



Service based access to Oncosimulator - report

Project Number: FP6-2005-IST-026996
Deliverable id: D 4.5
Deliverable name: Service based access to Oncosimulator - report
Submission Date: 20/07/2010

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 4.5
Document name:	Service based access to Oncosimulator - report
Document type (PU, INT, RE)	PU
Version:	1.0
Date:	20/07/2010
Editor: Organisation: Address:	Juliusz Pukacki PSNC pukacki@man.poznan.pl

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

ABSTRACT:

The present document provides a description of Oncosimulator Service implementation. It is the alternative way of accessing Oncosimulator code integrated into workflow environment of ACGT.

KEYWORD LIST: Grid, grid services

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	10/03/2010	Draft	Juliusz Pukacki
0.2	12.05.2010	Draft	Juliusz Pukacki
0.3	20.07.2010	Draft	Michał Krysiński
1.0	28.07.2010	Final	Juliusz Pukacki

List of Contributors

- Michał Krysiński, PSNC
- Stelios Sfakianakis, FORTH
- Giorgos Zacharioudakis, FORTH
- Paweł Spychała, PCNS

Contents

ACGT ARCHITECTURE - OVERVIEW.....	8
ACGT WORKFLOW ENVIRONMENT.....	8
ONCOSIMULATOR SERVICE.....	10
INTRODUCTION.....	13
GENERIC ONCOSIMULATOR WEB SERVICE IMPLEMENTATION.....	13
PROXY ONCOSIMULATOR WEB SERVICE IMPLEMENTATION	31

Illustrations

Illustration 1: ACGT architecture.....	8
Illustration 2: Workflow Enactment Architecture.....	9
Illustration 3: Oncosimulator services - overview.....	11
Illustration 4: Oncosimulator services - implementation details.....	12
Illustration 5: ACGT Web Service Wizard welcome screen.....	13
Illustration 6: Step 1.....	14
Illustration 7: Java2WSDL settings.....	15
Illustration 8: WSDL2Java settings.....	15
Illustration 9: Create new operation dialog.....	16
Illustration 10: Add argument dialog.....	17
Illustration 11: Add structure.....	17
Illustration 12: Structures dialog.....	18
Illustration 13: Structure editor.....	19
Illustration 14: Step 2.....	20
Illustration 15: Step 3.....	21
Illustration 16: Step 4.....	22
Illustration 17: Step 5.....	23
Illustration 18: Step 6.....	24
Illustration 19: Step 7.....	25

Executive Summary

ACGT is an Integrated Project (IP) funded in the 6th Framework Program of the European Commission under the Action Line “Integrated biomedical information for better health”. The high level objective of the Action Line is the development of methods and systems for improved medical knowledge discovery and understanding through integration of biomedical information (e.g. using modelling, visualization, data mining and grid technologies). Biomedical data and information to be considered include not only clinical information relating to tissues, organs or personal health-related information but also information at the level of molecules and cells, such as that acquired from genomics and proteomics research. ACGT focuses on the domain of Cancer research, and its ultimate objective is the design, development and validation of an integrated Grid enabled technological platform in support of post-genomic, multi-centric Clinical Trials on Cancer. The driving motivation behind the project is our committed belief that the breadth and depth of information already available in the research community at large, present an enormous opportunity for improving our ability to reduce mortality from cancer, improve therapies and meet the demanding individualization of care needs.

Introduction

The Oncosimulator is an advanced information system which is able to simulate the response of tumours and affected normal tissues to therapeutic schemes based on clinical, imaging, histopathologic and molecular data of a given cancer patient. It aims at optimizing cancer treatment on a patient-individualized basis by performing *in silico* (on the computer) experiments of candidate therapeutic schemes.

Oncosimulator is implemented as a standalone sequential application that can be compiled and executed on computer running under Linux/Unix or Windows operating system. The input required to run the application consists of the initial shape of the tumour and set of parameters describing patient and treatment.

The basic and the most important scenario concerning Oncosimulator in ACGT project is a grid execution of the code. For that purpose a dedicated end user application was developed called OncoRecipeSheet. Using that interface researcher is able to define computational experiment by describing multiple runs of the simulator with different parameters. He is also able to search the results of a previous runs and visualize it in the same time.

In the background of the OncoRecipeSheet there is a ACGT grid infrastructure that takes care of the execution of the code on some grid node based on current state of the grid resources. The component responsible for the resource management is called GRMS (Grid Resource Management System). After the job is done it also handles transferring output files to data management system of the grid called GDMS (Grid Data Management System). Files that are stored there can be downloaded by the user or can be used then by visualization system to generate pictures or videos.

Because of the specific clients tool - OncoRecipeSheet - oncosimulator scenario is separated from the rest of the ACGT scenarios that are accessed via portal client. Although the integration is possible on the level of files in GDMS there is a need of much tighter integration of the simulation with rest of ACGT instruments.

The main client tool for accessing ACGT infrastructure is the workflow environment. It provides easy access to different services defined in the ACGT environment, and supports defining of action flows between different components by connecting the output of one service with the input of the other.

This deliverable describes the integration of the Oncosimulator application with the workflow environment of ACGT.

Oncosimulator service in ACGT architecture

ACGT architecture - overview

Design view of the ACGT architecture presented on the picture below, consists of five main horizontal layers.

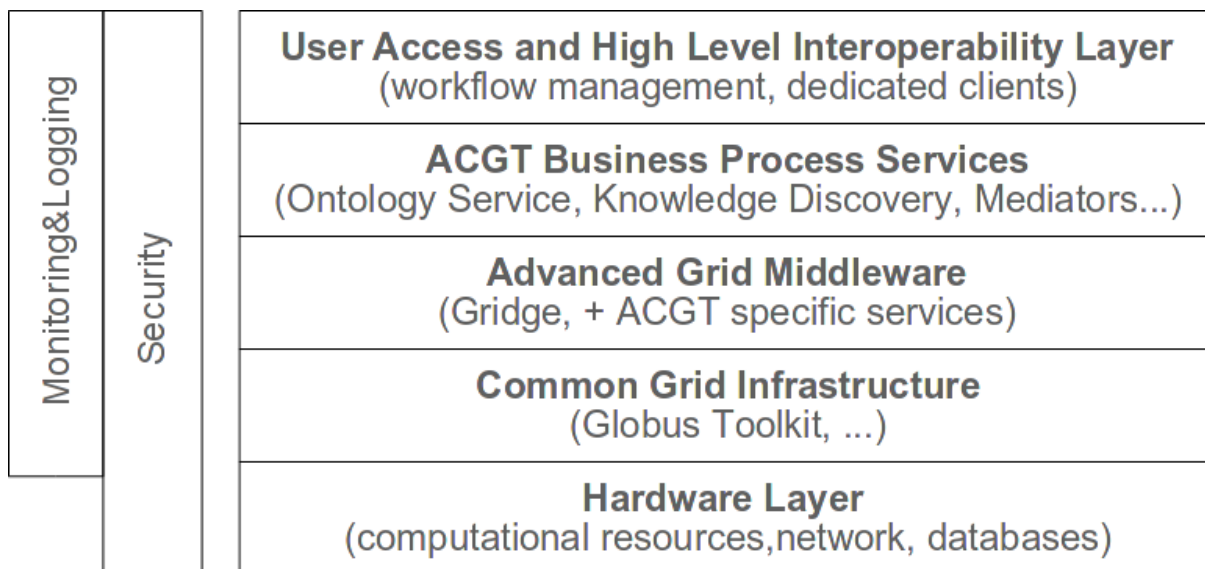


Illustration 1: ACGT architecture

The lowest one are the hardware resources where the computation is done and where physical data is stored. The role of the next layer above the hardware resources is to provide unified, remote access to physical resources. It provides basic functionality required for remote computing and data access (job execution and control, basic authentication and authorization, file transfer, databases access, hardware monitoring).

The last grid layer is Advanced Middleware Layer. It is responsible for providing more advanced mechanisms in the Grid environment. Services from this layer can be described as "collective" because they operate on set of lower level services, to realize more advanced actions - e.g. metascheduling service that submits jobs to different local queuing systems using Common Grid Infrastructure remote interfaces.

ACGT business process layer consists of components which are not aware of physical resources and Grid environment. Grid is used as a whole, to perform more abstract actions, and to get information required by end user. Services of this layer are using abstract description of the world defined as some ontology. There are also services that are able to translate high level description to language understandable by lower layer (grid). It provides higher level integration of different resources and data and makes them more similar to real world objects. It also a layer where the Oncosimulator service should be placed.

The User Access Layer contains all application and tools that provide access to the ACGT Environment for end user. There can be a wide variety of software, developed using different technology (portals, standalone applications, clients dedicated to specific operation, workflows editors, visualization tools). Client applications, are in the most cases tightly connected to functionality provided by underlying layer.

ACGT workflow environment

Workflow environment of ACGT consists of two components: Workflow Enactor and Workflow Editor. Workflow Enactor can be also called workflow engine because it is

responsible for managing of the execution of the submitted workflow. It is based on well known and commonly used BPEL (Business Process Execution Language) technology, used for describing components and precedence constrains among them.

Workflow editor is a GUI tool that allows a user to combine different ACGT services into complex workflows. This tool is accessible through the ACGT Portal and therefore has a web based graphical user interface. It supports the searching and the browsing of the available services and data sources and their composition through some intuitive and user friendly interface. The workflows created can be stored in a user's specific area and later retrieved and edited so new versions of them can be produced. The publication and sharing of the workflows are also supported so that the user community can exchange information and users benefit from each other's research. Finally the workflow editor supports the execution of the workflows and the monitoring of their enactment status

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.

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. These proxies or wrapper services provide BPEL friendly facades of the original, real ACGT services, effectively working as calls transformation bridges between the two worlds.

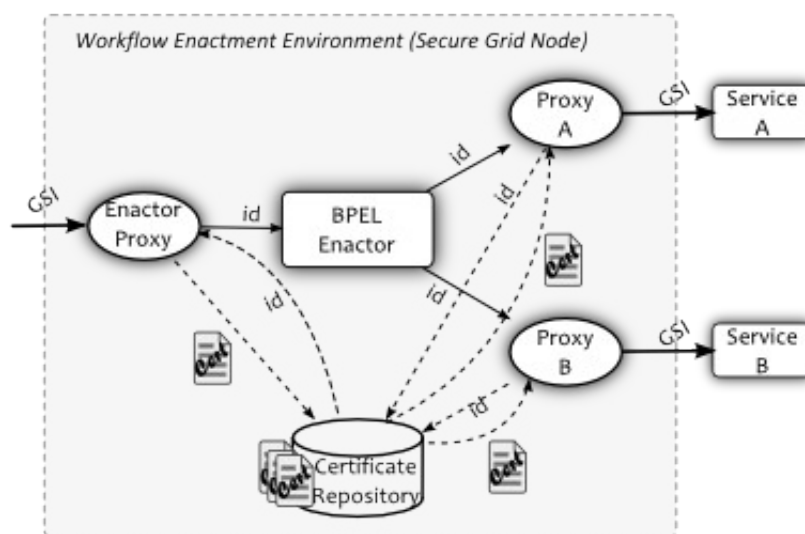


Illustration 2: Workflow Enactment Architecture

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 proxied. 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 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.
- 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

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. 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.

An additional point to make: 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 they are apparently many different WSDL styles. 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 in WSDL are the ones that are fully compliant with BPEL and ODE. Furthermore the BPEL transformation tool employed by the ACGT Workflow Editor works with the WSDL version 1.1 and its SOAP 1.1 binding.

Oncosimulator service

There are two different strategies to implement Oncosimulator service depending on the way how the simulation code is executed

- incorporate simulation code into service
- create a service as a wrapper for grid execution of the oncosimulator code

The first solution is a very obvious and straightforward. The service itself includes the full functionality of the Oncosimulator. The advantage of this approach is based on simplicity of the implementation. But there are a lot of disadvantages: the simulation code that can be very computationally intensive job is executed on the same machine as the one where the service is deployed. It is also a solution very difficult to maintain - each change in the simulation code have to be introduced in the service code immediately.

The second approach seems to be much more convenient. Service plays the role of the wrapper for the grid execution of the simulation code. It is compliant with the basic oncosimulator scenario and can take advantage of the currently available infrastructure. Thus it is possible to exploit grid environment (resource management) for distributing computational jobs all over the grid and for storing the results of the simulation (data management)

The only clearly visible disadvantage of that solution is necessity of more implementation effort to achieve required functionality.

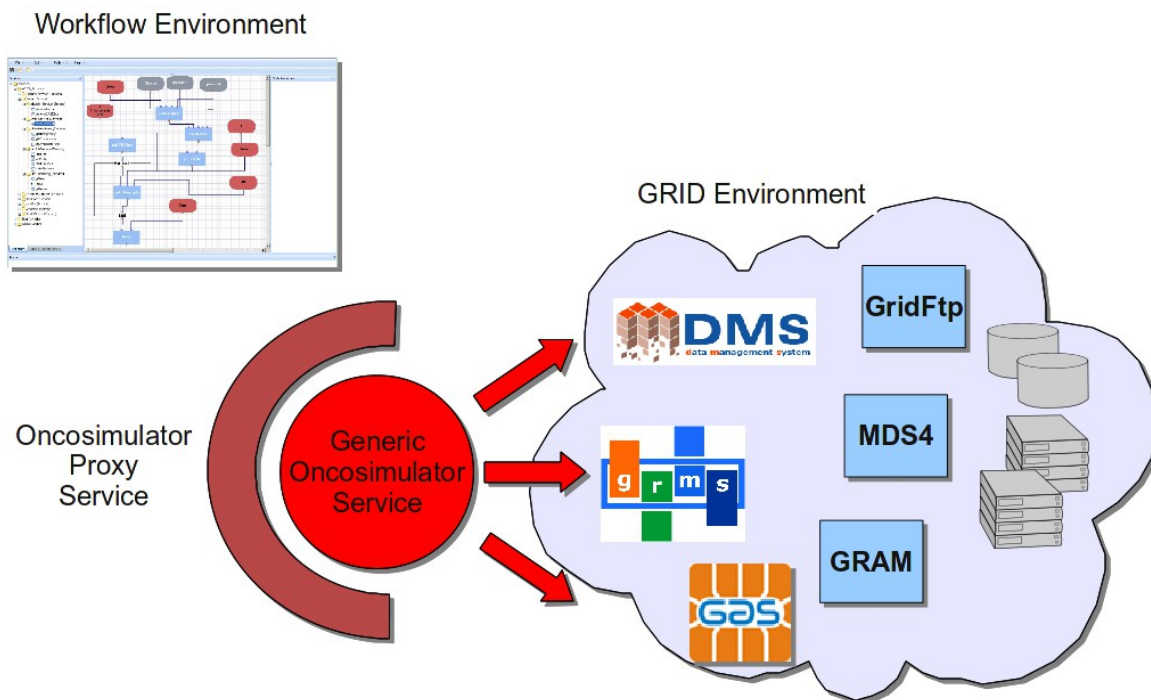


Illustration 3: Oncosimulator services - overview

Because of the limitations of the workflow environment described in a previous chapter two services are required to be implemented:

- Generic Oncosimulator Service (GOS)
 - GOS is a service that is wrapping calls to grid infrastructure for submitting simulation to the grid. The interface for running the simulation accepts parameters for the simulation invocation including logical identifiers of the input data stored in Data Management System of the grid (DMS). Also logical location of the output data generated by the application must be provided. Upon the call service dynamically creates proper job description for the grid metascheduler GRMS (Grid Resource Management System). Then job description (XML document) is submitted to GRMS that is responsible for the execution of the application in the grid environment:
 - finding proper computational node,
 - creating execution environment,
 - transferring input data,
 - execution of the code,
 - transferring output data to DMS

The call for running simulation is not a blocking one - it will not wait until the computation is done. It returns as soon as GRMS give back the identifier of the job.

GOS implements the notification mechanism that is used for keeping a track for the execution of the application in the grid. As soon as a status of the submitted job changed GRMS is sending the information about it to GOS.

For its clients GOS provides interface for checking the current status of the simulation. Status can be checked based on the identifier returned during submission call.

GOS is implemented as a GSI enabled web service - it requires credential delegation. Delegated credentials are used for job submission to the grid - GRMS requires the credentials to be able to act on behalf of the end user.

- Oncosimulator Proxy Service (OPS)
 - As described in a previous chapter GSI enabled web services needs to be wrapped with proxy service to be compliant with workflow environment of ACGT. Additionally OPS interface can be tailored to the needs of end users responsible for

creation of workflows. Thus it has vary flexible design that allows easy development and deployment different versions of service depending on user requirements. OPS is also responsible for communication with DMS and creation of logical identifiers of the files that GRMS will use for transferring output of the simulation.

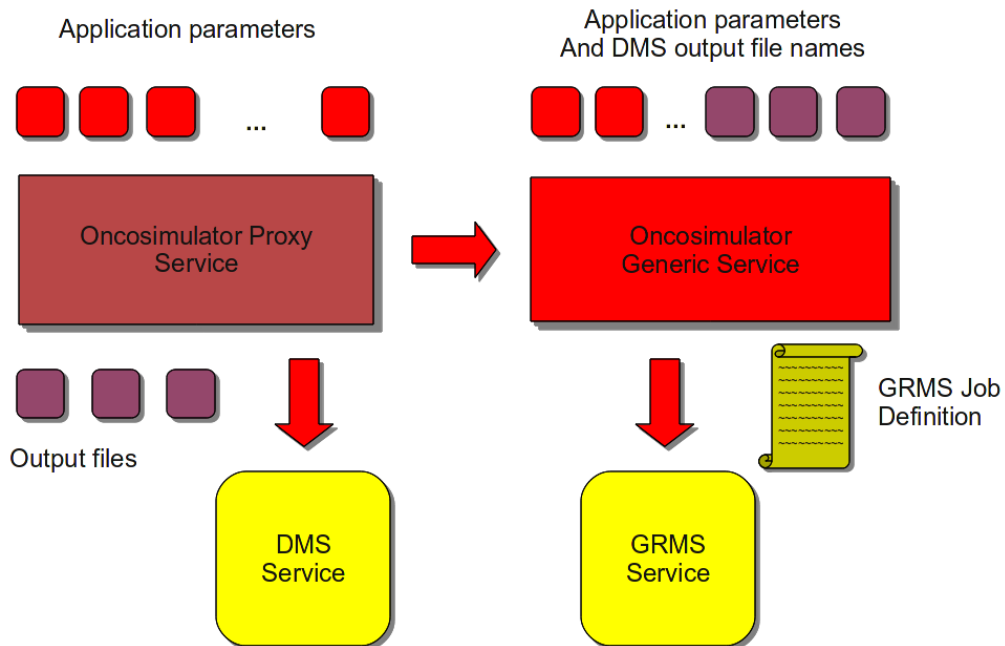


Illustration 4: Oncosimulator services - implementation details

Implementation of service

Introduction

ACGT Web Service Wizard was used in the process of implementing the Oncosimulator web services. It is a utility that facilitates the creation of GSI-enabled web services by automating activities such as: deploying Axis and Java-WS-Core into Tomcat, configuring Tomcat, generating WSDL file, stubs et etcetera. Rather than wasting time dealing with said complexities, the developers using ACGT Web Service Wizard can focus on solving the actual problem. This chapter describes the development of Oncosimulator services in detail. Emphasis will be put on how ACGT Web Service Wizard was used in the process.

Generic Oncosimulator Web Service implementation

Using ACGT Web Service Wizard

Before starting ACGT Web Service Wizard, the user has to make sure that Java Development Kit is installed and properly configured. In order to launch the application, user needs to change directory to directory containing the file `WebServiceWizard.jar` and execute the following command:

```
java -jar WebServiceWizard.jar
```

Alternatively, user can omit changing directory and specify full path to the jar file:

```
java -jar <path-to>/WebServiceWizard.jar
```



Illustration 5: ACGT Web Service Wizard welcome screen

Illustration 5 shows the first screen of ACGT Web Service Wizard. At this point user decides whether to create a new service or edit an existing one. Let's proceed by choosing the former and clicking **OK**.

The screenshot shows a window titled "Web Service Wizard" with a "Step 1" header. Below the header is a text box containing instructions: "Enter basic information about the service and specify operations (you can also do this in the second step). Use 'Structures' button to define classes, interfaces and enums used by your operations." The main area contains several input fields and buttons. The "Web Service name" field contains "RunSimulation" and has a "Settings" button to its right. The "Destination package" field contains "pl.psnr.runsimulation" and has a "Structures" button to its right. Below these are empty fields for "Base type" and "Implemented interfaces". There is a checkbox for "Enable GAS authorization" which is unchecked. Below it are fields for "GAS Location" and "GAS port number". At the bottom left is a section labeled "Operations:" with a large empty text area. To the right of this area are five buttons: "Add", "Edit", "Remove", "Move up", and "Move down". At the very bottom of the window are three buttons: "<< Prev", "Next >>", and "Close".

Illustration 6: Step 1

The panel shown in Illustration 6 is where the process of creating our service begins. Basic information can be entered using input fields located in the upper-centre of the panel. In this case we need to fill only the first two, namely **Web Service name** and **Destination package**. Clicking **Settings** in the top-right corner of the panel will bring up a dialog that allows you to configure WSDL2Java and Java2WSDL options (Illustrations 7 and 8). The values of **Location**, **Namespace** and **Namespace for implementation wsdl** presented in Illustration 7 are exemplary and can be modified according to your preference. Other values need to be set exactly as shown to ensure interoperability. As for WSDL2Java settings, it is advisable to keep default values.

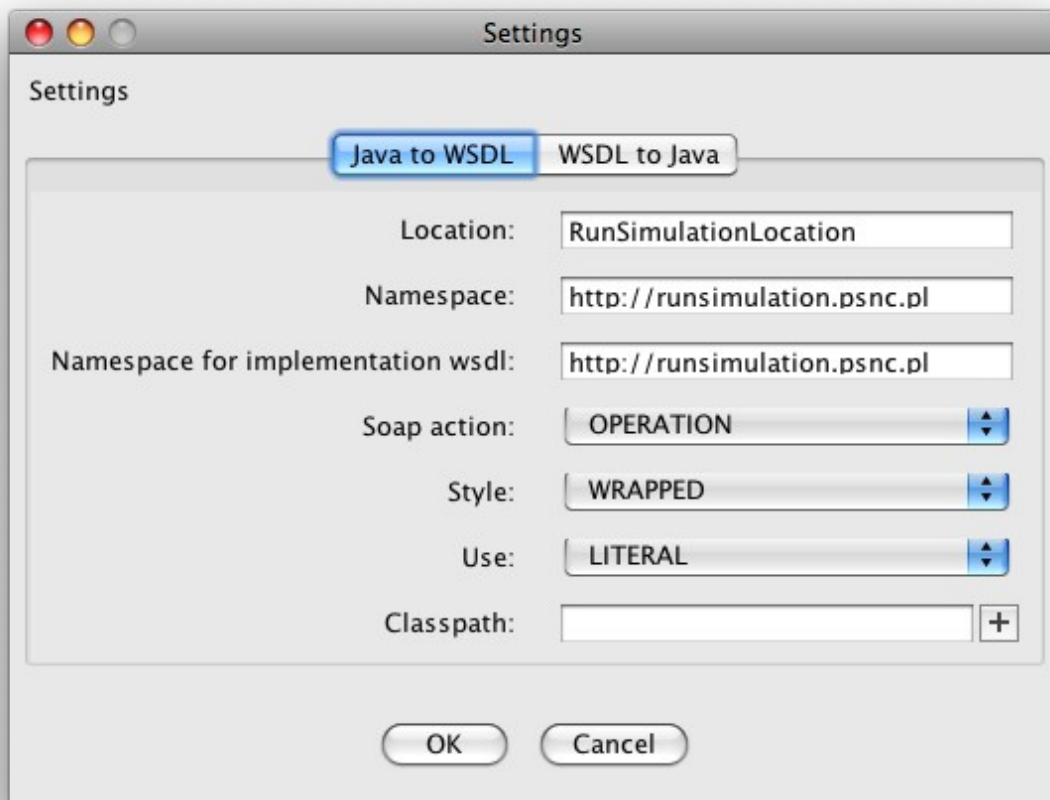


Illustration 7: Java2WSDL settings

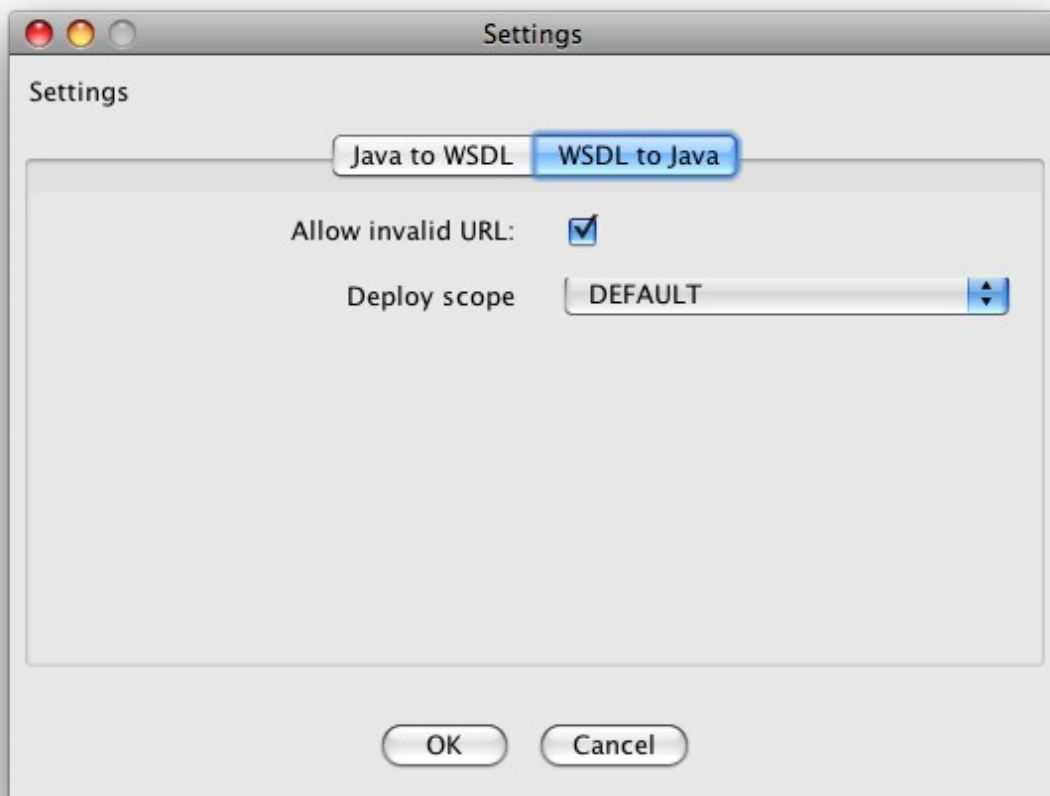


Illustration 8: WSDL2Java settings

Next, we need to define the operations of our service. Only “runSimulation” operation will be described here, as the remaining operations can be added analogically. Click **Add** to open the **Create new operation** dialog (Illustration 5). Specify the operation's name and return type by filling the corresponding fields. The dialog shown in Illustration 6 allows for adding arguments of the operation. The value entered in the **Name** box must be a valid Java variable name. If it is not, the ACGT Wizard will not produce a warning immediately, but an error will occur when the application tries to generate a WSDL file. Also the value in the **Return type** box has to conform to Java syntax.

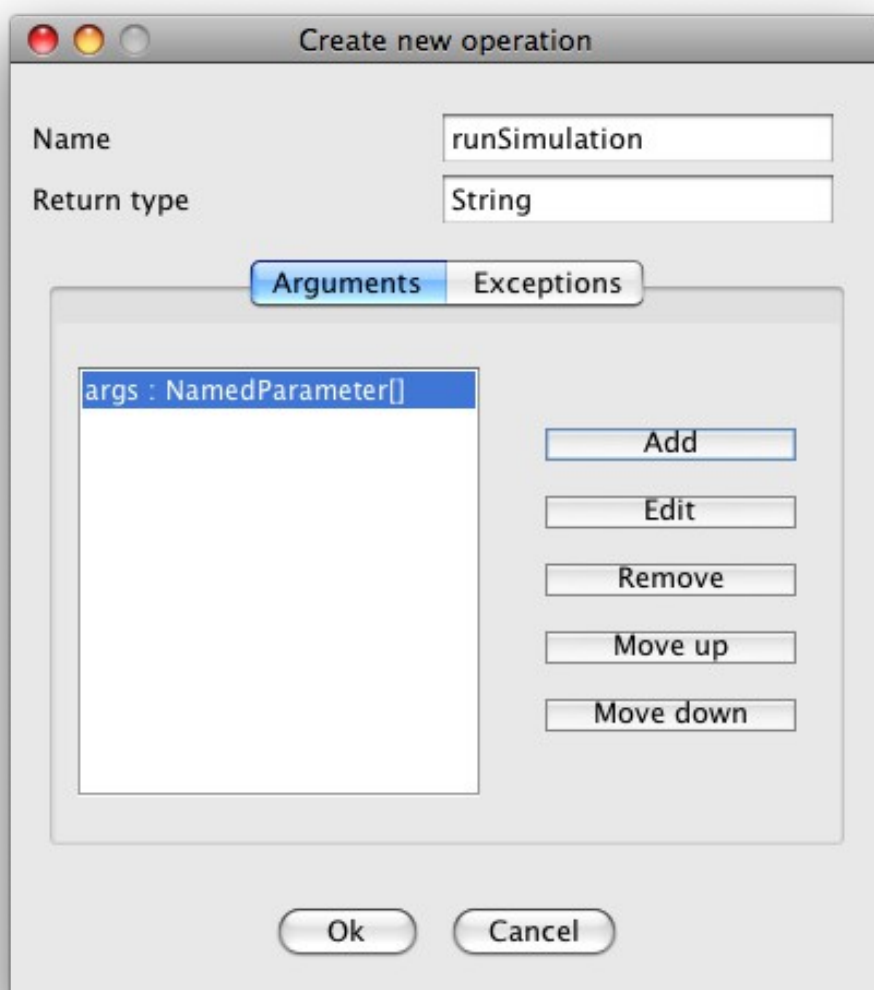


Illustration 9: Create new operation dialog

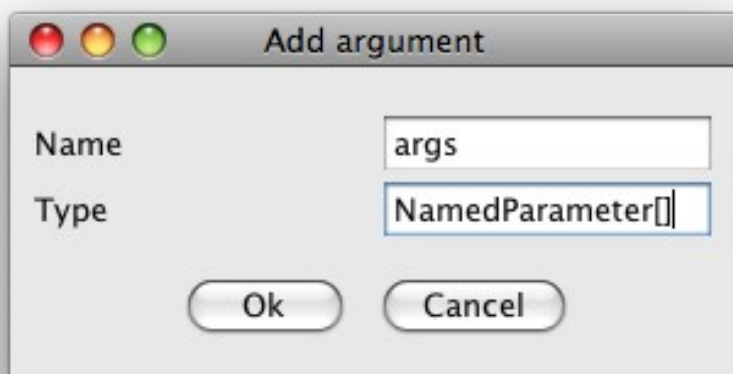


Illustration 10: Add argument dialog

After we have defined all the operations we are back at the **Step 1** panel. Note that we used an array of `NamedParameter` as argument for the “runSimulation” operation. It is not a standard Java type, so we need to define it. Open the structures dialog (Illustration 11) by clicking **Structures**. Next, click **Add**, which will bring up the window shown in Illustration 12. Fill in the fields and click **Ok**. In the newly opened window (Illustration 13) type in the code of the data structure.

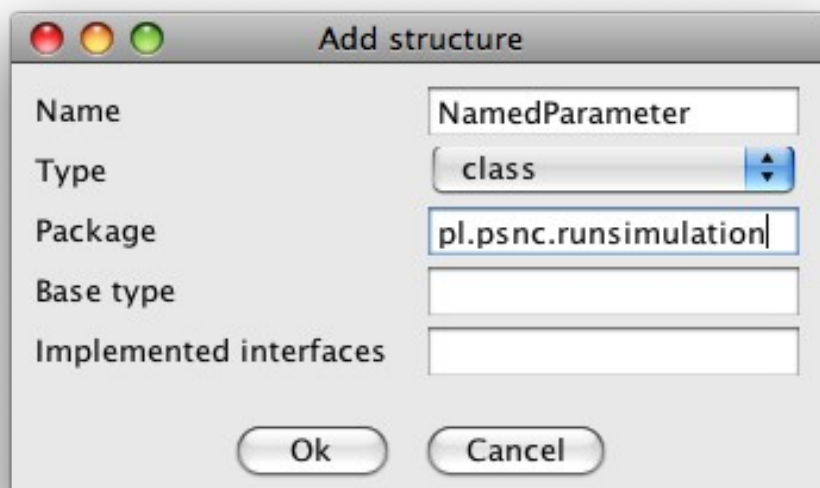
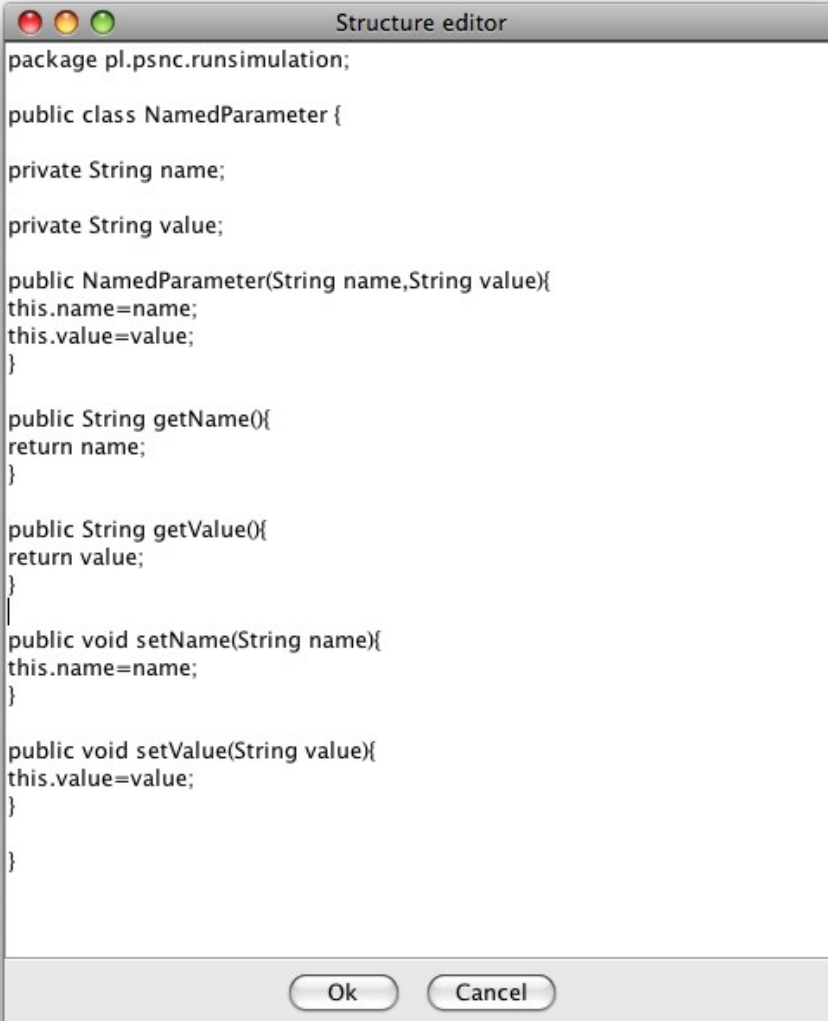


Illustration 11: Add structure



Illustration 12: Structures dialog

A screenshot of a 'Structure editor' window. The window has a title bar with three colored buttons (red, yellow, green) on the left and the text 'Structure editor' in the center. The main area contains Java code for a class named 'NamedParameter'. The code includes package declaration, class declaration, two private fields ('name' and 'value'), a constructor, and three methods: 'getName()', 'getValue()', and two setter methods ('setName()' and 'setValue()'). At the bottom of the window, there are two buttons: 'Ok' and 'Cancel'.

```
package pl.psnr.runsimulation;

public class NamedParameter {

    private String name;

    private String value;

    public NamedParameter(String name,String value){
        this.name=name;
        this.value=value;
    }

    public String getName(){
        return name;
    }

    public String getValue(){
        return value;
    }

    public void setName(String name){
        this.name=name;
    }

    public void setValue(String value){
        this.value=value;
    }

}
```

Illustration 13: Structure editor

Clicking **Next** takes us to the **Step 2** panel, where the code generated in the previous step is displayed (Illustration 14).

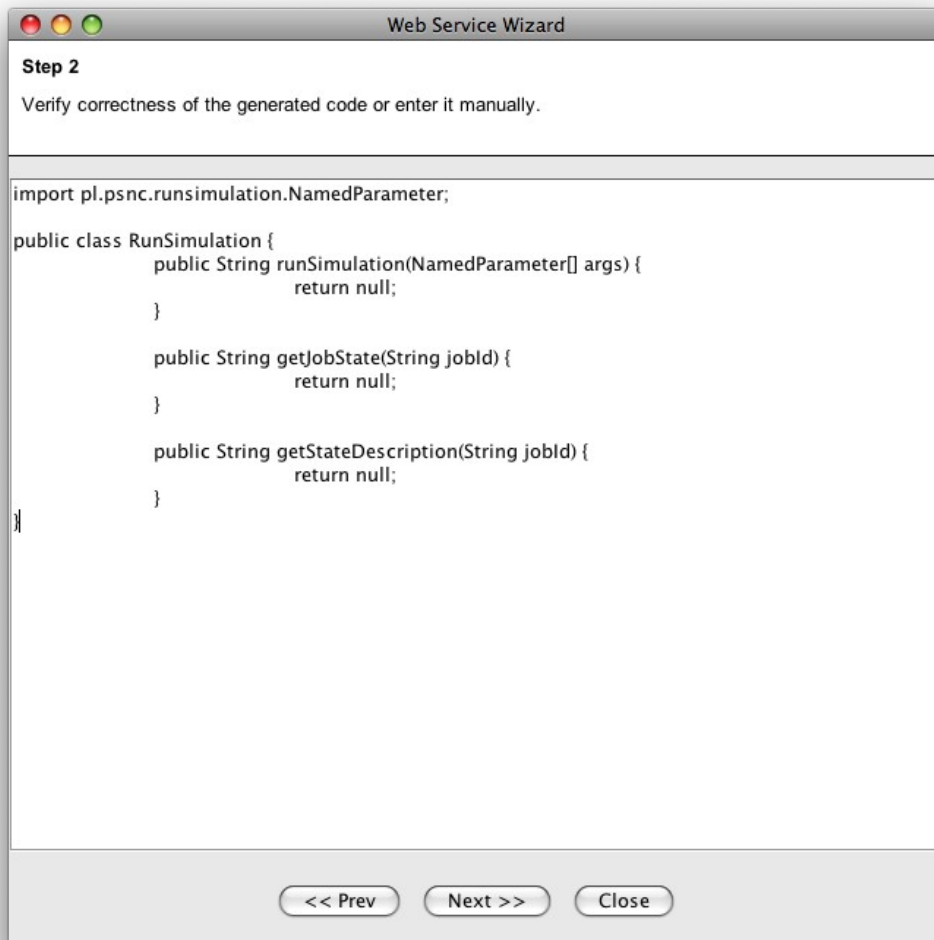


Illustration 14: Step 2

In the **Step 3** panel we can see the WSDL code generated in previous steps (Illustration 15).

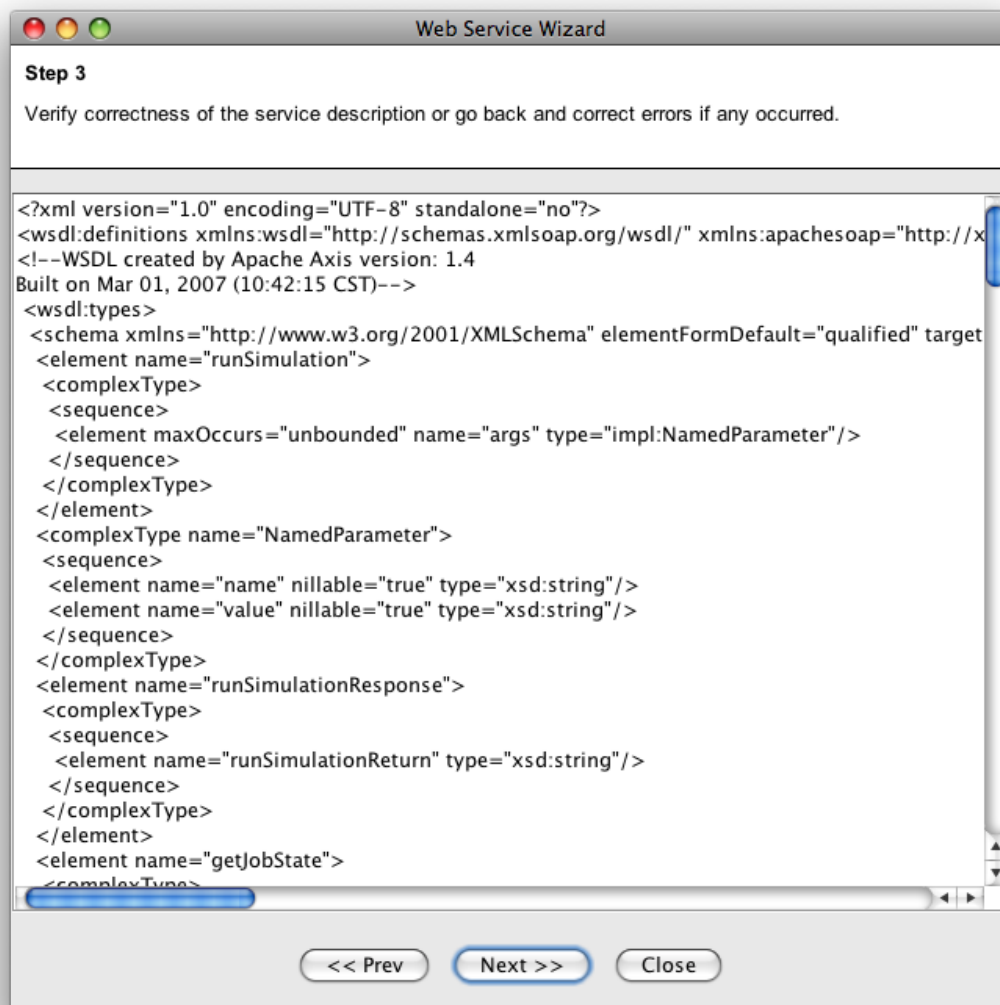


Illustration 15: Step 3

The next panel (**Step 4** – Illustration 16) can be used to type in the business logic code for the web service. The build-in editor does not offer advanced features such as code completion or inspections, therefore it is inconvenient for implementing more complex services. Luckily, in such cases we can use an IDE. Later in this chapter we will see how.

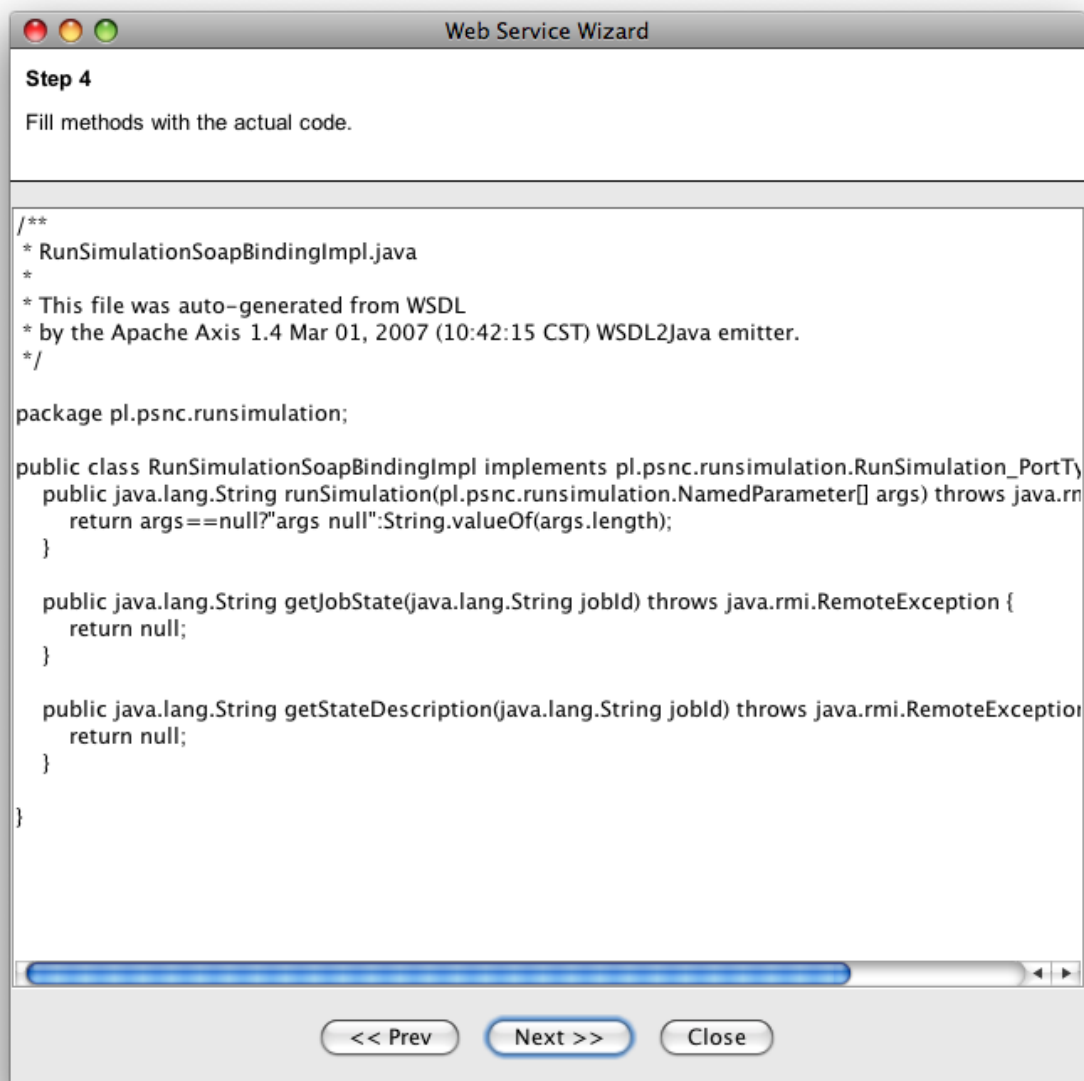


Illustration 16: Step 4

Next, we need to choose whether we want to deploy the service in a new container or an existing one. If we choose the latter, we need to make sure that the target container is properly configured (i.e. Axis and Java-WS-Core are deployed and configured and so on). Let's go with the first option and allow ACGT Wizard to download and configure a Tomcat instance for us (Illustration 17).

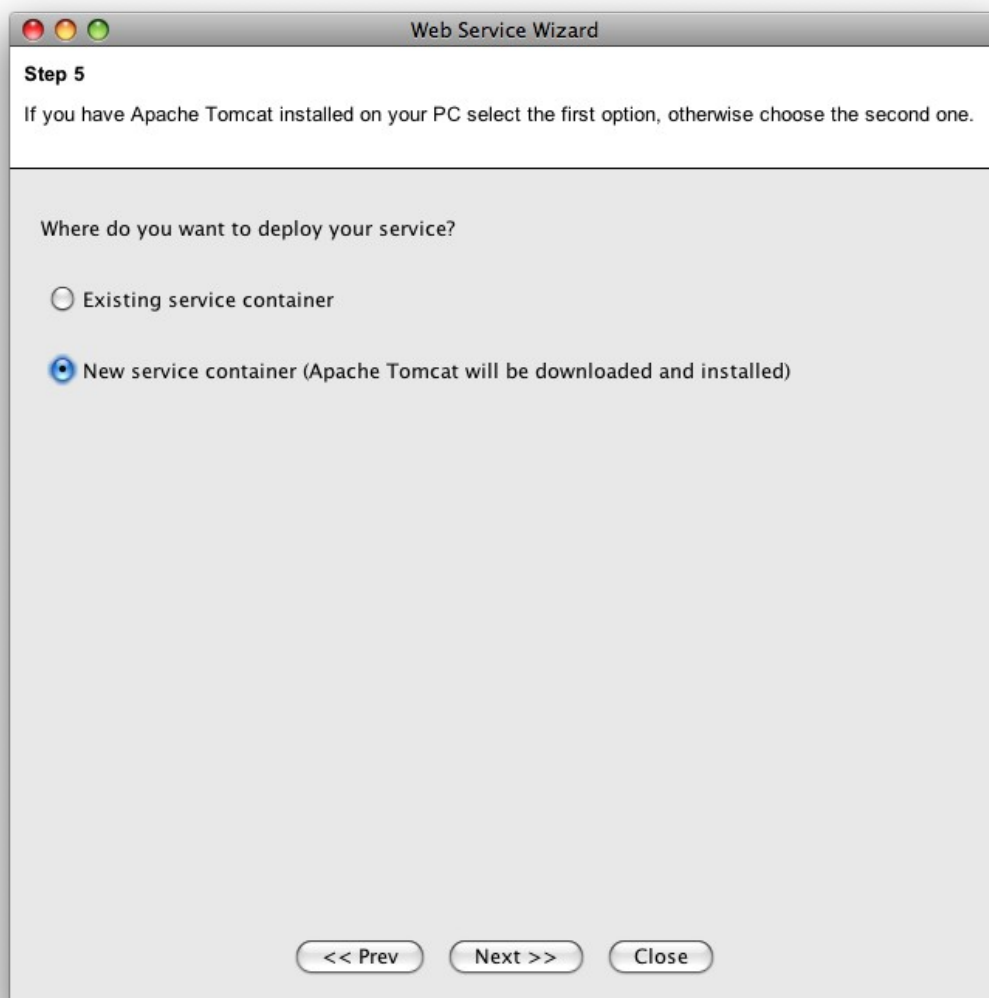
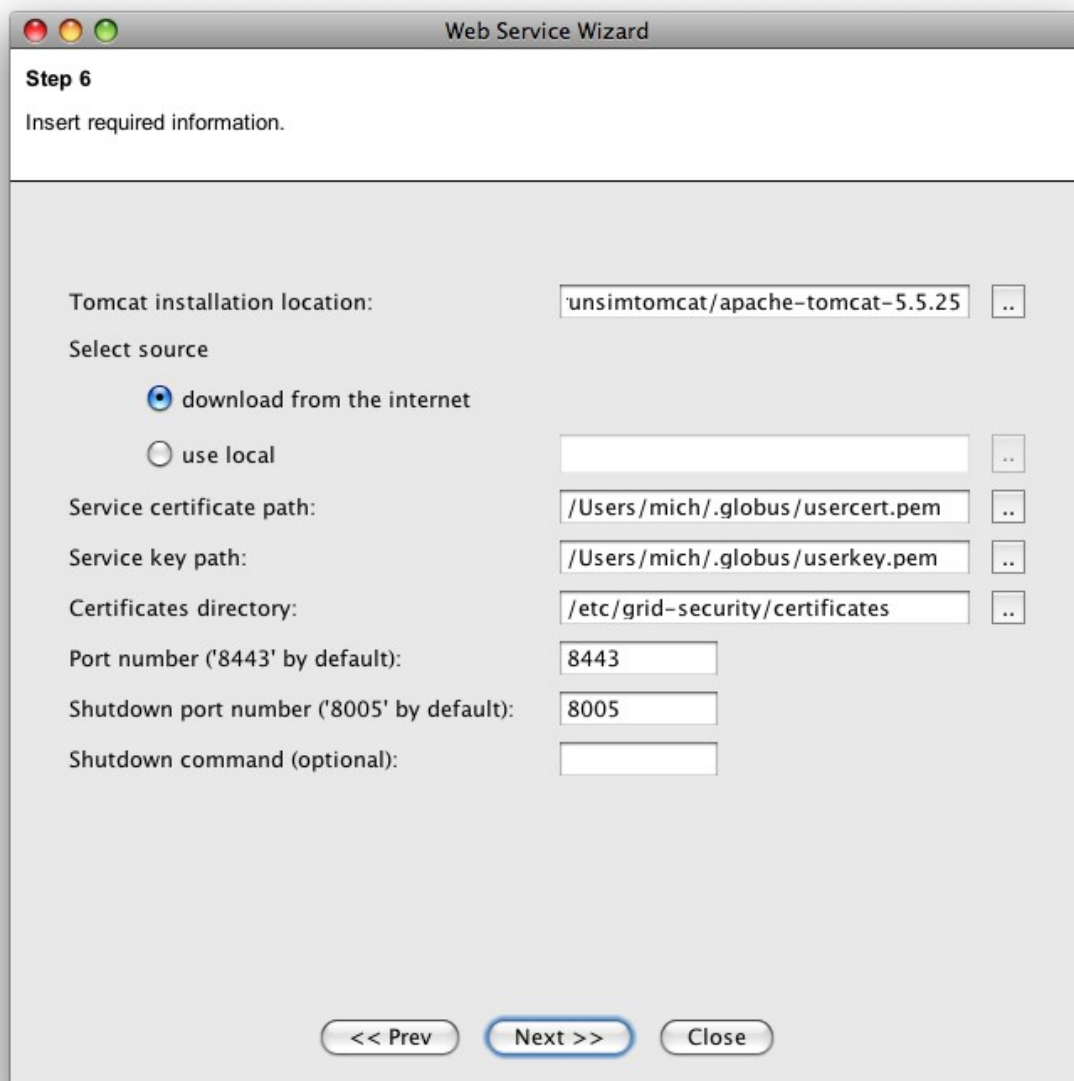


Illustration 17: Step 5

In **Step 6** (Illustration 18) we need to provide some deployment information such as Tomcat installation location, paths to certificate and key to be used by the service, port on which the service will listen for requests etc. After we have done that and pressed **Next** we are informed that once Tomcat is restarted, the service is ready to use (Illustration 19).



The screenshot shows a window titled "Web Service Wizard" with a header "Step 6" and the instruction "Insert required information." The window contains several configuration fields:

- Tomcat installation location:** A text box containing "unsimtomcat/apache-tomcat-5.5.25" and a browse button ("..").
- Select source:** Two radio buttons: "download from the internet" (selected) and "use local".
- Service certificate path:** A text box containing "/Users/mich/.globus/usercert.pem" and a browse button ("..").
- Service key path:** A text box containing "/Users/mich/.globus/userkey.pem" and a browse button ("..").
- Certificates directory:** A text box containing "/etc/grid-security/certificates" and a browse button ("..").
- Port number ('8443' by default):** A text box containing "8443".
- Shutdown port number ('8005' by default):** A text box containing "8005".
- Shutdown command (optional):** An empty text box.

At the bottom of the window are three buttons: "<< Prev", "Next >>" (highlighted with a blue border), and "Close".

Illustration 18: Step 6

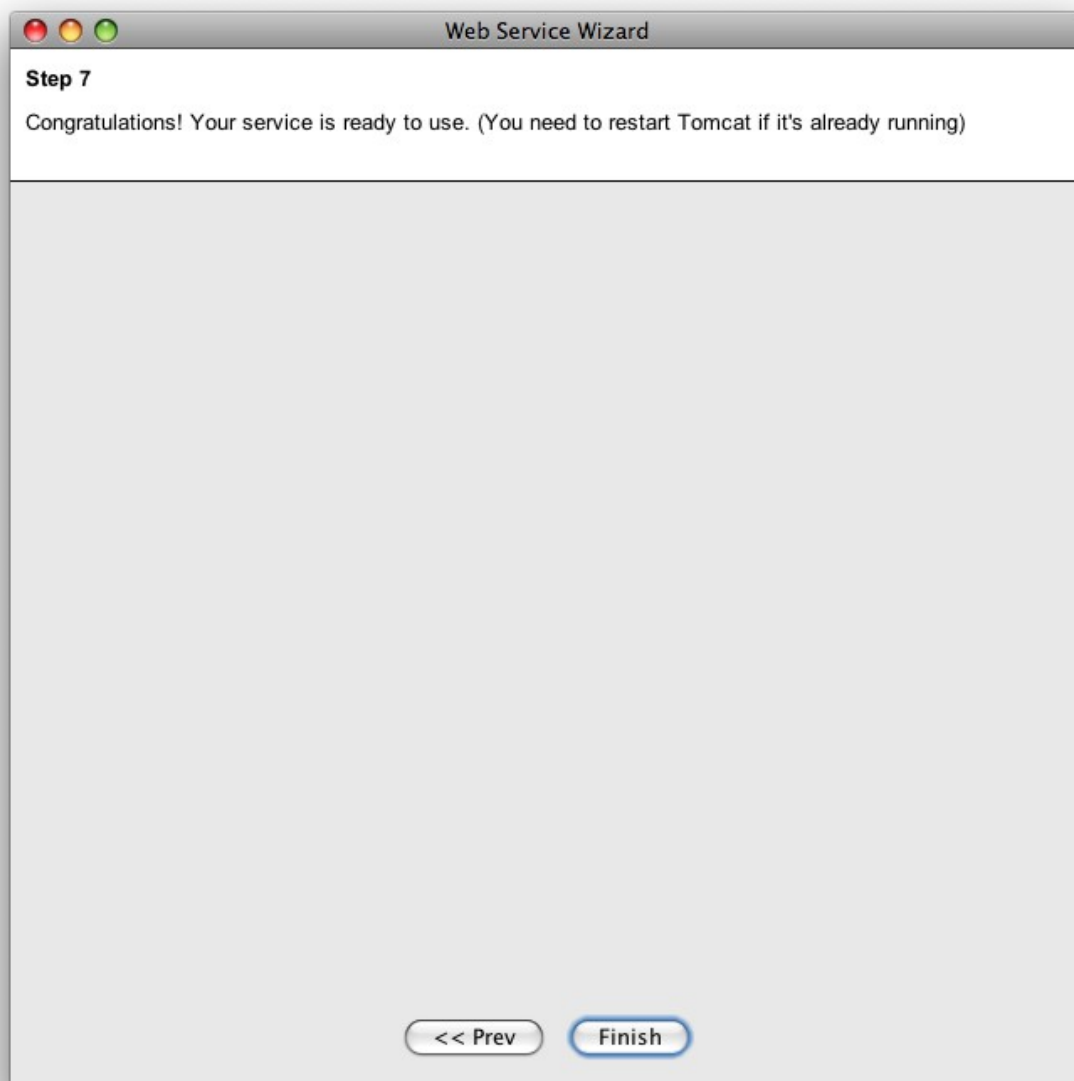


Illustration 19: Step 7

Using an IDE to develop ACGT services

It is possible to use an IDE to facilitate the process of implementing an ACGT web service's business logic. All the Java files that were generated for the service can be found under:

`$WSW_PATH/sources/$WS_NAME`

where:

`$WSW_PATH` is the path to ACGT Web Service Wizard directory

`$WS_NAME` is the name of the web service

In order to use an IDE to develop services created with ACGT Web Service Wizard it is advisable to:

1. Create a new project in the chosen IDE.
2. Set the project's sources directory to `$WSW_PATH/sources/$WS_NAME`
3. Set the following as project's library directories:
 - `$TOMCAT_LOCATION/common/lib`
 - `$TOMCAT_LOCATION/webapps/acgt/WEB-INF/lib`
 - `$TOMCAT_LOCATION/webapps/acgt/WEB-INF/classes`where `$TOMCAT_LOCATION` is the location of the Tomcat instance installed by ACGT Web Service Wizard

4. Add any other libraries that the web service requires
5. Create an Ant build file that compiles the project's sources and replaces the service's jar (\$TOMCAT_LOCATION/webapps/acgt/WEB-INF/lib/\$WS_NAME.jar) with a fresh one. Note that the old jar needs to be excluded from classpath when building the new one.

Generic Oncosimulator Service – interface

Generic Oncosimulator Service exposes the following operations:

- `public String runSimulation(NamedParameter[] params)`
As its name implies, this method is responsible for starting simulations. Generic Oncosimulator Service does it by calling the *submitJob* operation of GRMS. The only argument this method takes is a GRMS job description. It contains information that is used by the resource management system in the process of executing the job (e.g. name of the application to execute, its input parameters, files to stage in and out, etc.). Rather than constructing a GRMS job description from scratch each time *runSimulation* is invoked, Generic Oncosimulator Service uses templates that are customized basing on the values passed in the *params* array. This approach allows the service to easily adapt to changing requirements of the users. For example if more parameters need to be passed to the Oncosimulator executable, the administrator needs only to update the template file. The service does not have to be rebuild and redeployed. Another important role of templates is the fact that they are used for generating Oncosimulator Proxy Service. This process will be described later in this chapter. The following is an excerpt from an exemplary template file:

```
<grmsJob appid="Oncosimulator">
  <task persistent="true" taskid="WT_Run_001319_a_unpack_sim">
    <executable count="1" type="single">
      <execfile name="tar">
        <url>file:///bin/tar</url>
      </execfile>
      <arguments>
        <value>-xzf</value>
        <value>oncosim.tar.gz</value>
        <file name="oncosim.tar.gz" type="in">
          <logicalId><![CDATA[oncosim_tar_gz_id:int]]>
            </logicalId>
          </file>
        </arguments>
      </executable>
    </task>
  ...
</grmsJob>
```

As the above fragment shows, an Oncosimulator Generic Service template slightly differs from a regular GRMS job description. The important part is the following line:

```
<logicalId><![CDATA[oncosim_tar_gz_id:int]]></logicalId>
```

When *runSimulation* is called, the service reads the template file and searches for character data blocks. These blocks contain tokens, which have the following structure:

```
[#][@attribute_name:]parameter_name:type_name
```

The square brackets are not part of a token, they indicate that the enclosed content is optional. A CDATA block containing a token starting with # is omitted in the process of creating a WSDL for Oncosimulator Proxy Service. This is useful when a parameter is required to create a job description, but will not be directly provided by the user. Another optional part of a token is *@attribute_name*. It indicates that when a job description is created from a template, an attribute with name specified by

attribute_name and a value taken from the *params* array, will be added to the parent element of the CDATA block containing such token. If there is no *@attribute_name*, then a text node is added to the parent. The next part of a token is *parameter_name*, which indicates the value in the *params* array that will be used when processing the CDATA block. The parameter name needs to be unique for the entire template. If there is no value in the *params* array corresponding to a *parameter_name*, an exception will be thrown and simulation will not be started. Finally, *type_name* indicates the data type of given parameter. It is insignificant in the creation of a job description, but plays an important role in the process of creating a WSDL for Oncosimulator Proxy Service. The following data types are supported: byte, int, long, short, boolean, String, float, double.

Oncosimulator Generic Service makes use of the GRMS notifications system, which allows for receiving information about job statuses using either web service interface or sockets. Oncosimulator Generic Service uses the latter. The service itself does not create sockets that listen for notifications each time a simulation is started. Instead, only one socket is used to handle notifications for all simulations.

The *pl.psnr.runsimulation.RunSimulationContextListener* class implements the *javax.servlet.ServletContextListener* interface, which contains methods that, assuming proper configuration, are invoked during the container's start-up and shutdown, namely *contextInitialized* and *contextDestroyed*. The first method is responsible for creating a socket that listens for notifications and starting a thread that handles the incoming information. The second method stops the said thread and closes the socket when the server shuts down. In order to enable the *RunSimulationContextListener*, changes have to be made to the ACGT webapp configuration. The following excerpt from ACGT web.xml file shows the required modification (in bold):

```
<web-app>
  <display-name>WSRF Container Servlet</display-name>
  <listener>
    <listener-class>
      pl.psnr.runsimulation.RunSimulationContextListener
    </listener-class>
  </listener>
  ...
</web-app>
```

Notifications were used mainly for performance reasons. Instead of calling GRMS every time a call to *getJobState* occurs, the service returns the last status received for the given job.

Generic Oncosimulator Service uses DMS to create files and directories required by simulations.

The operation returns the GRMS job id of the simulation.

- `public String getJobState(String jobId)`
This operation allows for monitoring statuses of simulations started using the *runSimulation* operation. Instead of calling GRMS to determine the job's status, it relies on the notification system as described earlier.
- `public String getStateDescription(String jobId)`
This operation calls *getStateDescription* on GRMS and returns its value.

All operations of Generic Oncosimulator Service perform call to other services (GRMS, DMS) on behalf of the user by delegation of credentials. Before calling these services, Generic Oncosimulator Service checks if the user is authorized to perform the requested operations. It is done by calling *getAuthorizationDecision* on GAS. This means that privileges regarding said services must be explicitly granted to the user by the GAS administrator.

Generic Oncosimulator Service - configuration

The service can be configured using the file \$TOMCAT_LOCATION/webapps/acgt/WEB-INF/classes/config.properties. It is a regular properties file.

Following are supported properties and their meanings:

runsim.service.cert.path – path to file containing the certificate to be used by the service

runsim.service.key.path – path to file containing the key to be used by the service

runsim.service.gas.url – URL of a GAS instance

runsim.service.grms.url – URL of a GRMS instance

runsim.service.gas.dn – the distinguished name of the GAS instance identified by
runsim.service.gas.url

runsim.service.grms.dn - the distinguished name of the GRMS instance identified by
runsim.service.grms.url

runsim.service.template.file – path to simulation template file

runsim.service.notification.port – number of port used for receiving job status notifications

Proxy Oncosimulator Web Service implementation

Configuring the container

Proxy Oncosimulator Service uses the same container and libraries stack as Generic Oncosimulator Service, however, some changes needed to me made to the server's configuration to ensure compatibility with ACGT workflow engine. Following are the aforementioned modifications:

- A connector allowing for unencrypted connections was added in `$TOMCAT_LOCATION/conf/server.xml`.

```
<Server port="8006" shutdown="SHUTDOWN">
  ...
  <Service name="Catalina">
    <Connector acceptCount="100" connectionTimeout="20000"
      disableUploadTimeout="true" enableLookups="false"
      maxHttpHeaderSize="8192" maxSpareThreads="75"
      maxThreads="150" minSpareThreads="25" port="8666"/>
    ...
  </Service>
</Server>
```
- x509-based security was disabled for Globus Java-WS-Core as the following excerpt from `$TOMCAT_LOCATION/webapps/acgt/WEB-INF/etc/globus_wsrf_core/server-config.wsdd` shows:

```
<deployment name="defaultServerConfig" ...>
  <globalConfiguration>
    ...
    <!--
      <parameter name="containerSecDesc"
        value="etc/globus_wsrf_core/global_security_descriptor.xml"/>
    -->
    ...
  </globalConfiguration>
  ...
</deployment>
```

It is also possible to set-up Tomcat from scratch, deploy Axis and add all required libraries. This approach would be preferred for production environments.

Proxy Oncosimulator Service – interface

Proxy Oncosimulator Service implements the following operations:

- `public String runSimulation(String enactmentId,...)` invokes `runSimulation` of Generic Oncosimulator Service and returns its result
- `public boolean hasFinished(String enactmentId,String jobId)` checks if the job identified by `jobId` has finished by calling `getJobState` of Generic Oncosimulator Service
- `public ResultsStruct getResults(String enactmentId,String jobId)` returns logical ids of files containing results of the simulation identified by `jobId`

Proxy Oncosimulator Service WSDL and code

Because the list of parameters of `runSimulation` of Generic Oncosimulator Service could vary with regard to used simulation template, the operation accepts an array of key/value pairs as its only argument. This solution is very flexible and allows for using different

simulation patterns without the necessity of recompiling and redeploying the service. Unfortunately this approach was not feasible for the proxy service because of requirements imposed by ACGT workflow engine. It implies that the proxy service needs to be modified and rebuilt should any change affecting the parameters list occur in the simulation template. An application was created to facilitate this process. It generates a WSDL file and code for the proxy service basing on:

- simulation template file, which provides the list of parameters required by simulations
- WSDL template file, which contains WSDL code that is not affected by changes in parameters list
- ProxyServiceSoapBindingImpl template file, which contains Java code that is not affected by changes in parameters list

It may occur that some parameters that are not provided by the user directly or must be processed before being passed to the generic service. In that case, relevant code needs to be entered by a programmer.

Proxy Oncosimulator Service - configuration

The service can be configured using the file `$TOMCAT_LOCATION/webapps/acgt/WEB-INF/classes/config.properties`. Following are the supported properties:

`runsimulation.service.url` – URL of Generic Oncosimulator service

`runsimulation.service.dn` – distinguished name of the service instance identified by `runsimulation.service.url`

`dms.service.url` – URL of IDataBroker service instance

`dms.service.dn` - distinguished name of the service instance identified by `dms.service.url`

References

- [1] Gridge Toolkit <http://www.gridge.org>
- [2] ACGT D4.1 Prototype and report of the ACGT GRID layer
- [3] D3.4 – The ACGT technical architecture: Final Specification

Appendix A - Abbreviations and acronyms

<i>API</i>	Application Programming Interface. The public interface provided by libraries and services.
<i>GDMS</i>	Gridge Data Management System. The grid-based file storage system that is used in ACGT.
<i>GAS</i>	Gridge Authentication Server. The authentication server that is used within ACGT
<i>GRMS</i>	Gridge Resource Management System
<i>OGSA</i>	Open Grid Services Architecture
<i>OGSA-DAI</i>	OGSA standard for Data Access and Integration. A middleware product that supports the exposure of data sources onto the grid.
<i>SVN</i>	Subversion, the version control system used within ACGT.
<i>URI</i>	Uniform Resource Identifier. A string of characters that identifies or names an object on the Internet. It is a generalisation of URL.
<i>URL</i>	Uniform Resource Locator. A type of URI that specifies where a resource is available, and the mechanism for retrieving it.
<i>XML</i>	Extensible Markup Language. The format that is used by web services to exchange data.
<i>JMS</i>	Java Message Service