Strongly connected to this environment is the definition of the metadata for data, services, and workflows because such metadata largely drive the composition of services, the execution of the resulted workflows, and the management of the data and the meta information produced. In the second part of this document we therefore describe the ACGT approach on the production and consumption of the metadata descriptions.

2 The Design and Implementation of the ACGT Workflow Environment

2.1 Designing the ACGT Workflow Environment

One of the most important constraints for the management of personal clinical and genomic data is the compliance to the ethical and legal data protection requirements. In ACGT a generic data protection framework has been defined which is based on a technical security infrastructure as well as on organizational measures and contractual obligations [4]. In this setting user authentication and authorization are very important but also auditing of user actions is of great significance so that every access to the data is recorded along with the identity of the principal user who initially requested this access.

Most of these security requirements are dealt with the Grid infrastructure layer. In particular the Grid Security Infrastructure (GSI [23], Deliverable 11.2) supports user authentication through digital signatures and also the delegation of user privileges to a service so that it can retrieve data or perform an action on the user's behalf and without the user's intervention. The delegation mechanism is important because it allows ``single sign on'' for the users of the Grid and also simplifies from the users' point of view the execution of long running chains of services that need authentication since the user is not required to be present and supply his credentials in every processing step.

In agreement with the authors in [3] the decision of using a standard workflow definition language in ACGT has been taken. The scientific workflows domain lacks any such language that is supported by more than one workflow management tool and therefore BPEL [2] was chosen from the business process management world as the most prominent and well supported technology: BPEL is the de facto standard for Web Services orchestration and business process modeling. It features a complete set of control structures that exist in imperative programming languages (if/else, repeat/until, while, etc.) and also constructs for parallel loops and XML processing. BPEL also supports the definition of both abstract and executable workflows and the deployment of a workflow as a new composite web service. Finally its wide popularity and support both from the enterprise and open source worlds give better guarantees about its future and evolution.

Nevertheless the choice of BPEL gave rise to two more requirements. The first one is the provision of a user friendly workflow authoring environment. It is true that there are a couple of open source BPEL editors, for example the Eclipse BPEL Designer[2], but the level of abstraction is too low to be used for bioinformatics workflows: the XML syntax and the BPEL specificities would alienate most domain users. In simple words, such workflow designers require their users to be fairly fluent in BPEL in order to be used effectively. Therefore we saw the need for a new ACGT workflow editor that is accessible over the web and more tailored to the needs of the bioinformatics user community. The requirement for this new workflow editor to be web based was introduced due to our confidence that the World Wide Web has a lot of advantages over traditional standalone desktop applications such as:

- Ubiquitous access from a wide range of networks and devices e.g., laptops, netbooks, and handheld devices like iPADs and tablet PCs, and contemporary smartphones.
- Zero install and upgrade, because the actual execution code is delivered on demand over the web and through the browser.

[2] http://www.eclipse.org/bpel/

- Portability and autonomy, because most of the computation is performed on the server side where also the users' data are kept.
- Strong momentum and community support, since in the last couple of years many web applications, such as the GMail and Google Docs from Google, have been on the spotlight and replacing desktop applications of similar functionality. The Yahoo! Pipes[3] web application was particularly influential because it's a kind of pipeline/workflow editor for the data on the web.

The second prerequisite introduced by the use of BPEL for the ACGT workflows is an infrastructure that would make possible the invocation of the ACGT secure grid services from inside the BPEL-based workflows. Especially the use of GSI for securing Web Services requires a BPEL Engine that is able to invoke them without compromising security. In particular each BPEL workflow should also be a GSI secured service that is able to accept the delegated user credentials and subsequently delegate them further to all the services that need to be contacted in the context of the specific workflow.

Unfortunately BPEL and the Web Services standard security specifications do not comply with such requirements. The whole concept of "credential delegation" is based on the use of "proxy" certificates, which are certificates where the user is considered as a Certification Authority. Proxy certificates are extensions to the standard X.509 certificates and do not enjoy much of support outside the Grid community. This problem required the introduction of "Proxy" services, described in the next section, that following the "Proxy design pattern" [9] offer the bridge between the business process view of BPEL engines and the Grid secure services of the ACGT ecosystem.

2.2 Architecture

In software engineering terms the ACGT Workflow Environment comprises a number of tools, services, and components that provide the infrastructure for the design and management of scientific workflows and, at the same time, address the architectural and security requirements of the ACGT platform. This environment is centralized, hosted inside the ACGT Grid but outside the users' premises, and supports the whole workflow life cycle: from designing new workflows, to their enactment, provenance management, reusing and repurposing. It comprises the following entities:

- Workflow Editor: the graphical web-based tool that enables the design and editing of workflows in an easy and intuitive way
- Workflow Enactor: the BPEL compliant, third party orchestrator for the enactment of the workflows
- Proxy Services: the gateways and representatives of the real ACGT services that present a BPEL and Web Services compliant interface to the Enactor
- Enactor Proxy, for executing the deployed workflows and providing access to the enactment and monitoring information.
- Workflow Repository: the database for keeping workflow definitions, execution state, provenance information, user sessions, etc.

Most of the above are shown in Figure 1 below and will be described in the following sections.

[3] http://pipes.yahoo.com

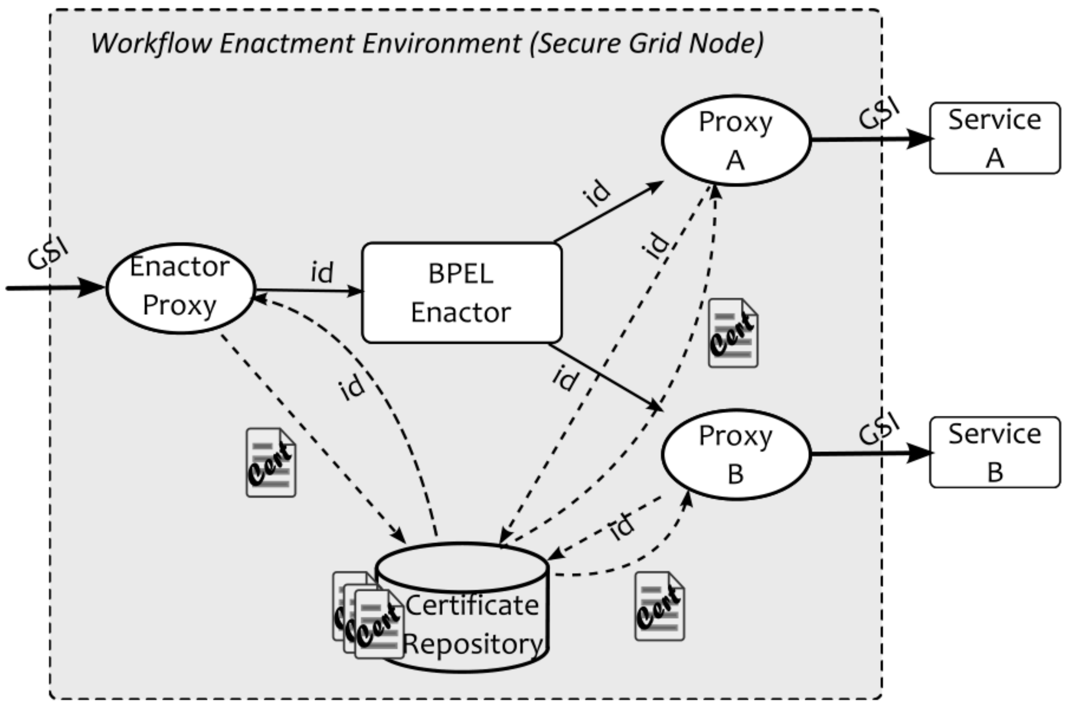


Figure 1 The secure workflow environment through Proxy services

2.3 The workflow editing environment

The ACGT Workflow Editor aims to assist physicians in their scientific research by supporting the composition of different data access, knowledge extraction and analytical services into complex workflows. In its initial design and also its subsequent revisions the following requirements were followed:

- The Workflow Editor should be as user friendly as possible and follow the abstractions the users (clinical and biomedical researchers, bioinformaticians, etc) are familiar with. In particular implementation details such as XML messaging, Grid authentication and authorization, workflow languages and relevant technologies are transparent to the users. An example of this is the case of processing tools and services where the requested functionality is delivered "asynchronously", after the invocation in a specific order of two or more methods. The Workflow Editor presents such services as single abstract activities that given some specific input, the final output is returned, in a single step irrespective of the underlying message exchange protocol.
- Integration with the rest of the ACGT platform is of utmost importance. For example user authentication is performed "under the hood" by the Grid authentication mechanisms using X.509 credentials and MyProxy (see Deliverable 11.2), the Grid file system (DMS, see Deliverable 9.3 "Data and Metadata Management in ACGT") is always accessible through the Editor, and all the services that can be included in the users' workflows are retrieved from the ACGT Metadata Repository.
- Build upon the semantic data integration in ACGT so that, in addition to rudimentary checks based on the syntax of the data, connections between services that represent data exchange are checked based on their semantics as well.
- A desktop application look and feel by supporting the graphical building of the workflow, menus and dialogs

- Efficiency, performance

2.3.1 Implementation

The ACGT workflow editor has been designed as a web application consisting of two components:

- The client side, which runs as a “rich internet application” (RIA) inside the user’s web browser, using ubiquitous web technologies like HTML and Javascript.
- The server side, which is located in a central point and is responsible for more computational heavy or the core “business-logic”.

These two components communicate with each other in a unidirectional, “firewall friendly" way, i.e. the client side makes requests to the server side and not the other way round. The server side of the editor consists of a number of Java Servlets that communicate with the editor through asynchronous web requests, what is called AJAX in technical terms, with a Javascript Object Notation (JSON) formatted payload. With this asynchronous exchange of messages between the browser and the server, the Workflow Editor provides a desktop application's look and feel and interactivity.

Each activity put in a workflow represents a kind of “service” (R script, Data access service, Semantic Mediator query, third party service, etc) that has certain functionality delivered through its outputs when supplied with a set of inputs. Therefore the service descriptions supplied by the server side to the client side workflow editor include in addition to the name and textual description of the service the schema for the input and output parameters. This schema is formatted according to the JSON Schema [27] extended with the additional information providing the semantic type of the parameters. All the information shown in the Editor is retrieved from the ACGT Metadata Repository and therefore is more abstract, mostly supporting semantics based composition of the services.

The users start their session with the Workflow Editor by entering their user name and password into the ACGT Portal. The editor itself is agnostic to the real authentication process that is taking place in the server side and it just requires a user name and a password that are sent through encrypted HTTP authentication mechanisms to the back end. In the current Grid enabled deployment these user name/password credentials are used for MyProxy authentication. After their successful authentication users are presented with a screen similar to what is shown in Figure 2.

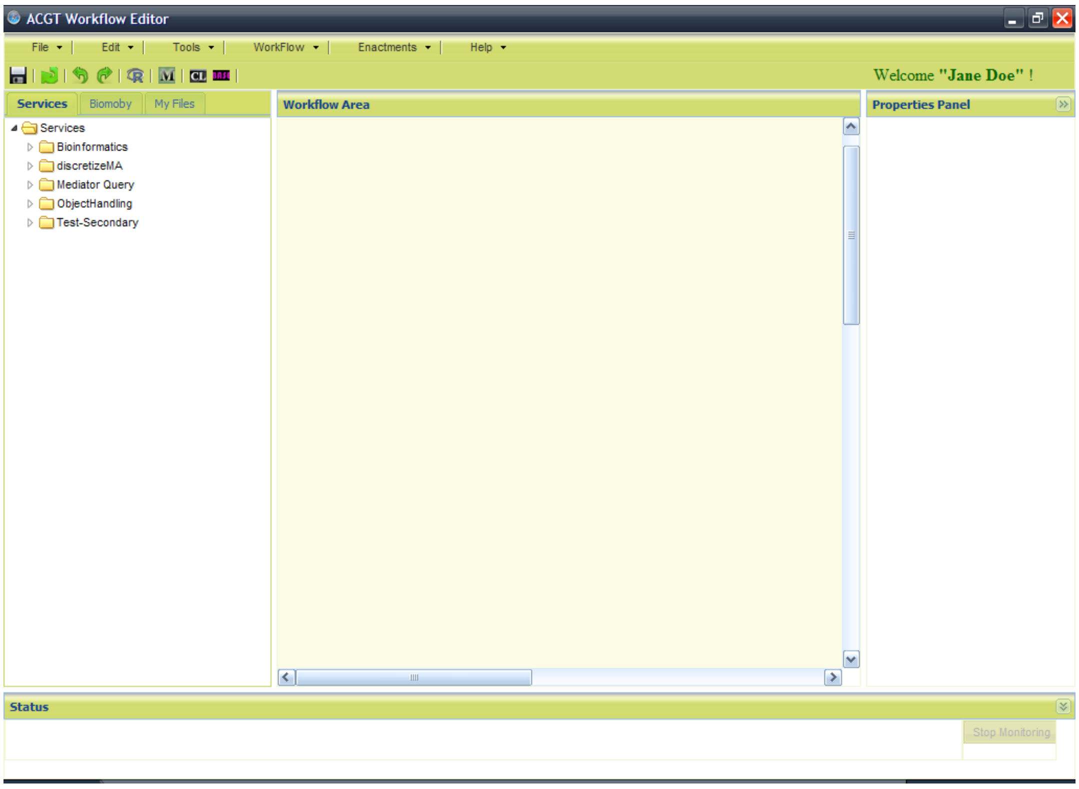


Figure 2 The initial screen of the Workflow Editor

Each user has his/her own virtual private workflow repository for storing his/her workflows. The stored workflows can be subsequently retrieved, changed, and new versions can be saved. The available ACGT services are shown in a tree like structure according to their classification in the back end service registry. Furthermore, the users are allowed to browse their private area in the Grid based file system to locate files that they wish to enter into their workflows.

Following the examples of existing scientific workflow management tools, the ACGT Workflow Editor follows a dataflow, instead of control flow, paradigm: its workflows are directed acyclic graphs (DAGs) where the links between the nodes of the graph represent data that are transmitted between the services and each service will start execution when it has data in all its input parameters. The actual workflow construction is done visually, by “dragging-and-dropping" and connecting, by drawing lines, the output and the inputs of analytical and data management services. For each processing step inside the workflow there is always metadata information about the input and output parameters and in the cases of a GridR data analysis script [22] or a mediator query the script or query code is shown as well.

When a workflow is ready the user is allowed to “enact” it (i.e. run it) by specifying any required input parameters. The execution state of the workflow is readily available to the end user in the editor’s notification area. The active, i.e. currently running, services in the workflow are highlighted and the start and end times of all the completed execution steps are shown (see paragraph “Workflow execution monitoring functionality” below). Users are also able to execute the same or a different workflow multiple times in a row and even close their session with the Editor by logging out. The workflows are executed in the server side and their state is saved so that, when the user connects again, he/she can get immediate feedback about the status and results of the active or finished “enactments”. Finally, a workflow that a user feels confident about its status and usefulness can be published so that also other people can use it. The publication process includes the supply of the necessary metadata (e.g. descriptions of inputs and outputs, functional classification, etc.) and the registration of the executable workflow in the ACGT service registry.

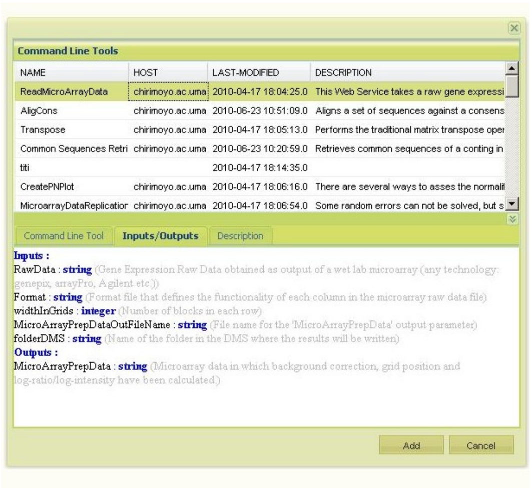
The editor is accessible through the ACGT Portal. More information about the technologies used, user interfaces, the functional and non-functional requirements of the first version of the workflow editor can be found in the Deliverable D9.2. Of course since 2008 there are a lot of updates and new functionalities in the workflow editor.

In the following sections we will describe the new components of the workflow editor and the new look and feel of the environment.

2.3.2 Command Line Tools

Command line tools are internally treated differently from regular services (the same is true for Mediator queries and R-Scripts). To insert a command line tool the user has to click on the corresponding icon from the upper toolbar of the editor: CL

A window appears which displays all the command line tools that are available and can be added to the workflow. As we can see in Figure 3 a special window appears which displays all the available command line tools to the user (available command line tools are tools that he/she has created and tools that are public to the ACGT community). From the command line tools window the user can select one tool and view the inputs/outputs of the tool (command line tools in ACGT workflow environment are considered to be services that have inputs and outputs and run over the grid as web services) and the description of the tool.



NAME	HOST	LAST-MODIFIED	DESCRIPTION
ReadMicroArrayData	chirimoyo.ac.uma	2010-04-17 18:04:25.0	This Web Service takes a raw gene expressi
AligCons	chirimoyo.ac.uma	2010-06-23 10:51:09.0	Aligns a set of sequences against a consens
Transpose	chirimoyo.ac.uma	2010-04-17 18:05:13.0	Performs the traditional matrix transpose oper
Common Sequences Retri	chirimoyo.ac.uma	2010-06-23 10:20:59.0	Retrieves common sequences of a conting in
titi		2010-04-17 18:14:35.0	
CreatePNPlot	chirimoyo.ac.uma	2010-04-17 18:06:16.0	There are several ways to asses the normalit
MicroarrayDataReplicatior	chirimoyo.ac.uma	2010-04-17 18:06:54.0	Some random errors can not be solved, but s

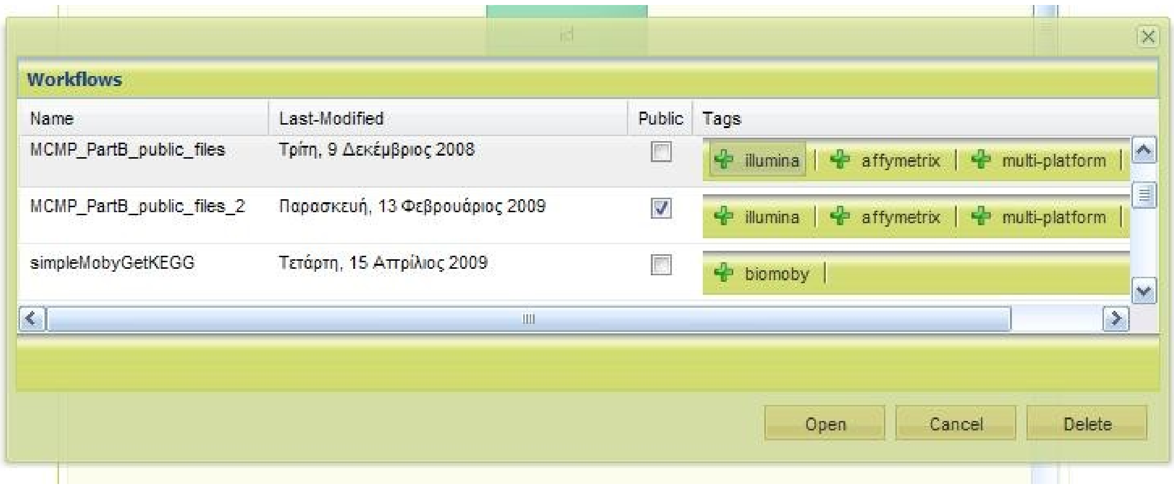
Inputs :
RawData : **string** (Gene Expression Raw Data obtained as output of a wet lab microarray (any technology: genepix, arrayPro, Agilent etc.))
Format : **string** (Format file that defines the functionality of each column in the microarray raw data file)
widthInGrids : **integer** (Number of blocks in each row)
MicroArrayPrepDataOutFileName : **string** (File name for the 'MicroArrayPrepData' output parameter)
folderDMS : **string** (Name of the folder in the DMS where the results will be written)
Outputs :
MicroArrayPrepData : **string** (Microarray data in which background correction, grid position and log-ratio/log-intensity have been calculated.)

Figure 3 Command line tool window

2.3.3 Tags

The ACGT Workflow editor is a Web2.0 application based on the principles of the semantic web. One of the most informative systems for categorization in the social semantic web is the tagging system. Taking advantage of the tagging system we added this functionality at the workflows repository. Each workflow can be mapped by the user to one or more categories. These categories can be one of the already existing in the system or new ones. In such a way, a user who wants to search for a workflow can narrow the search by adding tags and make the searching of related workflows fast and easy (see Figure 4).

In order to be insensitive to misspellings, plural versus singular words, etc the Double Metaphone phonetic algorithm [16] is used to index the tags and provide approximate matching in tag searches.



Workflows

Name	Last-Modified	Public	Tags
MCMP_PartB_public_files	Τρίτη, 9 Δεκέμβριος 2008	☐	illumina | affymetrix | multi-platform
MCMP_PartB_public_files_2	Παρασκευή, 13 Φεβρουάριος 2009	☑	illumina | affymetrix | multi-platform
simpleMobyGetKEGG	Τετάρτη, 15 Απρίλιος 2009	☐	biomoby

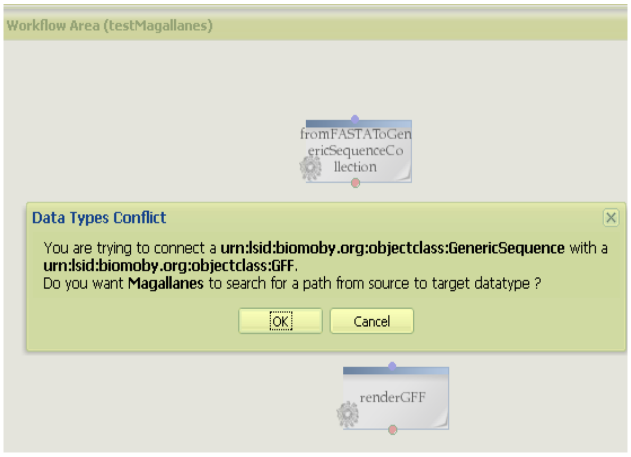
Figure 4 Tag based search for workflows in the Workflow Editor

2.3.4 Magallanes integration

Magallanes (Magellan) is a tool that can be used for discovery of web services and associated data types [18]. Additionally, Magallanes can connect compatible web-services into pipelining workflows that can process data sequentially to reach a desired output.

The new and final version of the workflow editor integrates the automatic service composition functionality of Magallanes. This functionality aims to automate the task of connecting independent services as much as possible. Two services can be connected if the output of one is compatible with the input of the other. The goal is to find an optimal set of services that match inputs with outputs. A detailed description about the technology and architecture of Magallanes can be found in the Deliverable D9.4 "Semantic Integration in ACGT".

Taking advance of Magallanes the workflow editor gained "intelligence". A user has the option to connect objects (services, Biomoby, r-scripts, mediator queries and command line tools) with each other. Sometimes the data types of an input to an output mismatch. In this case the workflow editor will check the data type compatibility and see that the data types are incompatible. When for example a user tries to connect a service B after the service A in a workflow the editor checks if the output of service A is the same or compatible with the input of service B. If the output and input are incompatible then the editor prompts the user that he/she cannot connect these two services. Figure 5 shows the warning message when a data type conflict occurs.



Workflow Area (testMagallanes)
fromFASTAToGenericSequenceCollection
renderGFF

Data Types Conflict

You are trying to connect a **urn:lsid:biomoby.org:objectclass:GenericSequence** with a **urn:lsid:biomoby.org:objectclass:GFF**.
Do you want **Magallanes** to search for a path from source to target datatype ?

Figure 5 Data Types conflict

Then asks the user if he/she wants to invoke Magallanes tool and try to find a path from service A to service B. If the user selects to invoke Magallanes the editor send a request to Magallanes tool with the specific input and output services, using servlets, and a new window will appear with an image of a graph from the selected input to the selected output via various services (Figure 6).

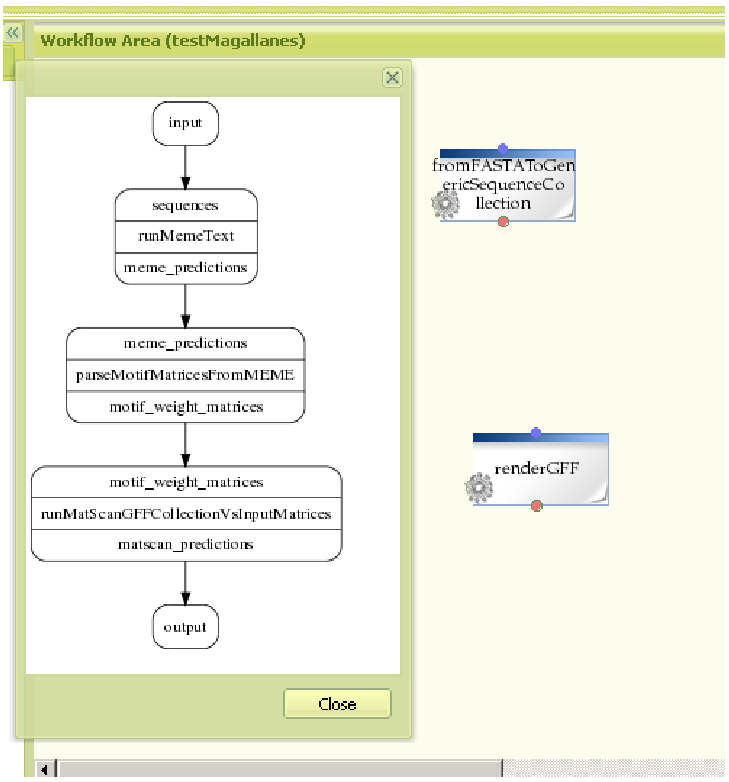


Figure 6 A partial workflow recommendation from Magallanes

2.3.5 New "look and feel"

In the new version of the workflow editor colors have change in order to be in-line with the general look and feel of the ACGT portal. Furthermore the figures of the services have change making the services in a workflow more informative (different figure for different type of service such as Biomoby, r-script, mediator query and generic service). We also have new functionalities in the workflow editor, such as the tagging system of a workflow, Magallanes proposal and command line tools, which is supported by the user interface with icons, windows and figures which are in-line with the general look and feel of the workflow editor. In the following figures you can see the new workflow editor, the command line window and the tagging system of the workflows.

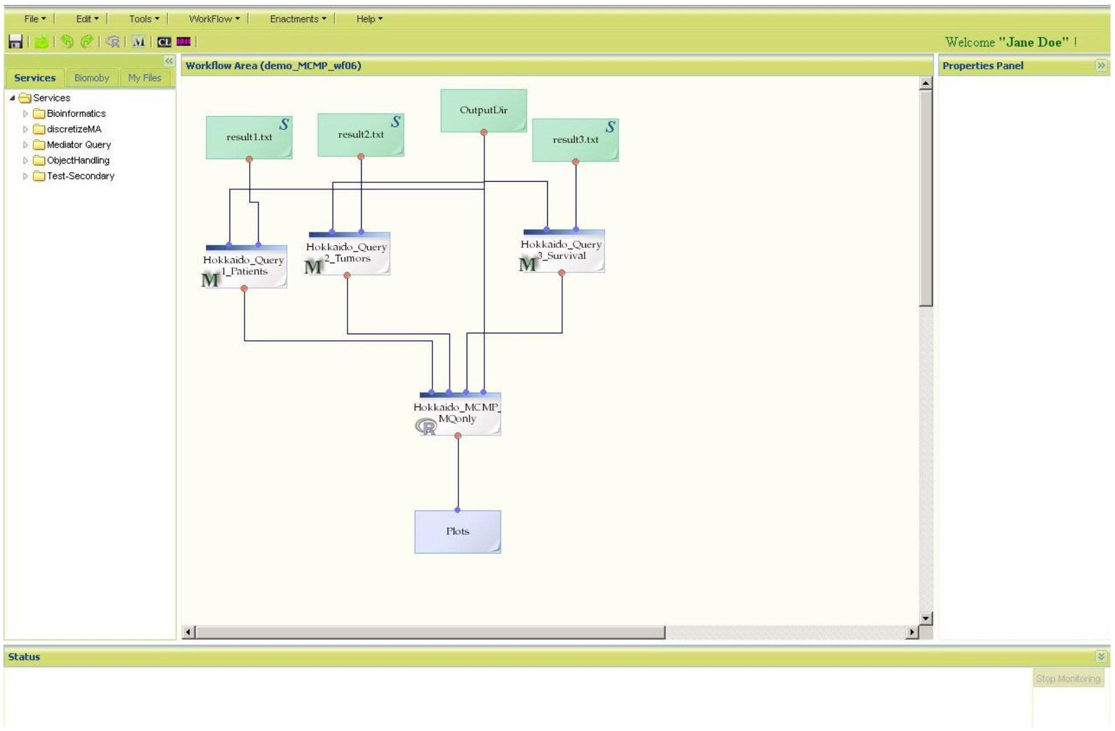
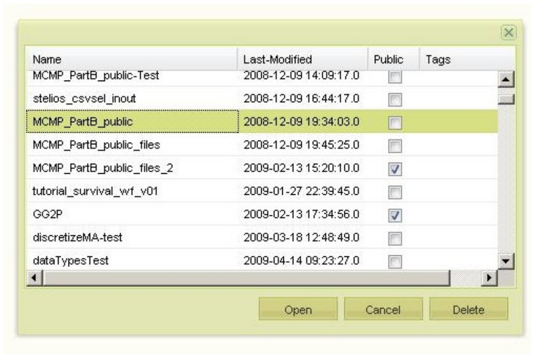


Figure 7 The new workflow editor user interface

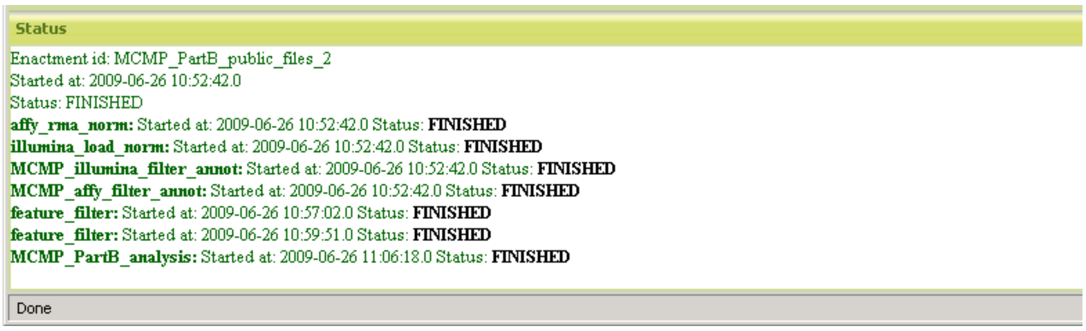


Name	Last-Modified	Public	Tags
MCMP_PartB_public-Test	2008-12-09 14:09:17.0	☐	
stelios_csvsel_inout	2008-12-09 16:44:17.0	☐	
MCMP_PartB_public	2008-12-09 19:34:03.0	☐	
MCMP_PartB_public_files	2008-12-09 19:45:25.0	☐	
MCMP_PartB_public_files_2	2009-02-13 15:20:10.0	✓	
tutorial_survival_wf_v01	2009-01-27 22:39:45.0	☐	
GG2P	2009-02-13 17:34:56.0	✓	
discretizeMA-test	2009-03-18 12:48:49.0	☐	
dataTypesTest	2009-04-14 09:23:27.0	☐	

Figure 8 Tags at the workflow editor

2.3.6 Workflow execution monitoring functionality

The execution of a workflow is monitored in the “Status” panel at the bottom of the workflow editor (Figure 9). Status panel contains information about the workflow status (e.g. response from enactor, errors during deploy/run etc) and information about the execution (start-up time, status and time of all the services in the workflow).


Status

Enactment id: MCMP_PartB_public_files_2
Started at: 2009-06-26 10:52:42.0
Status: FINISHED
affy_rma_norm: Started at: 2009-06-26 10:52:42.0 Status: **FINISHED**
illumina_load_norm: Started at: 2009-06-26 10:52:42.0 Status: **FINISHED**
MCMP_illumina_filter_annot: Started at: 2009-06-26 10:52:42.0 Status: **FINISHED**
MCMP_affy_filter_annot: Started at: 2009-06-26 10:52:42.0 Status: **FINISHED**
feature_filter: Started at: 2009-06-26 10:57:02.0 Status: **FINISHED**
feature_filter: Started at: 2009-06-26 10:59:51.0 Status: **FINISHED**
MCMP_PartB_analysis: Started at: 2009-06-26 11:06:18.0 Status: **FINISHED**

Done

Figure 9 Monitoring status panel

Monitoring starts automatically when an execution of a workflow starts. In addition active workflow elements (services, r-scripts, mediator queries, etc) can be seen blinking in the workflow area. Once the execution of a workflow has started, it is running as a background process. The user can leave the session and the process continues. The status of a specific workflow which is running or has ended or has failed can be retrieved/watched by selecting the corresponding execution "process" from the list that pops-up after selecting "Open…" in the "Enactments" menu. When an execution is selected the workflow is automatically loaded and the monitoring process starts.

2.4 Proxy services infrastructure

In designing and building the ACGT workflow environment the integration of the Grid security to the workflow enactor proved to be the most challenging task. In particular, the workflow engine should support the delegation of user rights so that all the services that participate in a workflow are contacted by the enactor in the name of the end user with no need for the user to be present during the workflow execution. What is more, there are numerous existing third party services which it would be nice to be integrated in scientific workflows and which do not support these security standards and it is neither possible nor desirable that these services be re-implemented. Thus, what is needed is a design approach which permits the mixture of heterogeneous systems. Taking into consideration that this mechanism must conform also to legal requirements, it becomes obvious why it is challenging to comply with the above and at the same time provide a functional, efficient and non-restrictive platform.

There's an old adage saying that "Every problem in computer science can be solved by another layer of indirection". The incompatibilities between the BPEL processes and the GSI secured ACGT services can be overcome by supplying the necessary layer of indirection: the Proxy Services. The idea is nothing novel: the "Proxy design pattern" allows the provision of "a surrogate or placeholder for another object to control access to it" [8]. In our case the proxies or wrapper services provide BPEL friendly facades of the original, "real" ACGT services, effectively working as calls transformation bridges between the two worlds.

The core idea behind wrapper services is that we transform the interface of the underlying service and we pass extra pieces of information, which empowers us to detour the usual flow of credential delegation and bypass the enactor. This information is an ID, unique for each enactment, with which the proxy service can retrieve the delegated credentials and pass them on to the service which is "substituted" (see Figure 1). The BPEL workflow therefore is constructed in such a way as to pass this extra parameter in every outgoing request. The value of this extra parameter enters the BPEL engine through the Enactor Proxy: this is an inverse proxy service that presents a GSI compliant interface of the BPEL workflow. The Enactor Proxy saves the security context of the workflow enactment in the database and submits the corresponding database ID to the BPEL engine with the other parameters. The BPEL engine accepts these parameters and forwards the ID to the Proxy Services, which, based on this ID, subsequently retrieve the security context and make the GSI compliant request to the real service.

The proxy services mechanism can be also used to provide higher level abstractions. An example of this is to remodel the interface of the service in order to simplify it or provide through one ``gateway'' adapter service access to many services by dynamic creation of custom interfaces. In the case of the “GridR Proxy Service”, by using the wrapping technique we were able to encapsulate the underlying “GridR Service” [22] and hide its technical details, but also to support the notion of ``Scripts as Services'', in which a different web service interface is exposed from the same single proxy service, depending on the actual R-script that is being proxied. All the inputs and outputs of the R script are presented as input and return parameters of the corresponding web service, through a dynamically created script-specific WSDL interface.

In a similar manner the wrapping technique has been applied to provide secure access to WSRF OGSA-DAI resources [1] and simplify the connection to them by enclosing and hiding the technical hindrances of OGSA-DAI, like perform documents and stateful resources. Instead of writing and applying “perform documents” to OGSA-DAI resources, users can view the data resources as web services and each OGSA-DAI data resource as a different web service. Furthermore, the “Proxy services” design strategy is flexible enough to overcome many technical obstacles, like long running executions which are transformed into asynchronous executions with “polling” interfaces, transparent and non-intrusive addition of logging mechanisms, and so forth.

2.4.1 Generic Guidelines for the Implementation

The implementation of Proxy services depends a lot on the kind of the original service that will be "proxied". From the above discussion we see two major uses of proxies:

- Implement the security delegation mechanism in an enactor transparent way. Since this type of proxies provide the same functionality as the original services but they also augment ("decorate") it with the proper security context we call them Decorators[4].
- Provide either a high level view of the original service functionality (e.g. as is the case with OGSA-DAI services where we don't want the user to be aware of low level details like "perform documents") or an enactor friendly interface (i.e. a WS-I compliant web service interface). Let's call this specific type of proxies as Adapters[5].

In either case the proxy service presents a modified WSDL interface to the enactor but the details differ. The main difference is that in the case of credential delegation every operation supported by the Proxy service must require an additional "`enactmentId`" parameter of type (XML Schema) String[6]. This parameter should have this exact name and type and should be the first in the list of parameters of every operation of the Proxy service interface.

A Proxy service could be very well both a Decorator and an Adapter. In fact this is the case for the OGSA-DAI Data Access (Proxy) Services because they require the user credentials to perform the delegation (and therefore the "`enactmentId`" parameter) and they are also Adapters since their WSDL is a lot different than the OGSA-DAI one.

The final thing to keep in mind is that Proxy services are contacted by the BPEL enactor so their interface needs to comply with its requirements. In designing the web service interface

[4] http://en.wikipedia.org/wiki/Decorator_pattern

[5] http://en.wikipedia.org/wiki/Adapter_pattern

[6] http://www.w3.org/TR/xmlschema-2/#string

they are apparently many different WSDL styles[7]. Nevertheless it seems that BPEL and the Apache ODE Workflow Enactor specifically do not support Web Services implemented in accordance to the RPC/Encoded style. The WS-I consortium also recommends against the use of SOAP Encoding rules. The services implemented following the Document/Literal style[8] in WSDL are the ones that are fully compliant with BPEL and ODE. Furthermore, the ACGT tool for the BPEL transformation works with the WDSL version 1.1 and its SOAP 1.1 binding. So in conclusion the requirements for the Proxy service interface description are the following:

- Use the WSDL version 1.1 web service description language
- Use (and specify in the WSDL document) the SOAP version 1.1[9] protocol
- Use (and specify in the WSDL document) Document/Literal wrapped style

Getting access to the proxy certificate

In cases where the original service that is "proxied" is a GSI enabled Grid service, the delegated proxy certificate can be retrieved using the "enactmemtId" parameter. In order to make this easy and transparent to the proxy service developers a "' proxyAuth.jar"' jar library is provided. The ProxyAuth class of this jar offers a "getCertificate()" method that given the enactmentId returns the [GSSCredential] proxy certificate.

Having the credential the proxy service code should use it and proceed to the actual call to the "original" service as if it were the real client.

An (easy) Example: Data Access Services

The original ACGT Data Access Service (DAS) presents an OGSA-DAI[10] compliant interface. The target (original) OGSA-DAI service requires valid ACGT user credentials when contacted and therefore the corresponding proxy services would be a "Decorator" as described above and all its operations will have an "enactmentId" string parameter. Furthermore we aim to provide a simple interface to the user based on the core functionality of the service, which is "data access" in relational (or Semantic Web) data. Therefore we design the Proxy interface to offer the following operations:

- `submitSQL (enactmentId: String, DataSet: String, SQL_query: String, DirName: String, FileName: String): String` This is an operation to send an SQL query. "DataSet" is the (OGSA-DAI) resource id of the database, "DirName" and "FileName" designate the names of the folder and the file in DMS where the results will be stored. The return value is a "job id" that references this specific request (query) so that the client (i.e. the enactor) can use in order to find the status thereof.
- `submitSPARQL (enactmentId: String, DataSet: String, SPARQL_query: String, DirName: String, FileName: String): String` The same as above but for SPARQL queries for Semantic Web databases.
- `hasFinished(enactmentId: String, jobid: String): boolean` This operation returns the status (finished or not) of the query identified by the "jobid"

[7] http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[8] http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/#N1019C

[9] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[10] http://sourceforge.net/apps/trac/ogsa-dai/wiki/UserDocumentation

- `getResults(enactmentId: String, jobid: String): int` This returns the DMS id of the file that persists the results of the query (if finished).

The resulted WSDL is at https://iapetus.ics.forth.gr/proxies/DASProxyService.wsdl

2.4.2 A Case Study: Interoperability with the Biomoby Services

The BioMOBY project [24] offers an impressive number of bioinformatics tools and services. It builds upon an registry where the Biomoby compliant tools are registered and discovered by interesting clients based on their metadata and semantic based descriptions. In particular, as described in Deliverable 9.4, the Biomoby framework is based on a set of end-user-extensible ontologies as its framework to describe data semantics, data structure, and classes of bioinformatics services. These ontologies are shared through a Web Service registry system, MOBY Central, which uses the ontologies to semantically bind incoming service requests to service providers capable of executing them [25].

In order to foster interoperability the Biomoby tools should be created according to the following guidelines:

- Use of the Biomoby object ontology (Figure 10) to define the inputs and outputs. The Biomoby object ontology is a graph where the nodes represent object classes and supports two types of relationships: ISA, representing subclasses and HAS-A/HAS representing containment. The root (the more general) data type is the Object type and every other class is a direct or indirect subclass of it.
- Use of the Biomoby service classification ontology to define functionality and tools capabilities. This is a simple subclass hierarchy which defines a (generic) set of data manipulation and bioinformatics analysis types such as “Retrieval”, for retrieval of records from a database, “Parsing”, for the extraction of information from various flat-file formats, “Conversion”, for data-type syntax changes, etc.
- Implement the tool programmatic interface according to Biomoby web access protocol and serialization formats. In particular the Biomoby tools use a SOAP based (i.e. Web Service) communication protocol that is described below.
- Register to the Moby Central registry(-ies) with the appropriate tool description so that categorization and semantics-based discovery of the tool is possible.

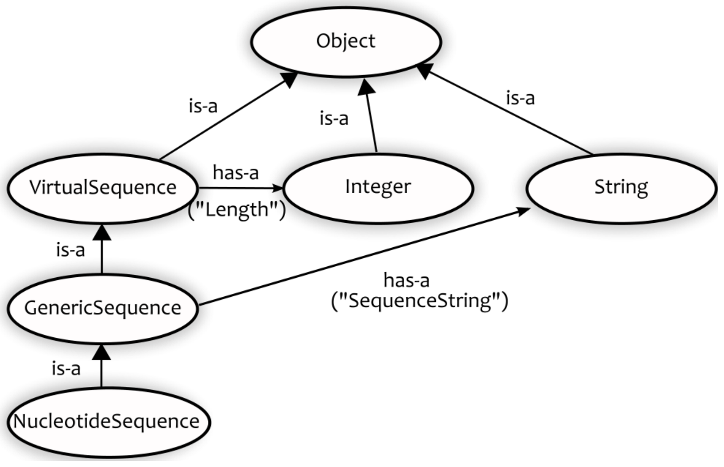


Figure 10 A small part of the BioMOBY object ontology.

Therefore in order for the ACGT platform to gain access to the Biomoby based bioinformatics tools a "gateway" infrastructure needs to be in place that give access to the MOBY central registry and "understands" the Biomoby specific protocols and conventions in order for the Biomoby services to be invoked and accessed from inside the ACGT knowledge discovery platform. The details of the implementation of this gateway are given in the following sub sections.

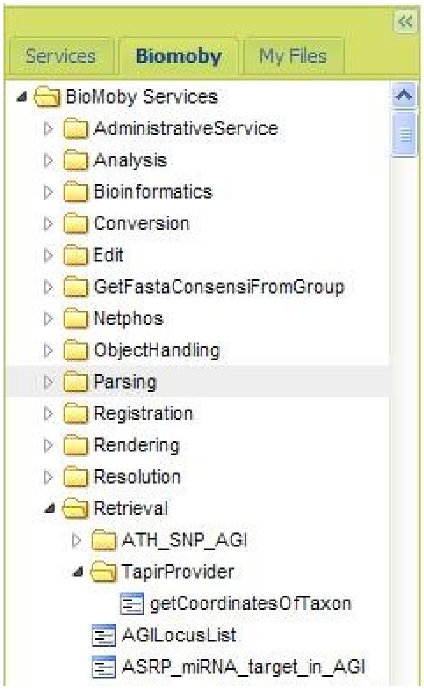


Figure 11 Part of the BioMOBY Service Ontology as shown in the ACGT Workflow Editor

MOBY Central integration

In order to offer an efficient integration mechanism the ACGT-Biomoby gateway uses a local repository of the contents of the MOBY central. The rational is that the contents of the Biomoby registries do not change very often and having a local "cache" of the registry will provide additional opportunities for making optimizations. In general there are a few Biomoby registries available worldwide with different contents but currently the most comprehensive one seems to be that at the University of Calgary, Canada (http://moby.ucalgary.ca/). The contents of this registry were retrieved in RDF format and stored in a local RDF (Sesame[11]-based) repository. This local repository is also augmented by the contents of the Biomoby object, classification (service), and data types (Namespace) ontologies that were also downloaded from the Calgary registry. The relevant web addresses for directly accessing these ontologies are shown in the table below.

Object ontology	http://moby.ucalgary.ca/RESOURCES/MOBY-S/Objects
Service Ontology	http://moby.ucalgary.ca/RESOURCES/MOBY-S/Services
Namespace Ontology	http://moby.ucalgary.ca/RESOURCES/MOBY-S/Namespaces
Services (instances)	http://moby.ucalgary.ca/RESOURCES/MOBY-S/ServiceInstances

Table 1 Web addresses for accessing the BioMOBY service registry in Canada

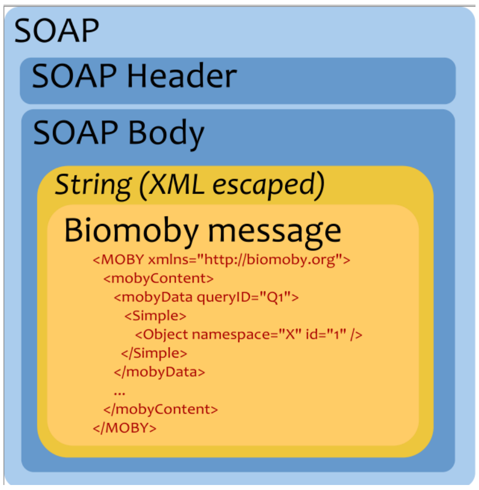
[11] http://www.openrdf.org/

It's evident that the content of this local repository is not fully synchronized with that of the original MOBY Central. It can be though periodically updated by downloading the content of the MOBY Central in an offline mode and rebuilding the local database.

Invocation of the Biomoby tools

The ACGT-Biomoby gateway is a software component that is also in charge of making the actual communication with the Biomoby tools, i.e. for submitting requests to them and retrieving the results from these invocations. The need for such a "middleman" in the invocation of the Biomoby services is imposed by the Biomoby specific messaging principles.

Normally after retrieving a tool description from the MOBY Central, clients interact with the actual tool by sending a request in the form of an *input* message, and then receiving the response in an *output* message. This request response communication is based on Web Services technologies [5], i.e. the messages are serialized in XML and the communication protocol is SOAP. But SOAP and XML define the envelope of the input and output messages: the "payload" of the message is formatted in compliance with the Biomoby conventions. These conventions explicitly define the syntax of the input messages and the output messages in a way that is strongly connected with the data types of the input parameters and the results as defined in the Biomoby Object ontology.



```
<MOBY xmlns="http://biomoby.org">
  <mobyContent>
    <mobyData queryID="Q1">
      <Simple>
        <Object namespace="X" id="1" />
      </Simple>
    </mobyData>
    ...
  </mobyContent>
</MOBY>
```

Figure 12 The "Russian doll"-type format of the Biomoby messages

A high level view of a Biomoby message "on the wire" can be seen in Figure 12. The SOAP message contains, in its body, a string-encoded Biomoby message. In general this Biomoby message consists of (Figure 13):

- An outermost XML tag "MOBY", identifying the message as a Biomoby formatted message.
- A child XML element "mobyContent" within which formatted service provision information can be placed such as database version, software name and version, etc. In addition, this element is the container for any error reporting and exception information that the service provider wishes to pass back to the client.
- One or more "mobyData" XML elements within "mobyContent", which contain:
- Zero or more instances of a BioMoby Object being passed into or out of the service. These are the input parameters submitted in an input message or the results in an output message.

```
<MOBY xmlns="http://biomoby.org">
  <ProvisionInformation>
    ...
   </ProvisionInformation>
   <mobyContent>
    <mobyData queryID="Q1">
      <Simple>
        <Object namespace="X" id="1" />
     </Simple>
    </mobyData>
    <mobyData queryID="Q2">
       <Simple>
         <Object namespace="X" id="2" />
       </Simple>
    </mobyData>
  </mobyContent>
</MOBY>
```

Figure 13 The serialization of the Biomoby input message

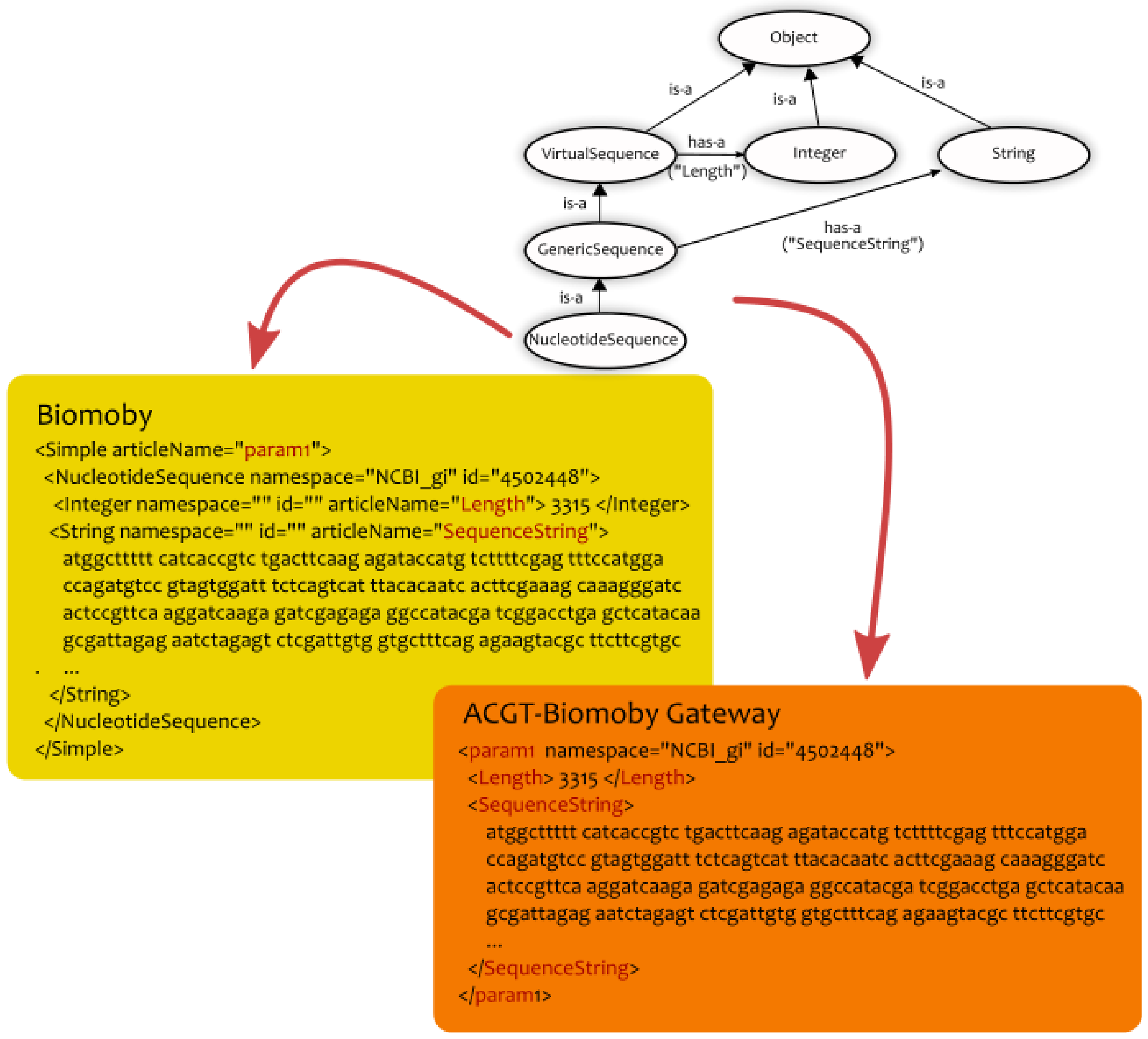
Based on the information above, we have identified a number of problems with the Biomoby messaging formats and protocols when used in a state-of-the-art service oriented software architecture:

- The whole message as shown in Figure 12 presents an "onion" layered mix of encodings. The Biomoby specific message with its parameters etc. is encoded as a String that is further encapsulated into a SOAP call.
- Some of the existing functionality of SOAP and web services is duplicated. For example, errors are sent inside the Biomoby message instead of declaring the possible error conditions in the WSDL and using the standard fault mechanisms of SOAP.
- Interaction with the Biomoby services requires knowledge of these serialization conventions and the Biomoby specific ontologies so no generic Web Service clients can be used.
- The web service description (WSDL) of a Biomoby service does not provide enough type information for making possible the interoperability with modern Web Service platforms. In particular the WSDLs of all Biomoby tools declare operations that accept a single string input but of course this cannot be any string: it should be a Biomoby message encoded as a string.

In conclusion, the main problem introduced by the Biomoby API in a Service Oriented Architecture such as the ACGT platform is that the machine readable descriptions of the Biomoby services are not type safe and compliant with the existing generic Web Services tools and infrastructures. The objective therefore is to build an ACGT-Biomoby gateway that presents a standard web service interface to the rest of the ACGT platform. This gateway can be used as a Biomoby "Proxy" service (see the "Proxy services infrastructure" section above) that, although a single instance, through dynamic inspection of the messages can contact all the existing Biomoby compliant service and be a "proxy" for them. In order to do so all the necessary information about the parameters and data types of each Biomoby service are explicitly defined in dynamically created WSDL documents. Therefore the mapping to WSDL that is performed by the gateway includes the following

- Each parameter of the Biomoby service, be it “primary” or “secondary”, is explicitly defined as an input parameter of the SOAP call. This is also the case for the results of the invocation.
- Each input and output data type is fully described in an XML schema that is included in the WSDL. Biomoby defines how each type (node) in the Object ontology should be serialized in a transitive way based on its super types and the “contains” relationships that itself or its super types may have. The XML schema produced is also compliant to the semantics of the Object ontology (i.e. there’s mo loss of information) but the actual syntax slightly differs. Each Biomoby object type is represented by an XML Schema *elementType* while the containment properties (e.g. “SequenceString” in the example of Figure 10) are serialized as XML elements. This arrangement makes possible to use a more specific data type in place of one of its super types because “content-wise” they have the same structure (with the possibility of the more specific type to have more properties, but not less). Figure 14 shows an example of the differences between the two syntaxes i.e. the Biomoby original data type serialization compared to the one of the ACGT-Biomoby gateway. Both represent the XML syntax for a parameter called “param1” of the NucleotideSequence type. It is worth mentioning that in the case of the Gateway the same XML fragment can be used for both the NucleotideSequence and its super type GenericSequence because they have the same “Length” and “SequenceString” properties.
- Due to the way the mapping of the Object ontology to the XML Schema is performed the explicit semantic information is lost. For example the Gateway’s XML fragment in Figure 14 there’s no indication that the “param1” object is a NucleotideSequence. Nonetheless, this semantic information still exists in the Gateway’s WSDL. Every input parameter and result is annotated with its Biomoby Object type through the means of Semantic Annotations for WSDL and XML Schema (SAWSDL [20]).

Following the above design decisions the ACGT-Biomoby Gateway presents a Biomoby service-specific WSDL and makes the appropriate transformations and mappings between the two messaging paradigms. For the rest of the ACGT platform each Biomoby service appears to be a standard Web Service that is also semantically described using the SAWSDL annotations inside the WSDLs. On the other hand, in the Biomoby world, the Gateway presents itself a Biomoby compliant client application.



```
<Simple articleName="param1">
 <NucleotideSequence namespace="NCBI_gi" id="4502448">
  <Integer namespace="" id="" articleName="Length"> 3315 </Integer>
  <String namespace="" id="" articleName="SequenceString">
    atggcttttt catcaccgtc tgacttcaag agataccatg tcttttcgag tttccatgga
    ccagatgtcc gtagtggatt tctcagtcat ttacacaatc acttcgaaag caaagggatc
    actccgttca aggatcaaga gatcgagaga ggccatacga tcggacctga gctcatacaa
    gcgattagag aatctagagt ctcgattgtg gtgctttcag agaagtacgc ttcttcgtgc
.   ...
  </String>
 </NucleotideSequence>
</Simple>
```

```
<param1 namespace="NCBI_gi" id="4502448">
 <Length> 3315 </Length>
 <SequenceString>
   atggcttttt catcaccgtc tgacttcaag agataccatg tcttttcgag tttccatgga
   ccagatgtcc gtagtggatt tctcagtcat ttacacaatc acttcgaaag caaagggatc
   actccgttca aggatcaaga gatcgagaga ggccatacga tcggacctga gctcatacaa
   gcgattagag aatctagagt ctcgattgtg gtgctttcag agaagtacgc ttcttcgtgc
   ...
 </SequenceString>
</param1>
```

Figure 14 Data type serialization in Biomoby and in the ACGT-Biomoby gateway

2.5 BPEL Transformation

In order for the data flows created by the workflow editor to be "enacted" a transformation to an executable language should be performed. In our case the BPEL workflow enactor is actually used as a "virtual machine" for the execution of the high level scientific processes designed in the workflow editor but a transformation step to BPEL is required.

It should be noted that this transformation is needed because the Workflow Editor does not use the BPEL as its native workflow description language. This can be the case in some future version of the editor but so far the workflows are persisted in a custom format. The reason for this is mainly simplicity: we don't explicitly represent all the BPEL structures in the editor either ways. Instead, additional information about the positioning of the services in the graph (i.e. the coordinates for the visualization of the workflow) needs to be stored, and this information has no semantic effect, e.g. to the actual execution of the workflow.

Although the internal format used by the workflow editor for storing the workflow descriptions is XML-based, the transformation to BPEL presents a lot of challenges. Some of the reasons for this are the following:

- In the workflow editor a simplified view of the workflow activities is shown. This is required in order to facilitate the end users when they design their workflows by presenting a more coarse-grained and abstract view of the services and activities. In particular details of the service interface such as complex message exchange patterns as in the case of “polling” (asynchronous) operations or the passing of the “enactment id” to the proxy services are hidden. Instead, each node in the process graphs of the workflow editor represent the high level functionality the users are familiar with, irrespective of the low level interface, security, and other characteristics of the services.

- BPEL is mostly an imperative language whereas the process view in the workflow editor uses by and large a declarative notation. More specifically, the data flows of the workflow editor need to be mapped to assignments of variables in the BPEL descriptions. Furthermore the BPEL variables need to be correctly initialized and XML specificities like namespace handling increase the complexity of the mapping.
- The BPEL specification mandates a lot of additional information to be specified like the proper definition of partners and partner roles, the specification of XPATH expressions and XSLT mappings for complex handling of XML messages, etc. The XML basis of BPEL and its focus on the more coarse grained business requirements adds up verbosity and makes “programming in the small” [6] more difficult. The extension mechanisms of BPEL can of course ameliorate these “shortcomings” at the cost of making the produced process descriptions more coupled to a specific BPEL engine implementation.

The mapping to BPEL and the creation of the BPEL descriptions of our workflow editor's data flows are performed by the BPEL transformation tool which is a subcomponent of the workflow editor's back end. This tool is accessible over the network by the editor's client side and accepts the editor's workflow XML definition and produces a zipped package of all the necessary files for the process deployment in the Apache ODE[12], which is the third party BPEL enactor used in ACGT.

For each Proxy service there's a unique identification and the configuration of the BPEL mapping tool includes all the necessary information. In particular each proxy service is described by its network end point, whether it requires user proxy (GSI) credentials, etc. Each service, be it Proxy or not, is a “partner” in the BPEL parlance and its WSDL is either specified in the input workflow definition of the editor or in the case the Proxy services in the mapping tool's configuration. The BPEL transformation tool retrieves these WSDL descriptions, slightly modifies them to include the partner role(s) specifications, and stores them temporarily for inclusion in the output zip archive in the last step of the transformation process.

The transformation tool leverages the “flow” construct of the BPEL to map the graph representation of the editor's workflows to an identical graph in the BPEL description. The use of the “flow” element has the additional benefit that at runtime, when the final BPEL process is deployed and enacted, independent activities are executed in parallel. The data flows between the execution steps in the input description represent (data) dependencies and therefore the corresponding “source” and “target” links are introduced in the BPEL output. Since BPEL is more control-flow oriented, certain *variables* are introduced to represent the exchanged messages and the data connections between the activities in the input description are modeled as the assignments of “output” to “input” BPEL variables.

The whole BPEL process is a new web service and therefore a WSDL is also produced with the appropriate process and client partner roles. The interface of this process/web service is a simple one, having just one asynchronous operation called “enact”. This operation accepts the necessary input for the initiation of the underlying process and it's asynchronous because the process could take a lot of time to finish. In the BPEL process description the very first activity is a “receive” of the input message of the “enact” operation and the corresponding

[12] http://ode.apache.org

initialization of the process' variables. By symmetry, the final activity is the return of the process results to the initial requester in a "callback" message.

Finally the new BPEL process is deployed in the Apache ODE and made available as a new web service.

3 Metadata for the ACGT data, tools, services and workflows

Metadata is the crucial component of every large and complex system that abstracts and captures the essential information of the underlying data independent of their representational details. In ACGT the use of metadata is pervasive: from digital artifacts (data) that are produced or consumed by tools, to the tools and services themselves, and the high level processes (workflows) expressing the orchestrated execution of a series of processing tasks.

Metadata are a special type of data that comprises the information that is pertinent to an entity without being the entity's "content" per se. Metadata can vary from simple *descriptive* annotations, such as the creator of a file or its last modification time, that provide more context and explanation about the entity described, to more high level information used for the *interpretation* of the data, for example the format used, the purpose of the data, or in abstract terms its *meaning*. In ACGT we make use of metadata for various artifacts:

- Data sets stored as files and databases
- Services and tools invoked over the network
- Workflows
- Results of workflows and services invocations

Some of these metadata are provided by the users, such as descriptions for their workflows, while the majority of the metadata are produced by the system itself or entered by the administrators and the services implementers.

So for the administration and manipulation of the metadata we have defined the following components in the ACGT architecture:

- Metadata Repository and its access services, which is currently used for storing metadata about services, workflows, R scripts, mediator queries, and command line tools.
- Grid Data Management services, used for the management of users' data files.
- Workflow Repository, in the premises of the ACGT Workflow Environment, with enactment and provenance information

So there's a clear separation about the storage of metadata:

- For the file related metadata the DMS is to be used. This means that data uploaded by the users or produced by the enactments of their workflows are annotated with metadata which are persisted by the DMS
- For the "active" content, i.e. services and workflows, the Metadata Repository is used both as a registry and a "yellow page" directory. The service related metadata therefore are stored in the Metadata Repository.
- Provenance information i.e. what was produced by whom and from what input is stored in the Workflow repository.

This separation of the service, data, and provenance related metadata was decided because the data which the users are in control for are stored in the DMS anyways and therefore it's better for consistency reasons to have a single database to store both the data and their metadata. Finally the workflow repository is somewhat in the middle of the two other repositories in the sense that it relates the service and workflow static information, which persists in the metadata repository, with their results, which are stored in the Grid data management facilities, through the means of the enactment information, which is dynamic in nature, that is maintained in the workflow repository. Linking and correlation between these repositories is achieved by the means of the unique identifiers supported by each repository.

3.1 Metadata for Data

File related metadata is the key component to integrate the data in ACGT. In the ACGT environment, file related metadata consists basically of the following attributes describing the file in different ways:

- Identification. Each of the data file stored in DMS is identified by a unique number.
- MIME type. This provides the generic file format, e.g. zip file, text, etc [8].
- Datatype, which gives the semantic type of the content of the file. The value range of this element is the datatype ontology managed in the metadata repository (see paragraph 3.2.3 on page 34).
- Owner – this attribute contains security-related information, e.g. information on the owner of the file., i.e. its X.509 distinguished name (DN) such as "`/C=EU/O=ACGT/OU=DemoOrganisation one/CN=Jane Doe`"
- Description – this attribute contains a textual description of the data
- Provenance – this attribute contains information on how the file was generated, e.g. by which workflow or service.

3.2 Metadata for Tools and Services

A well-defined metadata schema is essential to enable tool providers to publish metadata descriptions for their tools and for clients to find (discover) tool based on metadata. The scope of the ACGT metadata repository is tools and their operations (specific functions of a tool), data-types and functional categories. In the following subsections, only the main concepts of the schema are described as this was previously described in D6.3.

3.2.1 Tools

A Tool represents a grouping of software components which can be used to solve a specific problem. A Tool acts as a container of operations with closely related functionality. An operation is a software sub-component that solves a specific task and could have several parameters; either input or output as well as for service customization (optional input parameters). Each I/O parameter is associated with a specific data type.

The metadata for tools include the author and authority (organization of the author). Each tool and operation can be associated with human-readable descriptions (long and short versions). The short description is intended for quick browsing and the longer version is intended for users that wish to more carefully study tool/operation documentation.

The tool metadata also specifies the underlying protocol. Currently several types of tools are supported: traditional SOAP services [5], BioMOBY [25], BPEL workflows and secure ACGT services. Currently, there are at least two special types of such secure services; based on R-scripts [22] and command lines [10].

SOAP-based web-services can include their WSDL descriptions when publishing. The WSDL contains specifications about data format and information needed for the client to bind to the service (such as the protocol specifics and endpoint). For BioMOBY web-services, WSDL is not used to specify the format of the data. In this case, the data type metadata is used to infer the exact format according to the BioMOBY specification.

Workflows are viewed as abstract tools ("black boxes") which require inputs and produce outputs. Potential workflow metadata is an image representing the workflow and definition (in BPEL format).

For ACGT services based on command line execution, the schema includes the command to execute (path). R-based ACGT services include the R-script which is retrieved by the service and executed on the GRID.

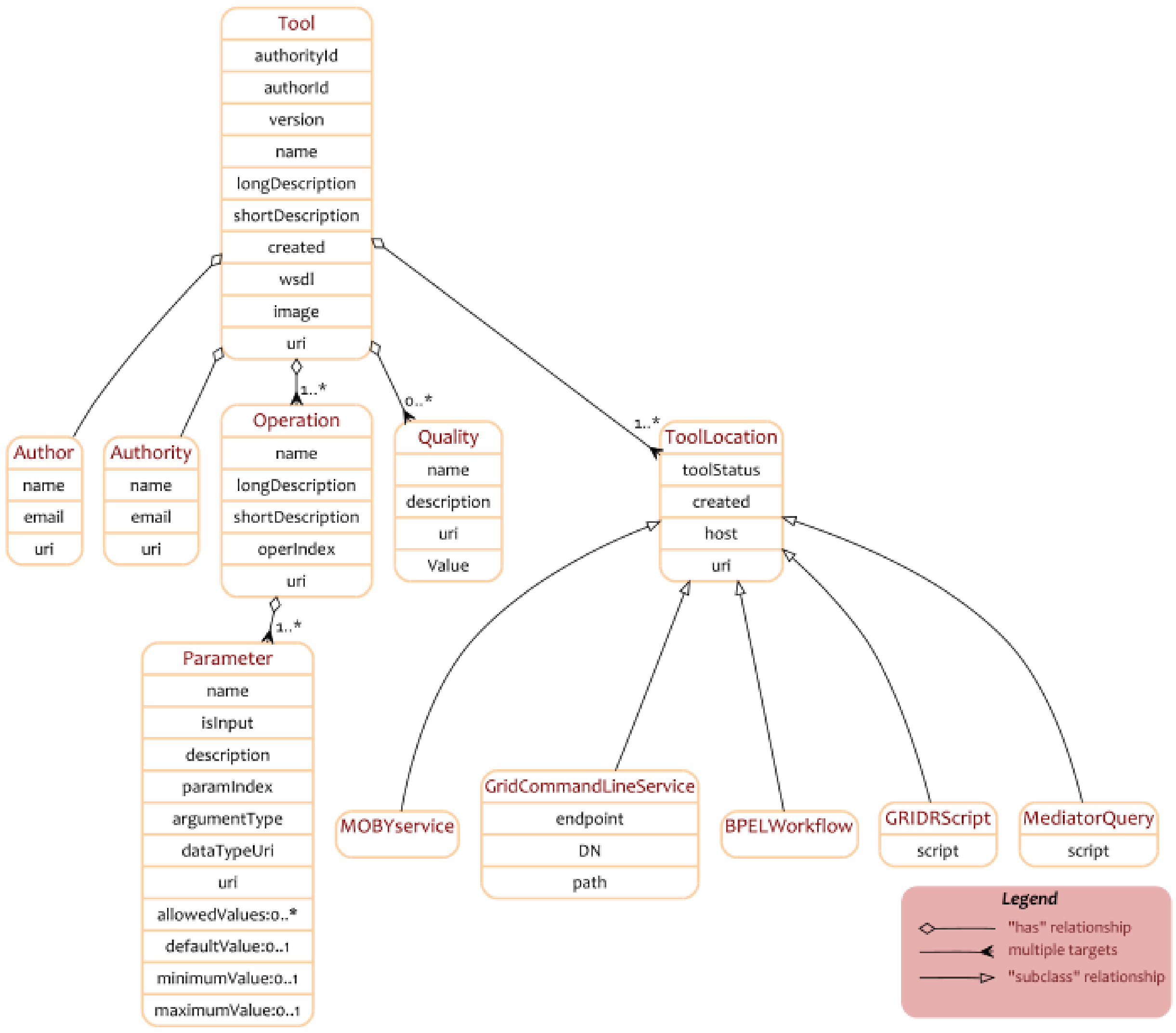


Figure 15 The Tool related metadata schema

3.2.2 Data types

The data type metadata defines a shared taxonomy of data types. This enables tool composition (combination). The taxonomy follows the object oriented paradigm where data types are related to other data types. Data types can inherit parts from another data type and add additional structure. They may include (contain) or be arrays of other data types. The interpretation of such relations between data types is domain specific for the service type. For example, BioMOBY web services would interpret these relations as directly specifying the data format. For generic SOAP services, these tables would only be used to specify a hierarchy of data types without any assumptions of the data formats (which are specified in the WSDL descriptions).

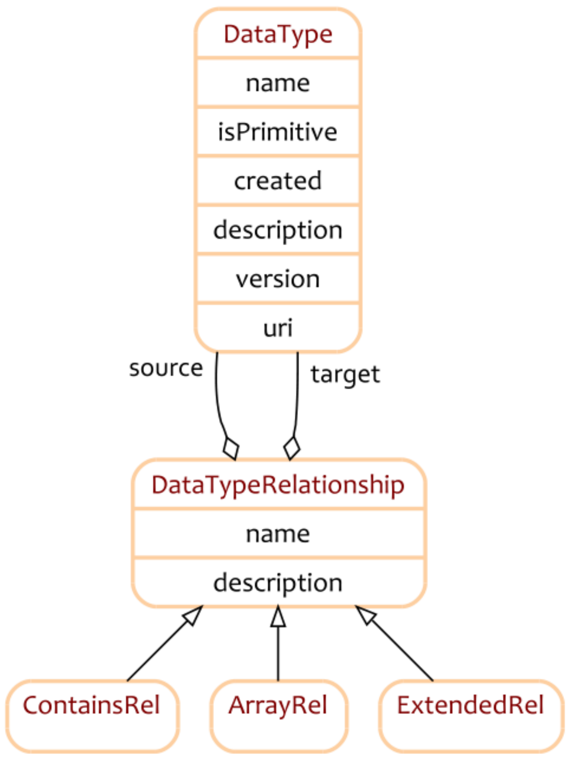


Figure 16 The Data Type related metadata schema

3.2.3 Datatype ontology

Rather than starting from scratch, ACGT has adopted the datatype ontology from the Spanish National Institute for Bioinformatics (based on BioMOBY) where many important datatypes in bioinformatics are already included, such as sequences and phylogenic trees. In some cases, it has been necessary to extend the BioMOBY datatype ontology with some ACGT specific additions:

- DMSFileID – this datatype represents identifiers of files in the DMS data repository [17].
- MicroarrayDerivedData: data-type that uniforms all gene expression raw data formats (Genepix, arraypro, etc.) keeping the following information for a raw o pre-processed gene (when available). This datatype is used for CLP services [6] based on PreP+07 [11]. Data of this datatype contains the following attributes:
    - Control Signal
    - Control Background
    - Target Signal
    - Target Background
    - LogRatio
    - LogIntensity
    - X position
    - Y Position
    - Block (Grid)
    - BlockX

  - BlockY

- GeneExpressionMatrix: File contains gene expression data derived from a collection of MicroarrayDerivedData files or cell files. Information is represented as a matrix, where each row corresponds to gene info (in different experiments) and each column represents values from each experiment. Additionally, we can add metadata per gene (row) or per experiment (column).
- Cluster: File contains a data (vectors) classification for matrix gene expression. It does not contain the original data, but their references.

3.2.4 Functional descriptions

Tools can be associated with one or more functional category. A functional category is a keyword that describes the general function of the tool. The functional categories can be related to other functional categories to create a taxonomy of keywords. Since the keywords are arranged in a hierarchical structure, this makes it possible for clients to discover tools that are annotated with a more generic functional keyword and all inheriting keywords. For example, if the functional category taxonomy consists of the keyword "clustering" and two sub-keywords inheriting from clustering "hierarchical clustering" and "k-means clustering", searching for a tool with annotation "clustering" would return also tools that are annotated with "hierarchical clustering" and "k-means clustering".

Functional category ontology

We have adopted the service type ontology from the Spanish National Institute for Bioinformatics with some ACGT specific expansions related to Gene Expression (GE) and Microarray data:

- GE Data Management – Procedures related to transformations of GE data (matrixes, reductions, transposing) and data management (uploading, downloading, etc.)
- Microarray PreProcessing
  - GE Matrix Preprocessing – Procedures for adjusting, filtering, filling, transposing and transforming original data sets and preparing them for clustering procedures
  - GE Data Normalization – One notable procedure in this functional category is Quantile based normalization which is well adapted to process large gene expression datasets, and has been implemented to run in multiprocessors. Other simpler methods can be also applied, like logarithms, standard deviations, etc.
  - GE Plots – Procedures for plots (intensity-intensity, density etc).
- Microarray PostProcessing
  - Clustering – Procedures for clustering (hierarchical, k-means, fuzzy methods, etc.)
  - Projection Methods – Procedures for projection (PCA and non-linear techniques like Sammon mapping and Self-Organizing Maps)
  - Statistical Analysis – Currently containing two methods to create histograms with the data distance distribution.

Service discovery

The service discovery in ACGT is based on Magallanes [18]. It permits users to search for data types and services using a scoring system based on the number of occurrences and relative word positions of matching hits.

The algorithm initially searches for words similar to the keywords in the service and data type metadata. The similarity threshold can adjusted. If no hits are found, it becomes necessary to fall back on approximate expression matching. There are two widely used approaches for approximate expression matching: the Hamming distance [12] which compares strings of the same length and the Levenshtein distance [13], which compares two strings not necessarily having the same length, measuring by the minimum number of insertions, deletions, and substitutions of characters required to transform one string into another. Levenshtein distance is also known as the matching with k differences or errors. If the search does not generate hits, the system suggests alternative search terms ("did you mean…?"). These plausible alternatives to the user's query are obtained by computing the Levenshtein distance automatically (and letting the user influence the suggestions) to identify words similar to each keyword, and to estimate the distance using multiple keywords.

Magallanes uses a feedback mechanism to continually learn and refine its discovery capabilities. The feedback values record user selections of resources associated with specific keywords. This value is adjusted when the user selects another resource using the same keyword.

3.3 Metadata for Workflows

The ACGT scientific workflows are digital entities of their own and therefore metadata are also needed in order to describe their origin, functionality, etc. Before delving into the details about the workflow annotation in Figure 17 we show the "workflow lifecycle" that is supported in the ACGT platform. It is important to emphasize the following:

- By default each workflow originates as private in its author's storage area: no one else but its original author and creator is able to view and edit it.
- Due to the way BPEL is designed each edit of the workflow in the Editor does not result in a version of it that is immediately executable. In order for some version of the workflow to become "ready to run", a "Deployment" (in the BPEL Enactor) must be performed. This means that the abstract representation of the workflow in the ACGT Workflow Editor must be transformed to BPEL and registered with the Enactor. The details of the BPEL transformation are given in Section 2.5.
- Each new version of a workflow in the Editor overwrites the previous, i.e. only the most recent version is kept. The same is also true for the executable versions. It can therefore be the case that the two versions, the BPEL executable one and the other that is the editable in the Editor, become "out of synch" with each other.
- After the deployment the workflow (actually its BPEL version) can be executed but it's still private. If the user wants to share it with the community then a publication step is necessary during which the workflow is registered in the Metadata Repository as a new service.
- After the publication the workflow is available to all legitimate ACGT users for execution but the downside is that it's not editable anymore. Nevertheless any user can see it and save a copy of this that can be edited. The new copy is again private and the whole edit/deploy/publish circle can be repeated by the author of the original workflow or any other ACGT user.

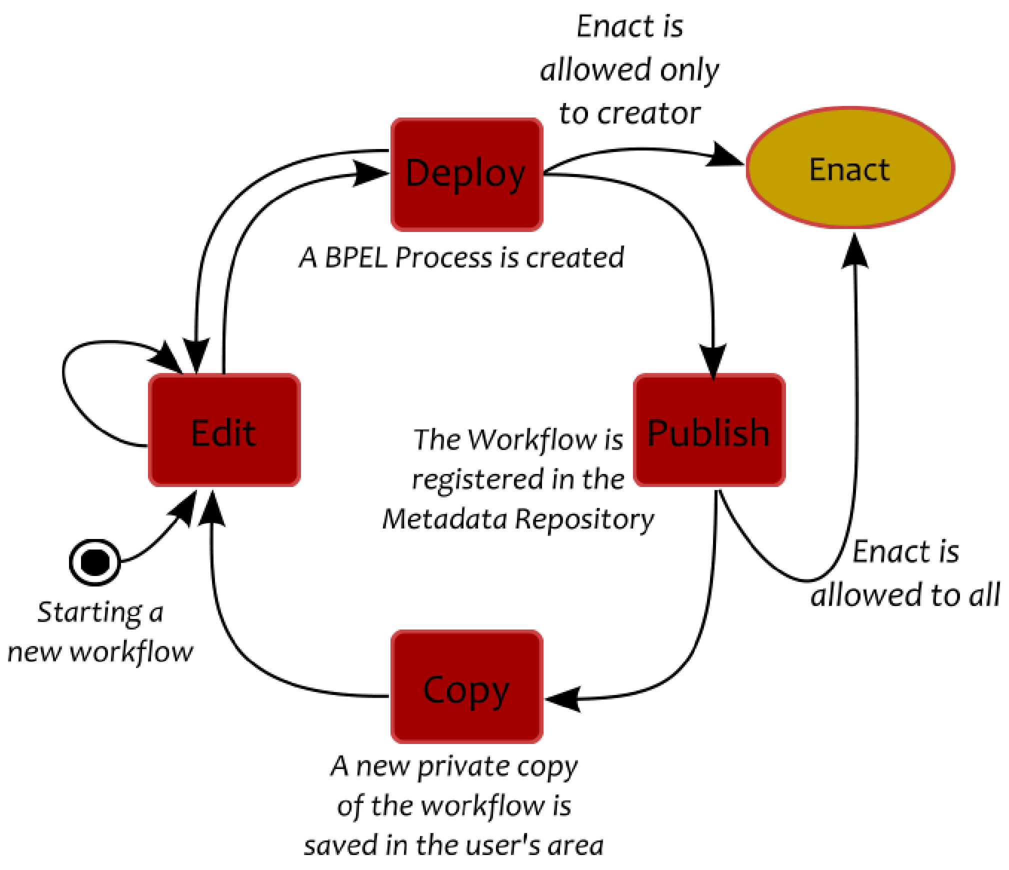

Enact is allowed only to creator
Deploy
Enact
A BPEL Process is created
Edit
Publish
The Workflow is registered in the Metadata Repository
Enact is allowed to all
Starting a new workflow
Copy
A new private copy of the workflow is saved in the user's area

Figure 17 The workflow lifecycle

Based on this lifecycle there are different types of workflow metadata. During the whole cycle the basic set of metadata used for workflows follow the Dublin Core Metadata Element Set [7] which has been standardized as ISO Standard 15836:2009. In particular we use:

- *Title*, the actual name of the workflow given by the user who created it
- *Creator*, the identification for the user who authored the workflow. This identification is based on the ACGT user certificate, i.e. it's his X.509 Distinguished Name (DN).
- *Date*, the UTC date (and time) in the ISO 8601 format, of the workflow's last modification. Furthermore, the initial creation data and time in kept through the *Created* metadata element.
- *Description*, a free text specification of what the workflow does or any other description given by the user.
- *Subject*, the list of the keywords ("tags") given by the user that possibly describe the "topic" of the workflow. There's no constraint on the what these keywords can be, e.g. drawn from a controlled vocabulary. The user is allowed to use whatever words or phrases he/she can come up with. Please refer to paragraph 3.3.1 below for the rationale for using tags.
- *Source*, the identifier of the original workflow that the described one was derived from. This metadata element is filled in when a new copy of a workflow is saved: the newly created workflow references back the copied workflow. This way a "pedigree" chain of workflows can be created where each one is based (in whole or in part) to another. Of course if a workflow is created from scratch then there's no "source" metadata element attached to it.

This set of elements is absolutely minimal for providing descriptive metadata of the users' workflows. More technical metadata are supported when a workflow is deployed in the enactor, i.e. when it has been transformed to BPEL (Section 2.5) and can be executed. These metadata elements include:

- The *Date* when the actual "deployment" took place, so that the system is able to identify cases where the deployed BPEL version of the workflow does not agree (i.e. it's older) with the most recent version that the user edited in the workflow editor
- The *Endpoint* where the BPEL workflow is accessible from. According to the BPEL specification a newly deployed workflow is a Web Service that can be sent requests in order to start execution and therefore this HTTP based endpoint is its entry point that triggers its enactment.

After the registration the workflow becomes a new service/tool, it is registered in the Metadata repository and therefore all the metadata information that is applicable for common services and tools, such as the functional category, semantic information about input and output parameters, etc. are also used.

3.3.1 Why "Tagging"?

A quite popular way of classifying content in Web 2.0 web sites is through "tagging". A tag is a keyword which acts like a subject or category. The user is allowed to attach whatever keywords she wants to identifiable content such as links in the case of social bookmarking, or videos and photographs in the case of digital content sharing. The important thing is that tags can be shared, used in searches, or recommended based on the choices of other users for the same content. The new term "folksonomy", as a fusion of the words "folks" and "taxonomy", has been suggested to describe this method of classifying content through tags that are collaboratively generated and shared. Of course these "poor man's" classification schemes are informal in nature, could contain duplication in meaning, or be simply erroneous but again they are contributed by the users and the more people contributing the more robust and stable these "folksonomies" become. A self adapting and auto regulating method is usually followed through the use of tag clouds (Figure 18). In simple terms a tag cloud is a visual representation of a user's tags where each tag is weighted based on the user preferences and how many times he has used the tag. Through such an approach "good" tags are likely to prevail assuming that the user participation is high.

In the ACGT Workflow Environment the tags are not shared among the users. For example there's no possibility for one user to see another user's tags. This isolation was again influenced by the security restrictions imposed by the ACGT architecture.

algebra algorithms altavista amazon
androulakis animation apache apache-ode
api app apple architecture arrows art
articles ascii asynchronous atom atomic
atomstore Authentication autocomplete
autoincrement Backup balancer bandwidth
barcodes bash Bayes BBC bcrypt Benford
Berkeley beryl beta bibliography bibtex
binary_search bioinformatics Biology
biomoby bionformatics bios bit-bytes
BK-tree blob bloom-filters book
bookmarking bookmarks books books-law
boost boosting BPEL bpmn breast-cancer
browser browsers bsd bsd-is-dying
business button C c# c++ caching
calculus cancer canvas cap captcha
categorization category-theory certificates
certification-authority chain-letters characters
checklist choreography Church classes-
as-instances classification clipart closures
cluster clustering code code-as-data codes
coffee cog color combinator comedy comet
comonads comparison compilers
complexity compression computation
computer concurrency condor consistency
constraint content-type content_negotiation
context-free converter cool cool-uris coop
coordination CORBA coroutines Course
CPR CRAP critters cron CRUD cryptography
cs csp css cts cts2 curl currency D.O.D
DAIS data data-mining data-structures

Figure 18. A tag "cloud"

3.3.2 Workflow Provenance

Provenance is important information gathered during the execution of a scientific workflow that ensures the reproducibility of the analyses performed [21]:

> *Metadata in the form of provenance information records the how, where, what, when, why, which, and by whom of data generated in a scientific experiment. Scientists can manually record provenance information, or software tools can automatically generate it. Scientists traditionally use provenance information, along with (implicit) domain expertise, to interpret and evaluate data accurately. Provenance information also lets researchers verify and validate experimental procedures.* [19]

In the ACGT Workflow Environment the Open Provenance Model[13] is used as the underlying ontology for recording and inferring the provenance information. The Open Provenance Model (OPM) defines the following entities (or "Nodes" as they say in the provenance graph):

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

[13] http://openprovenance.org/

The basic relationships between these entities are depicted in Figure 19. The artifacts are shown with circles, the Processes with rectangular shapes, and the Agents with octagons.

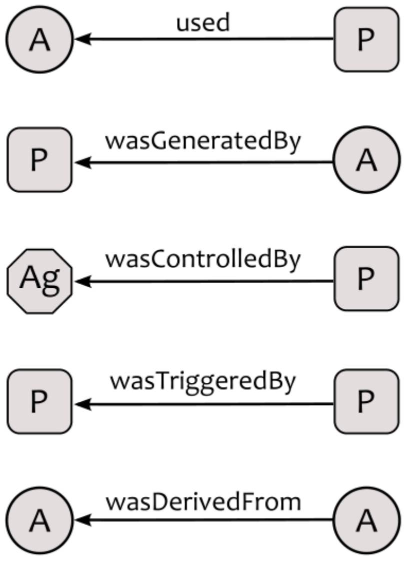


Figure 19 The major entities of OPM and their relationships

For the ACGT Workflow Environment the data files persisted in the Grid Data Management System (DMS) are the sole Artifacts. These files are immutable, although the immutability is not enforced by the system but rather it is a convention followed by the majority of the ACGT services. The DMS files are used as parameters or created as output in the services' invocations. The ACGT services therefore are modeled as Processes and the same is also true for the workflows themselves. In our provenance graphs the user is modeled as the Agent that originates and monitors the execution of the workflow. The causal relationships used in the ACGT workflow provenance graphs are:

- *Used*. This relationship binds together a Process (i.e. a service or workflow invocation) to its input data file(s).
- *WasGeneratedBy*. This is used to connect the output data files of Process with the Process itself.
- *WasControlledBy*. This relates the user (Agent) to the workflow (Process) enacted.
- *WasTriggeredBy*.This relationship connects the (source) Process that represents the service execution in the context of a workflow to the (target) Process that represents the same workflow enactment. Normally this type of relationship between the workflow and its constituent services is a kind of containment or membership but unfortunately the Open Provenance Model lacks any such notion. Therefore the *WasTriggeredBy* relationship seems to be a not identical but yet similar in meaning property, in the sense that it is compliant with the semantics that the run completion of the workflow required the execution of its participating services.

Implementation

The proxy services infrastructure described in Section 2.4 is used for the unobtrusive generation of provenance data. The proxy services work as interceptors of the requests to the ACGT services and therefore they have access to the data used and produced during the execution of the ACGT tool. Therefore they are perfect tools for the job and so far the proxies for GridR scripts and Semantic Mediator queries have been enhanced to persist the provenance information in a specific database that is part of the Workflow Environment.

Each wholly or partially successful enactment of a workflow results in the generation of a provenance graph. The unique enactment id therefore is used to also identify this provenance graph. In these graphs all the service executions of the proxies that support the generation of provenance information are recorded as Processes with their causal relationships and the DMS files are represented as Artifacts that were either consumed or produced by the services. The workflow itself is represented as a Process that triggered the execution of the participating services, as described above.

The OPM, in agreement with the Semantic Web, requires the use of Unique identifiers for the identification of the entities participating in a provenance graph and for the graph itself. The provenance component of the ACGT Workflow Environment uses HTTP and HTTPS URIs to identify the Artifacts and Processes to provide identification, retrieval, and linking facilities for constructing a web of data and metadata in accordance with the Semantic Web vision. The agents (users) are identified by their unique X.509 DN.

References

[1] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson et al., “The design and implementation of Grid database services in OGSA-DAI,” Concurrency and Computation: Practice & Experience, vol. 17, no. 2, pp. 357–376, 2005.

[2] A. Arkin, S. Askary, B. Bloch, I. Francisco Curbera, B. Yaron Goland, N. Kartha, S. Commerce, and O. Alex Yiu, “Web Services Business Process Execution Language Version 2.0.” [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[3] A. Barker and J. van Hemert, “Scientific Workflow: A Survey and Research Directions,” LECTURE NOTES IN COMPUTER SCIENCE, vol. 4967, p. 746, 2008.

[4] B. Claerhout, N. Forgó, T. Krügel, M. Arning, and G. De Moor, “A Data Protection Framework for Transeuropean genetic research projects.” Studies in health technology and informatics, vol. 141, p. 67, 2008.

[5] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2):86-93, 2002.

[6] F. DeRemer and H. Kron, “Programming-in-the large versus programming-in-the-small,” ACM SIGPLAN Notices, vol. 10, no. 6, pp. 114–121, 1975.

[7] Dublin Core Metadata Initiative. Dublin core metadata terms. http://dublincore.org/documents/dcmi-terms/, Last accessed: August 2010

[8] Freed, N. and Borenstein, N. Multipurpose internet mail extensions (MIME) part two: Media types, RFC 2046, November 1996

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Reading, MA, ISBN: 0201633612, 1995

[10] M. Garcia, J. Karlsson, and O. Trelles: Web service catalogue for Biomedical GRID infrastructure, Studies in health technology and informatics, 2010, 159: 76-87

[11] Antoon Goderis, David De Roure, Carole Goble, Jiten Bhagat, Don Cruickshank, Paul Fisher, Danius Michaelides, and Franck Tanoh. Discovering scientific workflows: The ^{my}Experiment benchmarks. IEEE Transactions on Automation Science and Engineering, April 2008.

[12] RW. Hamming: Error detecting and error correcting codes. The Bell System Technical Journal 1950, 29(2):147-160.

[13] V. Levenshtein: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physsics-Doklady 1966, 10(8):707-710. Original in Russian in Doklady Akademii Nauk SSSR (1965) 163(4):845-848.

[14] P. Lord, P. Alper, C. Wroe, and C. Goble. Feta: A light-weight architecture for user oriented semantic service discovery. The Semantic Web: Research and Applications, pages 17-31, 2005.

[15] V. Martin-Requena, et al: PreP+07: improvements of a user friendly tool to preprocess and analyse microarray data; BMC Bioinformatics 10(16), 2009.

[16] L. Philips. The double metaphone search algorithm. C/C++ Users Journal, 18(6):43, 2000.

[17] J. Pukacki, et al: Programming Grid Applications with Gridge. Computational Methods in Science and Technology; 12(1), 47-68, 2006

[18] J. Ríos, J. Karlsson, and O. Trelles. Magallanes: a web services discovery and automatic workflow composition tool. BMC Bioinformatics (2009) 10:334.

[19] S. S. Sahoo, A. Sheth, and C. Henson, "Semantic Provenance for eScience: Managing the Deluge of Scientific Data", IEEE Internet Computing, vol. 12, no. 4, 2008, pp. 46-54

[20] Semantic Annotations for WSDL and XML Schema, W3C Recommendation 28 August 2007, http://www.w3.org/TR/sawsdl/

[21] Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Record 34 (3) (2005) 31-36

[22] D. Wegener, T. Sengstag, S. Sfakianakis, S. Ruping, and A. Assi, “GridR: An R-Based Grid-Enabled Tool for Data Analysis in ACGT Clinico-Genomics Trials,” in e-Science and Grid Computing, IEEE International Conference on, 2007, pp. 228–235.

[23] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, “Security for Grid services,” in High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, 2003, pp. 48–57.

[24] M.D. Wilkinson and M. Links. BioMOBY: An open source biological web services proposal. Briefings in Bioinformatics, 3(4):331-341, 2002.

[25] Wilkinson M.D. et. al Interoperability with Moby 1.0 - it's better than sharing your toothbrush!, Brief Bioinform. 2008 May;9(3):220-231

[26] K. Wolstencroft, P. Alper, D. Hull, C. Wroe, PW Lord, RD Stevens, and CA Goble. The ^{my}Grid ontology: bioinformatics service discovery. International journal of bioinformatics research and applications, 3(3):303-325, 2007.

[27] K. Zyp, A JSON Media Type for Describing the Structure and Meaning of JSON Documents, Internet-Draft, March 23, 2010. Available at http://tools.ietf.org/html/draft-zyp-json-schema