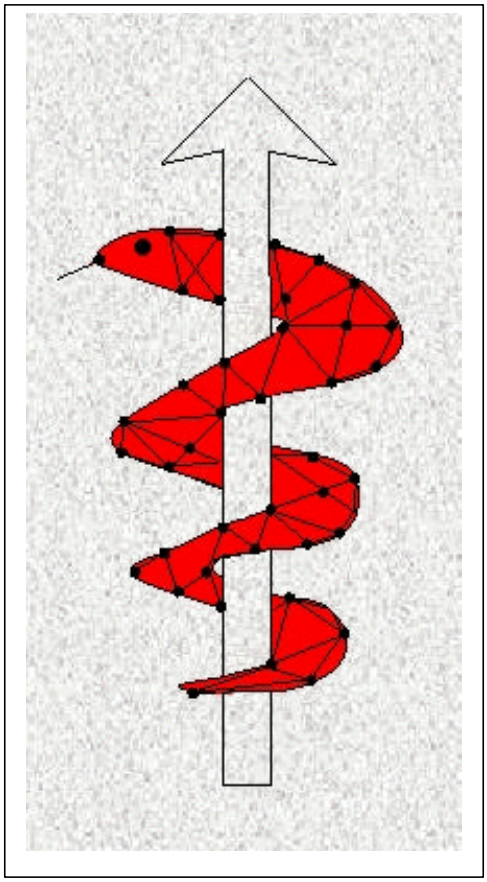
The IST Programme
Project No. 10378

SimBio

SimBio - A Generic Environment for Bio-numerical Simulation

http://www.simbio.de



Deliverable 6a
Component Interaction Report

Status:	Final
Version:	1.0
Security:	Public
Responsible:	ESI
Authoring Partners:	ESI and NEC

Release History

Version	Date
0.1	15.09.2000
0.2	22.09.2000
0.3	26.09.2000
1.0	29.09.2000

The SimBio Consortium:
NEC Europe Ltd. – UK
A.N.T. Software – The Netherlands
K.U. Leuven R&D – Belgium
ESI Group – France
Smith & Nephew - UK
MPI of Cognitive Neuroscience – Germany
Biomagnetisches Zentrum Jena – Germany
CNRS-DR18 – France
Sheffield University – UK

Contents

Introduction

This document describes the initial design report.

The purpose of this package is to allow all SimBio codes (and also in the future: external codes) to interact between each other inside the SimBio environment.
The interoperability of the environment's components will be realised using a portable, object-oriented, interoperable architecture, such as CORBA. (We will also examine the possibility of using standard Web technologies for implementing the SimBio environment.)

The purpose of this document is:
- to specify the intended functionality of the Component Interaction Module,
- to describe organisation between all SimBio components and
- to define how this functionality will be available through the SimBio environment,

We will also discuss the technical choices with respect to implementing the SimBio environment and attempt to illuminate the advantages and disadvantages of architectures based upon scripting languages, CORBA or standard Web technologies.

The SimBio Project

The central objective of the SimBio project is the improvement of clinical and medical practices by the use of numerical simulation for bio-medical problems.
The project builds on significant expertise and prior developments for specific applications to construct a generic environment, running on parallel and distributed computing systems, capable of handling a range of important problems relevant to the target community of clinical and medical service providers.
This innovative development is an enabling technology for advanced clinical practice and health care leading to improvements in: non-invasive prognosis and diagnosis, pre-operative planning, design and implantation of prostheses and postoperative verification and evaluation of treatment success. The application evaluations in the project will demonstrate the effectiveness of the SimBio environment and thus accelerate the take-up of this IT technology within the medical area.

The Component Interaction Work Package Definition

The goal of this Component Interaction Module is to organise interaction between all SimBio components. The object-oriented implementation of the SimBio environment will allow to simply sending requests to the SimBio server that will organise a solution by calling the appropriate components.

The component-based SimBio environment will offer the ability of distributed, heterogeneous execution of its components. The Common Object Request Broker Architecture (CORBA) is the most appropriate client/server middleware for the object-oriented implementation of the SimBio system.
CORBA provides a high flexibility and enables the user to optimally exploit the computer resources being at his disposal. A CORBA object bus defines the "shape" of the SimBio components that reside within the bus and also fixes how these components inter-operate. Beyond interoperability CORBA specifies an extensive set of bus-related services for creating and deleting objects, accessing them by name, externalising their states and defining ad hoc relationships between them. To make use of CORBA for SimBio an interface has to be specified for each component. These specifications are written in the Interface Definition Language (IDL) and define a component's boundary, i.e. a "contractual" interface with potential clients.

A particular example of the need for interacting components arises with the simulation of a prosthetic implant, where close interactive links between the visualisation package, the material database, an external prosthesis database and the biomechanical solvers are required. The portability of CORBA will allow the SimBio-internal and SimBio-to-external interactions to take place in a seamless manner. The platform-independence of this approach is of decisive advantage in the clinical field as hospitals are often equipped with very heterogeneous computer clusters.

However, before beginning to work with CORBA, we have decided to explore other possibilities such as Web Tools or scripting languages. After the description of the required functionality, the alternative solutions, including their advantages and disadvantages, will be discussed.

Some Definitions

1. ***SimBio Environment***: set of all applications, tools, products, etc. developed inside the SimBio project or used by SimBio applications.
2. ***SimBio Application***: tools developed or used inside the SimBio project such as the mesh generator, solver and so on.
3. ***SimBio Component***: each application and its interface (GUI, CORBA, Web Access…) developed inside the SimBio Environment; each component provides a list of services inside the SimBio Environment.
4. ***SimBio Service***: functionality provided by a SimBio component such as mesh generation, mesh visualisation, cropping, etc, for example.

The SimBio Environment

Functional Requirements

Generic

The SimBio Environment must be completely generic, i.e., open to any application. Adding new components must be as easy as possible. If the SimBio Environment is fixed (frozen) at the end of the project, it will be a failure. If one end-user wants to replace a SimBio component with his or her own component, it should be possible to do it in an easy way. In fact, if one application could perform a function, we could also imagine that another application could do the same work and the end-user does not need know which one is used; however, he or she could also choose the application that they want to do the given work.
Of course, we cannot say that adding new components will be automatic but we have to find an easy solution for such an action, which is generic and simple.

Heterogeneity

The SimBio components come from many different partners, are written with different languages (Fortran, C, C++, etc) and runs on heterogeneous platform but they have to work together inside the SimBio Environment.
Of course, only weak coupling between applications is expected; for instance: output from one application used as input for another one. We cannot imagine a complete integration of all SimBio applications in a complete GUI.
But if some applications work with a scripting language, we could imagine a way to pilot it from another application with a simple GUI.

SimBio components could be located in different places on different platform (Unix boxes, Wintel systems, etc.) however, the end-user should not need to know the location of the application he wants to use. On a Unix machine we must be able to call application from a Windows machine for instance and vice-versa.

Security / Authentication in Distributed Environments

This requirement is one of the most important of the SimBio Environment. By the fact that all SimBio components could be located on many sites, it is very important to check authorisation before running any one component. We could not imagine letting all people in the world run applications on NEC computers for instance or let someone use a segmentation program if he or she has no right to do that.
We must not forget that this software environment is developed for usage in the "medical world" with genuine patient data, thus, we have to address issues like general security, e.g., firewalls, and access restriction, e.g., which users have access to certain data sets.

User Interaction

What we expect from the end-user is for them to select the component (not the application but the generic name such as mesh generation) they want to use. Of course, once they have done that, a user interface with which to pilot the global SimBio Environment should not be too complicated. Nevertheless if one particular component needs a lot of input we must define the corresponding graphical interface or use the existing one if possible.
Moreover, the SimBio environment is a multi-user environment so we also have to be careful with concurrency issues.

File Format

An important issue to allow our environment to be as generic as possible and allow inter-operability between each SimBio component is the exchange file format.

The SimBio Consortium agreed to use the Vista file format because of its machine independence, versatility, efficiency and extensibility. In addition, three of the SimBio partners had a significant level of experience in using the Vista format.

Of course, applications should respect the data formats of their counterparts. If one application writes a file using its own format (*format-out*) and wants to inter-operate with another application, which is not able to read this kind of file but only its own format (*format-in*), we have the two following solutions:

1. Develop two translators, one from *format-out* to Vista and one from Vista to *format-in.*
2. Develop only one translator from *format-out* to *format-in.*

It seems evident that the first solution, which is longer to develop, is the closest to the generic requirement. By using this method we allow these two components to interact with each other but above all to interact with the complete SimBio environment and with other future applications.

Data Reuse

Data reuse is another important topic for this Component Interaction Module and also for the entire SimBio project.

The output from any application may be used multiple times, hence, if this output needs more than a few seconds to generate, it would be better to save it in some type of structured format for later reuse. Furthermore, for clinical purposes it is imperative that the intermediate output be saved for use in reconstructing and overcoming possible problems.

In this case, any application running will check if the corresponding job has not been done yet and if yes only get the result and send it to the user, otherwise the operation could be done normally. This checking is only of interest if the operation takes a long time to operate.
One of the tools which has to be developed inside the SimBio environment is the Database Management tool which needs to be able to inter-operate with the entire SimBio environment.

Non Functional Requirements

User Interface

By the fact that only weak coupling between applications are expected, we have two possibilities for the user interface:

1) A GUI to drive the complete SimBio environment. This solution has only one real advantage: the fact that we have only one GUI to pilot the complete SimBio environment. But there is also one big inconvenience: the fact that it is too rigid and above all no longer generic. If we want to add a new tool in the SimBio Environment, it is necessary to work again on the GUI. The number of tools in the SimBio environment is expected to increase therefore a single GUI will become poorly structured and overly complex.

2) Pass the control to the corresponding GUI of the given SimBio tool. This solution has a very important advantage, namely that it is generic. In this case adding a new tool with one adapted GUI will be very easy. The main disadvantage comes from the fact than we could have many different GUIs in term of implementation for instance (one with MFC™, one with QT™, one with X11/xwindows™) and the user could be confused by so many different designs of GUI. In this case, every application is responsible for its own GUI, the SimBio module will only connect application to application through their own interface. In this case, the SimBio environment could connect applications together through their own interface if necessary but could also call the application directly if no interface was needed.

As the main requirement of the SimBio environment is for it to be generic, we will probably choose the second solution for a CORBA based system. In this case, the end-user could also create the GUI he wants for a particular tool of the SimBio module. It could also choose the first solution and createits own global GUI depending on its need to pilot the complete SimBio environment. (Note: in a system based upon Web technologies this problem does not arise because a GUI can be defined on the fly using XML.)

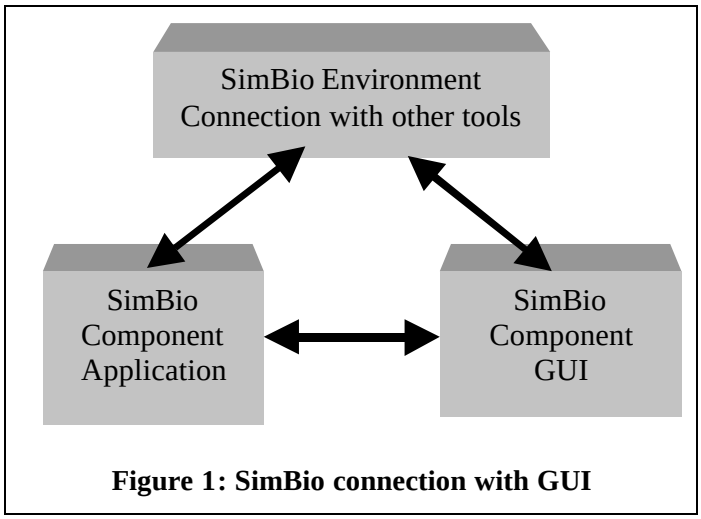


Figure 1: SimBio connection with GUI

Performance

From our point of view it is important to mention that the amount of data that has to be transferred from one machine to another which could degrade the performance significantly. Thus, compact file formats are really desirable.

Constraints

Before beginning to work on this SimBio Environment module we have to clarify the following points:

- ✓ Not all applications that we want to inter-operate have their source code available, which means we have to find a solution to create some kind of encapsulation above the executable.
- ✓ The SimBio applications work on very different platforms, from Unix boxes to Wintel systems.
- ✓ Not all applications run at the same site. They are located at many different sites (on remote computers) not close to each other.
- ✓ Not all applications have a GUI so we have to execute the application directly or create a simple wrapper GUI.

Scenarios

Many scenarios could be envisaged and the end-user will have the possibility to imagine as many scenarios as he or she wants. However, to clarify the SimBio environment somewhat, we will define two scenarios and will try to see how each will work inside the SimBio environment.

First of all, we will consider a simplified SimBio environment made of five applications:

1) Raw data,
2) Image Segmentation software,
3) Mesh Generator,
4) Numerical Simulation engine,
5) Visualisation software.

In the following sections we make references to these scenarios.

Case 1: A Simple scenario

The user has already generated their mesh before calculation and wants to visualise it before running their simulation. So the scenario will be:

1) Translate the mesh file to Vista format.
2) Send Vista data to the Visualisation tool.

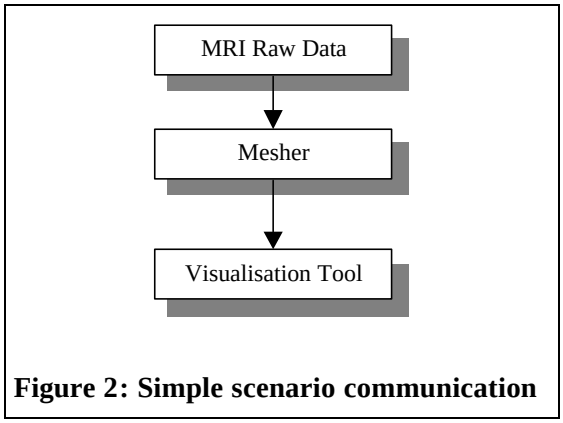


Figure 2: Simple scenario communication

Case 2: A More Ambitious Scenario

An ultimate post-project end-user scenario in the case of meniscus replacement might look like the following:

1. The patient's knees are scanned at some scanning facility and the data is inserted into the SimBio environment.

2. At the clinic of the patient's doctor, which may be in the same building, or across the street, or (as is often the case in Germany) across town, a doctor's assistant uses SimBio to access the scanned data and prepare a mesh of the patient's knee.

3. The doctor's assistant then selects a standard mesh for the meniscus prosthesis and instructs SimBio to prepare a simulation of the knee movement.

4. SimBio sends the data to a computing centre where the simulation is performed. Once it is completed, the doctor's clinic is notified of the results.

5. The doctor then examines the simulation results and makes any recommendations to his or her assistant regarding changes to the selection of prosthesis design.

While this scenario may sound far-fetched today, it is in the realm of what is possible within the next decade. In addition to the requirements already mentioned, this scenario yields one further point which need to be taken into consideration: Namely, not all users will be experts in the field of bio-numerical simulations, or Ph.D. level scientists, or even computer experts, therefore the SimBio environment must be easy to use.

Web based Approach

Outline of Proposal

From the discussion in the previous sections it is evidently useful –at least from the end-users viewpoint– to think of the SimBio environment as consisting of ***Resources*** and ***Services***. The resources in this case being the data files along with the applications which use and produce the files, while the services are the actions the user wants to accomplish, e.g., converting MRI scan data into a finite element mesh. The Web based approach is predicated upon the twin concepts of managing the resources while providing the user with a range of services.

It is important to understand the distinction between services and applications. A service *uses* applications to accomplish its goals, but there need not be a one-to-one mapping between the two. A single service may use more than one application. Likewise a single application may provide more than one service, with switching between the two controlled by some input parameters. From the end-users viewpoint it is completely immaterial as to how the services accomplish their goal, so long as they does so correctly and efficiently.

The basic idea behind the Web-based approach is to extend the functionality of the Web server to include the services needed by the SimBio environment. One common approach to extending a web server is to use the language-independent CGI (Common Gateway Interface) approach. This works by adding a number of small programs to a special directory reserved by the server and accessing these programs via the HTTP GET and POST methods. However there are a number of drawbacks to using standard CGIs for anything more than short, simple operations:

- Each request is answered in a separate process thereby introducing overhead each time the process is started and stopped.
- Since there is no persistent state, sharing memory between requests can only be done via the use of cookies and temporary files.

These problems can be overcome by creating a separate application server, which is continually running and then using the CGI interface to simply forward requests from the Web server to the application server. If we take this approach to its logical extreme then we can attach the application servers to a separate port and do away with the Web server entirely. The main drawback to such a course of action is that it introduces additional load on the system. Furthermore, since we are using the HTTP protocol for communications, there is little point in reproducing a commodity product like an HTTP based server when off-the-shelf-components provide all the necessary functionality.

As an alternative, some Web servers offer extensions similar to Apache's Module construct, which essentially performs the forwarding automatically by dynamically integrating the application server into the Web server. Although this is a step in the right direction, it obviously suffers from platform dependencies. Java, on the other hand, offers a platform independent solution to the above problems through its Servlet API. All major providers of Web servers now support the Servlet-API.

The primary reservation against using Java is its performance. Computationally intensive programs written in C/C++ can be as much as two orders of magnitude faster than Java on RISC workstations, although on Pentium systems, the C/C++ advantage is much smaller. For the SimBio environment, all of the computationally intensive calculations are performed in the applications themselves. The environments only job is to manage the data transfer from application *A* to application *B* after application *A* has finished its job. In other words, the applications in SimBio are not tightly coupled in any sense; rather they form a linear sequence of actions. Hence, the performance penalty for using Java instead of C/C++ is not expected to significantly impact on the overall system performance, thus it is out weighed by Java's provision of a simple, portable solution to the problems associated with a standard CGI interface.

Even though we would suggest implementing the SimBio environment in Java, all communications should proceed via the standard HTTP protocol. This would enable us to maintain a generic interface to applications written in other programming languages.

Figure 1 depicts the proposed architecture for each site of a distributed web-based system. A complete SimBio environment would consist of several sites; each dedicated to one or more applications. In an ideal world, all sites would be peers, i.e., able to act as both a client and a server; however, the growing use of firewalls presents us with some problems. Generally, in order for a firewall to maintain its integrity, each host behind the firewall is restricted to being *either* a client *or* a server when viewed from outside the firewall. Since, we want the SimBio environment to be firewall friendly; we have to restrict it accordingly. Each site must announce whether or not it is capable of acting as client or a server or both. Those which cannot act as clients must reject any request requiring actions only a client can make.

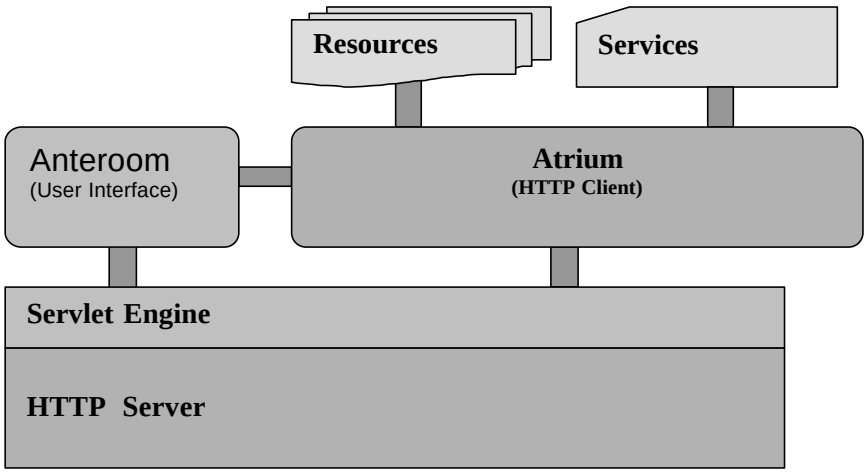


Figure 3: Proposed architecture for a single site of the SimBio environment.

The centre of each site is the Atrium, which is activated by HTTP GET and POST requests. It is important to emphasise that neither the HTTP server nor the Servlet Engine has direct access to SimBio's resources and services. These are stored in offline directories lying outside the normal server search paths. The only route to the resources and services from outside the host computer passes through the Atrium. Furthermore, the Servlet Engine may not directly send any Java code or other resources residing in the Atrium or Anteroom to the HTTP server, rather it may only send the output of any actions taken in these areas. Now, the preferred route to the Atrium is via the Anteroom in which the User Interface is located. The Anteroom is directly accessible to the Servlet-extension, and provides the end-user with a simple interface for performing all allowable tasks.

Another point to note is that the Atrium, while sitting on the server side, can also act as an HTTP client in order to retrieve resources it might need to perform a given service. Even though this is a laudable trait, it is, as mentioned above, often times curtailed by energetic system administrators who build ever "higher" firewalls. It remains to be seen how useful this property will be.
The SimBio sites themselves are collected together into an **environment**. An environment in this sense being a collection of shared resources and services. At any one time there may be more than a single SimBio environment on the Internet. Consider for example two hospitals each with their own environment, but having CCRLE as a common site. In this case it is essential that the CCRLE site be able to distinguish requests coming from one environment from those coming from the other and take the necessary actions to ensure that resources are not illegally transferred from one environment to the other.

The Anteroom

The Anteroom contains the user interface which we have chosen to implement using Java Server Pages (JSP). JSP technology provides a convenient and powerful mechanism for creating dynamic web pages. Furthermore, since they are Java based one has access to the full range of classes and objects available to any normal Java application.

The user interface itself is designed to help the end-user accomplish his or her objectives as efficiently as possible. Upon entering the Anteroom, the user is greeted with a page inviting him or her to select a language to be used for that session. (Internationalisation is well supported in Java and, for an international project like SimBio, should be included from the beginning.) After selecting the preferred language, the user is presented with a login page. Here the user will be asked to submit his or her user-ID and a password. In order to allow SimBio users to use different sites, each user-ID is a fully qualified URI, e.g., //*host:port/*simbio***/username***. When logging in to his or her home site only the ***username*** is needed. When logging into other sites, the user must provide his or her fully qualified user-ID. The hosting site should then contact the user's home site to verify the password.

Furthermore, at sites participating in more than one environment, the user must select from the list of environments, the one he or she wants to log into. Once logged in, the user's activities are tracked using standard session tracking mechanisms. In particular, the Atrium performs checks to makes sure that the user does not access any resources or services to which he or she is not entitled.

After successfully passing the login procedure, the user should be taken to the main page. It should consist of two frames, one on the left-hand side listing the various tasks the user can perform and a main frame where the output from the tasks will be displayed. The tasks should be grouped under the headings: "Resources" and "Services". Under "Resources", users may perform such tasks as uploading resources to the SimBio environment, deleting resource they own, searching for resources and downloading resources to which they have access permission for local processing. Under "Services" the user will also be able to search for services and request that a selected service be performed.

The services themselves should be divided into three categories. The first category consists of **basic** services which take a single data file as input and produce a single data file as output. These types of services would perform basic operations like translating between data formats. The second category consists of **standard** services which require simple input parameters from the user. For example, a service which numerically simulates knee motion would need to know how many seconds of real time should be simulated. The third category consists of **advanced** services which require expert knowledge on part of the user in order to be used correctly. In the knee simulation example, this might include inputting damping factors which the user knows will improve numerical stability without adversely effecting the results given the particular simulation conditions.

When the user requests a service, the Atrium reads the description of the services from a special *services.xml* file located in the environments home directory. The entry for each service contains information about what input, if any, is required from the user. After reading the description, the Atrium instructs the user interface to dynamically create an input form through which the user can enter the needed information.

Once a service has finished the user should be notified so that he or she can retrieve the output. Notification can occur in one of three ways. In most cases one can simply use persistent connections as defined in HTTP 1.1. (Most older browsers and servers have already implemented persistent connections using an unofficial extension to HTTP 1.0.) For longer running jobs one can use the *refresh* header to instruct the browser to reload the results page after a fixed time interval. For jobs needing hours or days to complete, one can resort to e-mail for notifying the user that the results are ready.

The notification should inform the user the name of the output file. The user can then download the file or use it for input to another service request. If the user chooses to download it, it should be returned with the "*Content-Type*" header set. The user can then configure his or her browser to handle the returned data. For example, if the output is in the SimBio-Vista format, then it should be sent back to the user's browser with the "*Content-Type*" header set to: *"application/x-simbio-vista*". The user can then instruct his or her browser to handle this content type by using the SimBio visualisation program, or any other software capable of handling this file format. As an alternative, the user may instruct the browser to simply save the file to a local disk. (Saving the file to disk is the *only* alternative for applications incapable of accepting command line options or standard input.)

The Atrium

As noted above the Atrium is activated using HTTP GET and POST requests. (Note: Although one can use either GET or POST, it is generally better to use POST because GET's idempotent property means that browsers are free to cache the results.) The syntax of a legal request is:

http://host:port/simbio/atrium?query-string

The format of a valid SimBio *query-string* is actually a tunnelled KQML message. (KQML stands for Knowledge Query and Manipulation Language) Normally the end user does not need to worry about specifying the correct query string because the user-interface takes care of this. If an external application want to communicate directly, then the query-string should take the form:

query-string: performative="performative"&"attribute"="..."&…

The *performative* is the task the user would like to have performed in the Atrium. The standard attributes available to all performatives include:

reply-with:	*a message id which the receiver should use for replies*
in-reply-to:	*an id referring to a previous message*
sender:	*the message's sender*
environment:	*the sender's SimBio environment.*
contractor:	*the person for whom this message is being sent*
cascade:	*a toggle indicating whether or not the performative should be passed on to all authorized hosts.*
content:	*the information over which the performative acts*
language:	*the language used in the content*
ontology:	*the ontology used by the language in the content*

The *language* attribute defaults to *XML* and the *ontology* attribute defaults to *SimBio.* Support for other languages and ontologies can be added later. (Note: not every performative need use all attributes.) The user performatives supported at present include:

ask-if:	*Ask if a performative is recognised.*
reply:	*Communicates a reply.*
tell:	*Communicates some information.*
ask-resource:	*List all resources matching the description.*
achieve-resource:	*Perform some action on the resource.*
ask-service:	*List all services matching the description.*
achieve-service:	*Perform the requested service.*
ask-user:	*Ask if specified user is known to the environment.*
achieve-user	*Perform some operation on the user account.*

The list of recognisable performatives is easily extended and we fully expect it to be extended during the course of this project. In addition to the standard attributes listed above some performatives define additional attributes for convenience purposes, i.e., they help the Atrium avoid having to parse the content. A complete description of all the convenience attributes will be given in the final implementation report. Note: All strings must be URL encoded before being added to the POST request otherwise the server is likely to reject the request before it ever reaches the Atrium.

The issuer of a POST request should always wait for a reply. Each request may return the response "*denied=reason*". If possible the issuer should try to correct the reason for the denial and submit a new request or if this is not possible, inform the user of the reason for the denial.

Deployment

The Web application described herein can be deployed using a relatively simple directory structure without having files dispersed throughout the system. Figure 2 depicts the basic layout needed for SimBio.

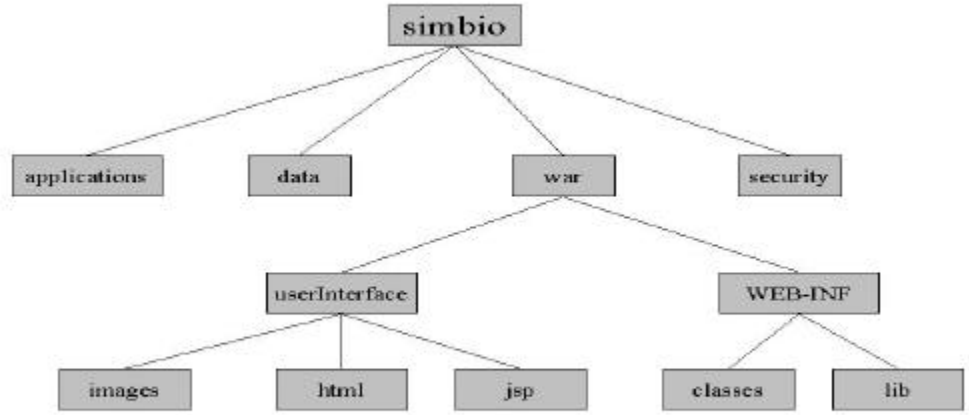


Figure 4: Basic Layout of SimBio

The *war* directory must be registered at the servlet-engine. The *WEB-INF* contains the servlets themselves along with any additional libraries. In SimBio the servlets handle the performative requests mentioned above. The servlet-engine is forbidden from serving any files from *WEB-INF* directly, instead it con only return the results of any servlet output. The servlet-engine may return the pages found in the *userInterface* directly to the end-user. For convenience we divide the *userInterface* directory into three subdirectories, thereby separating the static content (*images*, *html*) from the dynamic content (*jsp*).

The *applications* directory contains the applications themselves or shell scripts for starting the applications, the *data* directory contains all the data files, and the *security* directory contains security related files such as the password files, key stores, etc. If this site is part of more than two environments, then each of these directories should have one subdirectory for each environment, wherein all files related to that environment are placed.

In the *simbio* directory itself there are two site dependent files: *services.xml* and *environment.xml*. The latter contains a listing of SimBio environments to which the given site belongs, along with all the other sites belonging to the same environment. The idea here being that when there is more than one SimBio environment on the Internet, sites must declare to which environment or environments they belong. As an example, two hospitals might set up separate SimBio environments having NEC's CCRLE lab as common site. The site at CCRLE must take care not to transfer any sensitive data from one environment to the other.

The *services.xml* file contains a description of all the services offered by each of the different environments to which this site belongs, along with a key indicating which environment offers which service. As mentioned above, the service file also contains information about any input parameters that need to be set by the user.
The third site-specific file sits in the directory *WEB-INF* and is called *web.xml*. This file describes the site to the servlet-engine and contains, among other things, an adjustable parameter giving the site specific location of the *simbio* home directory. It also contains a few initialisation parameters related to localisation issues.

Security

Before discussing the types of security measures we can implement it is worth-while discussing what information we would like to protect, the types of threats we face and the costs/benefits of different security measures. Ideally we world we would like to be able to provide a security system capable of offering all of the following features:

1. Prevent unauthorised persons from matching patient names, insurance numbers, etc. to a diagnostic image.
2. Prevent unauthorised persons from identifying a patient on the basis of scan data alone.
3. Restrict authorised users access to internal resources or services, especially resources belonging to other users.
4. Prevent interception of and tampering with internal and external communications.
5. Prevent unauthorised access to the SimBio environment.

Item 1. is very important because one of the novel features of SimBio: is the ability to work with real patient data in a clinical setting. Under such conditions, it is imperative to keep the patient's personal data secure. Items like the patient's name, health insurance number, etc., should be stripped away *before* the data enters the SimBio environment. Instead all data imported in the environment should be marked only with an encrypted patient ID. It is outside the scope of SimBio to perform these duties.

Item 2. presents a far more difficult problem. If a patient can be identified from the scan data alone, then that data has to stored using a cryptographically strong encryption scheme. The data should only be decrypted by special helper applications when it is needed. Furthermore, the decryption applications should pipe their output directly into the application requiring the data, which should then pipe its output directly into an encryption application. Only those applications capable of reading from the standard input and writing to the standard output can be used by data needing this level of security. If this level of security is needed, SimBio can easily support it, however, users must be aware that encrypting and decrypting large amounts of data is a time consuming process.

Item 3. involves several subtleties due to the distributed nature of the SimBio environment. The requirements envision an environment with sites scattered throughout the Internet. Under such circumstances it is unlikely that it will be feasible, let alone desirable to have a single system administrator for the entire environment. Therefore, each site will have to take responsibility for administering itself. Each site administrator will assign users a locally unique username and password. In order to avoid naming conflicts, the locally unique usernames will be combined with the site's URI to create a globally unique user-ID of the form: *//host:port/*simbio/***username***. Whenever a user adds a resource to the environment it will be marked as being owned by that user. Furthermore, each user can set *read* and *write* permissions for each resource they own in order to allow other users access to these resources. The default permissions should not permit any user except the owner to access the resource.

In addition to setting access permissions for resources, they should also be set for the services. For basic and standard services the default access should allow all users to use them. Access to advanced services should probably be restricted to those users who have a demonstrated understanding of how the applications involved in carrying out the services really work. The premise here being that the basic and standard services should be more or less bullet proof, i.e., they should not exhibit any unexpected behaviour if the user enters in inappropriate data. The advanced services on the other hand may crash if given improper data (hopefully without taking the entire site down with them!).

Item 4. is perhaps the most intractable security related issue because building a network with secure communications is an extremely difficult task, made even more onerous by a mesh of conflicting international laws and regulations. In the realm of communications, poor security is worse than none at all, because the false sense of security provided by an inadequate system may lull people into complacency, thereby encouraging them to communicate information they would not normally communicate over a channel they knew to be insecure. Take for example, the standard login procedure described above: The user enters his or her name and password data into a simple HTML form and sends it unsecured over the network. Many people might recoil with horror at the thought of submitting passwords unsecured, yet this is the same procedure used by the ordinary *telnet* or *ftp* programs still in use by most organisations around the world.

How insecure are network communications? Communications between a host A and host B can be intercepted and tampered with at host A's network, host B's network, at any router between the two (in particular at the internet service provider of either one). Eavesdroppers can also read any communications carried over traditional media, i.e., copper wire, fibre optic cable or microwaves, used by any of the participants. As for as most hackers are concerned, the weakest links in this chain are the networks of host A and B – and indeed this is where most of the high profile break-ins occur. Monitoring your communications as it passes through numerous routers between the client and the server requires software akin to the FBI's *Carnivore* system. Although those providing routing services should be security conscious enough to prevent hackers from installing such software on their system, past experience has shown this is unfortunately not the case. The packet sniffing software that forms the basis for *Carnivore* is readily available on the internet and its use by professional hackers should not be underestimated.

If a hacker were to break into a network hosting a SimBio site, then he or she could monitor all SimBio communications. In this way the hacker can hope to capture passwords to SimBio accounts, tamper with requests sent to the server or even hijack a session started by a legitimate users.

Is there anything one can do to prevent eavesdropping and tampering? A standard Web technology for communicating information securely is SSL (Secure Socket Layer) developed by Netscape. In principle, SSL provides a high level of security – in practice, political decisions have made it about as useful as a child's security blanket. Most people around the world use Web browsers from Netscape or Microsoft, both of which were produced in the US and therefore fell under US arms export restrictions. Previously, US law did not permit the export of any type of public key cryptography system using keys containing more than 40 bits. However, a message encoded with a 40-bit key can be deciphered by a hacker with the appropriate (freely available) software and special (inexpensive) hardware in real time, i.e., within a few seconds. This is fast enough to enable a hacker to execute the so-called man-in-the-middle attack without either party noticing. Hence, the browsers do not provide an adequate level of security rather they simply lull the user into complacency. (The only reason on-line shopping is relatively secure is that credit card fraud is easily detected and aggressively prosecuted.)

As of January 2000, the US has amended its laws to allow the export of browsers with 56-bit standard encryption and 128-bit enhanced encryption technology to specific countries. 56-bit keys remove the threat of on-line attacks (at least for the next few years), but the keys can still be brute-forced with a few days of computing. If the attacker records your communications then cracks the key off-line, he or she would then have access to any passwords other critical information which you may have transmitted. In order to install HTTP Servers with 128-bit encryption technology to connect to the new browsers, the servers need 128 bit certificates recognisable by these browsers. At the moment only financial organizations and subsidiaries of US corporations are being issued such certificates.

Is it worth the effort to set-up secure communication channels? If SimBio were only to be used in a research setting, the answer is: probably not. Knowing what services a researcher is requesting, or intermediate data he or she requires would not appear to be of any particular interest to an outside observer. In a clinical setting, however, even this level of leakage may be intolerable. (If shortly after

a patient visits a clinic, the doctor starts performing simulations of brain tumour removal, then, rightly or wrongly, one might draw a connection between the two.)

Item 5. is the last, but not the least of the items on our security list. If someone were to hack into a SimBio site, then none of the precautions mentioned above would be of much use. Furthermore, until now we have assumed that anyone trying to break-in was doing so for a purpose, i.e., he or she wanted access to resources or services without proper authorisation, but there are many purely malicious hackers, who are only trying to create havoc.

Denial-of-service (DOS) attacks, for example, have been in the news recently. Fortunately, even though they are a major inconvenience, they do not compromise any of SimBio's resources, therefore, they do not need be explicitly countered by the SimBio environment. Most DOS attacks could be easily prevented if routers were outfitted with ingress or egress filters to prevent IP-number spoofing. This would expose the DOS attacker's real IP address, making it relatively simple to stop the attack. The installation of such filters has been hampered because some internet service providers claim there are legitimate uses for IP-number spoofing, e.g., in mobile communications.

If a malicious hacker were to obtain the password of a SimBio user he or she could delete all the user's resources and attempt to attack the forms used for starting the services by inserting certain control character sequences. System backups are the only defence against the former attack while the lattercan be prevented by inspecting all form input for legitimacy, i.e., a number is really a number or a file name is the name of a real file.

In the end, site security itself is one of the biggest problems to overcome. Any SimBio environment will only be as secure as its least secure site. If the sites themselves do not provide adequate protection against malicious intrusion by third-parties, then there is little that SimBio can add in the way of additional security. — What good is SSL if a hacker has broken into your computer and is reading your every key stroke? Why encrypt your data if it is available to knowledgeable users on the system swap pages while the applications using the data are running? Only the local system administrator can prevent these types of attacks.

Development Plan

The development of a Web-based SimBio environment should proceed in the following stages:

1. Construct the basic infrastructure for processing performatives. One servlet should check all the incoming requests for correctness and conformity with the active security policy. It should then forward the requested performative to another servlet. (By handling each performative in a different servlet we make it easier to extend the system.)

2. Integrate an XML parser into the environment for handling XML files. Define the format of the files: *services.xml*, *environment.xml*, *users.xml* and *index.xml*. Also Map out a simple strategy for supporting internationalisation on the user interface.

3. Implement the secure login procedure along with the security infrastructure needed to ensure all users are properly logged on and that their login session has not expired.

4. Create the user interface and performatives necessary for users to work with resources. (Up-load, delete, search and list, download, edit properties) This includes adding security checks to make sure users only access those resources they are authorised to.

5. Define a set of basic and standard services. Create the user interface and performatives necessary for users to use the basic and standard services. The services have to be wary of their client/server role.

6. Construct a management interface along with the necessary performatives for site administrators to add or delete users, manage resources and set permissions for using local services.

7. Create the user interface and performatives necessary for users to use advanced services.

8. Construct an interface to allow the user to manage his or her account. (E.g., change password, e-mail address for job notification, etc.)

9. Implement an infrastructure for enabling the user to have all data held in an encrypted form until needed by an application.

If we use the usual numbering system, *Major_version.minor_version.patch_level*, and start with 1 as the first minor version, then we see that version 0.5.0 would be the first version available for beta testing. According to the work project work plan, this version must be finished by PM 12. The final version should be finished by PM24.

CORBA based approach

This section describes in detail the design of the generic SimBio environment using CORBA as a middleware software layer. Besides the technical aspects of the realisation we address the important issues of security and user interaction that play a central role in this area. With respect to the technical implementation we base our design proposal on the experiences gained during the development of a small prototype that was comprised of a subset of the SimBio tools. Even though we did not cover the complete tool set, the prototype sufficiently reflects the conditions and requirements of the full version and does not hide mission critical aspects behind a curtain.
Taking into consideration the time needed for building the prototype, we conclude that we can implement the generic environment within the time frame specified by the technical annex (TA).

Introduction

In the area of distributed computing the so-called Common Object Request Broker Architecture (CORBA) belongs to the most important and powerful approaches. It was defined by the Object Management Group (OMG) which started as a consortium of 8 different vendors among them Digital, SunSoft and Hewlett Packard. Their mission statement was the definition of interoperability between applications in distributed heterogeneous environments using object-oriented technology. Currently, the OMG consists of more than 800 members ranging from software and hardware vendors, to research institutions and to individual software developers. Due to the complexity of the CORBA approach, we give in the following sections a brief overview of concepts and services provided by CORBA constituting central parts of SimBio's generic environment implementation.

Object-Model in CORBA

One of CORBAs central concepts is the interface description of objects. For this purpose the OMG provides an interface description language (IDL). With the help of this language developers can define the actual interface of the object, i.e., which operations are legal on the object, what parameters do these operations take, which attributes does the object possess, and which exceptions will the object throw if errors occur. Apart from the description of the object, the IDL is not used for the actual realisation of the object functionality. Conventional languages like C++, Java, Smalltalk and so on will do this. The IDL definitions are mapped to those languages by an IDL compiler. Language independent definitions are translated into language specific type definitions and application programming interfaces (API). In the final step, the developer uses these types and APIs to provide application functionality and to interact with the ORB.

Object Request Broker

Within a distributed system the Object Request Broker (ORB) enables a client to send a request to a particular object implementation. The ORB is responsible for all actions necessary to finding the appropriate server, preparing the client for receiving the output data from the server, and executing the actual communication. From the client's point of view, information on the location of the server object, its target language and the underlying operating system, have no impact on the actual request. On server side the Object Adapter (OA) delegates a request to its corresponding object. This object executes the method with the delivered arguments (if any) and returns the output data (if any) that have to be sent back to the client. In case of an error, an exception is thrown. Although there might be no explicit data exchange between client and server, the request could still change the state of the object. As figure 6 indicates, the ORB is comprised of several components: the dynamic invocation interface (DII), the dynamic skeleton interface (DSI), the OA and the ORB kernel. Stubs and skeleton are generated from the IDLs and added to the system. The ORB distinguishes between static and dynamic method calls. In the simple static case the operation is already completely described during compilation. Thus, the corresponding stub can be generated. Alternatively, the DII allows the construction of a request during run-time. The required information is here taken out of the IR, which

can be seen as a database or a special service for retrieving data on interfaces during run-time. Semantically both method calls are identical. On the other side, constructing dynamic method calls is quite expensive and should only be used if the static variant cannot be applied.

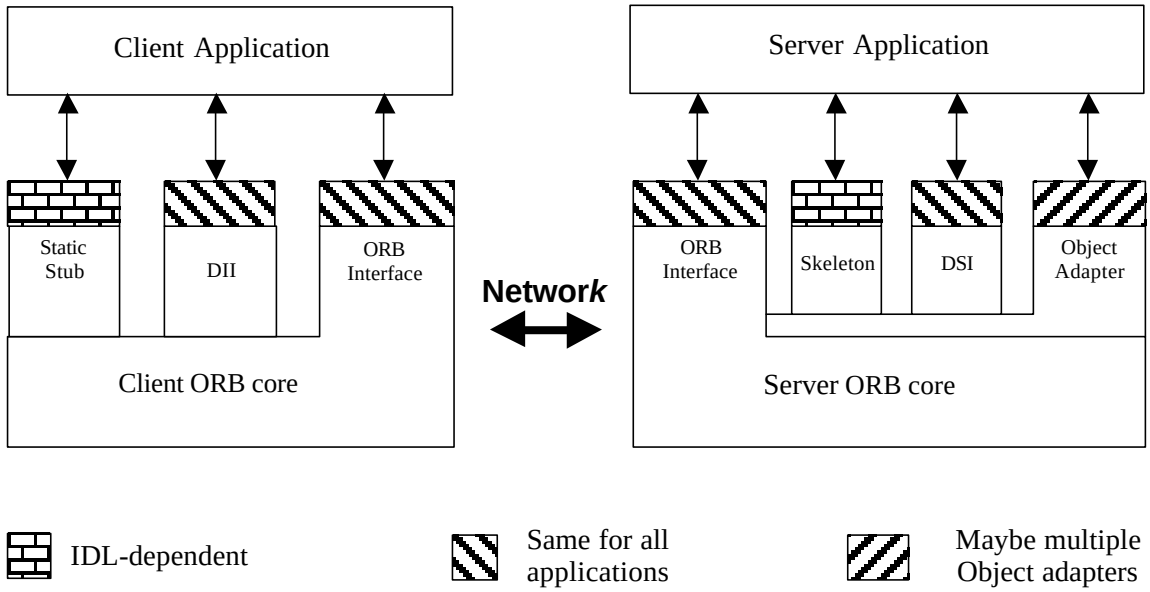


Figure 6: Common Object Request Broker (CORBA)

Object Adapter

The OA is responsible on the one side for the activation of objects and the dispatching of requests to their respective methods and on the other side for the generation and interpretation of object references. Additionally it offers objects the possibility of accessing the wide variety of services provided by the ORB. In this context CORBA distinguishes two variants of adapters: the basic object adapter (BOA) and the portable object adapter (POA). Whereas the BOA was specified in order to provide a minimal set of functionality mandatory for all ORB implementations, the POA offers a full suite of services and advanced features intended to allow developers to write scalable, high-performance server applications.

Services

A standardisation process of so-called CORBAservices accompanied the development and standardisation of the CORBA architecture. These services build a set of interfaces that are either implemented by the objects or inherited from other interfaces. From the developers' point of view the access to these services is realised as if the ORB would provide them, although CORBAservices are positioned out of the particular ORB in the CORBA architecture (ref. Figure 7). Important members of this class for the SimBio environment are the Naming Service and the Security Service.

Naming Service

The Naming Service is the most basic of the standardised CORBA services. Here, objects are registered with unique names, which can later be used to resolve their associated object references. The idea is similar to the Internet Domain Name Service (DNS), which translates Internet domain names into the actual IP-addresses. In general the Naming Service provides the following advantages to the client:

- Clients do not have to deal with stringified object references. Instead they can use meaningful names for object.
- While changing the value of the reference that is bound under a particular name, it is possible to re-direct clients to other implementations without having to change the source code.

- Applications can elegantly get access to initial references of a required service. Employing the Naming Service takes away the burden of storing stringified object references in files.

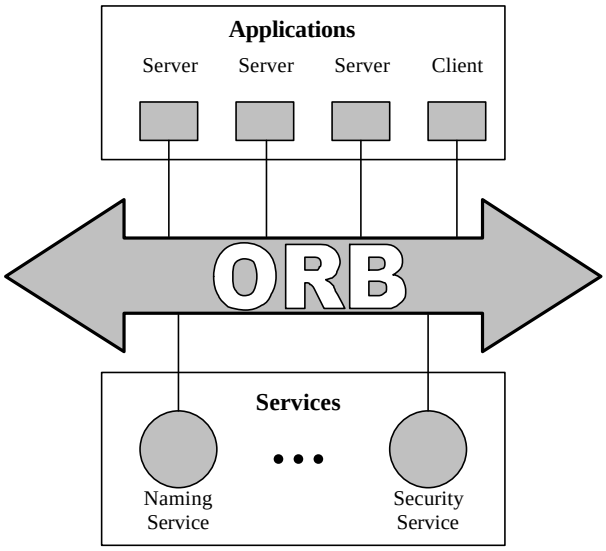


Figure 7: CORBA Services

Security Service

In general the Security Service allows implementations to provide protection against the following threats in a distributed object system:

- Allow users to access only that information they are authorised to see.
- Restricting the delegation of user rights
- Security controls being bypassed
- Eavesdropping on a communication line
- Tampering with communication between objects
- Lack of accountability

Even though conformance to the above mentioned issues will add a considerable amount of security to an implementation, there are still facilities lacking to counter other breaches caused by different kinds of attacks.
Due to the complexity of the security related specification, it is divided into several packages that cover different topics. This allows implementers of ORBs to successively evolve this service. However, before an ORB can claim to be secure, it has to provide a well defined range of packages. Some of the most important issues for the SimBio project are addressed by the Common Secure Interoperability (CSI) Feature package. Here, three levels of secure interoperability are specified which can be used in distributed secure CORBA compliant object systems where client and target objects may run on different ORBs and different operating systems. At all levels authentication between client and target as well as protection of messages for integrity and confidentiality are supported.
CSI level 0: The identity of the initiating user is transmitted to the target. There is no delegation possible to other objects on further invocations.
CSI level 1: Here delegation is possible. This provides a mean for intermediate objects to represent the actual user.
CSI level 2: Additional attributes and privileges are passed to the target. Furthermore composite delegation is supported, thus the attributes of more than one user can be transmitted.

Currently we are employing the free publicly available ORBacus implementation from OOC (Object Oriented Concepts). As the security service they are providing the FREE_SSL plug-in (128 Bit encryption). This guarantees CSI level 0.

Other CORBA Services

CORBA provides a long list of services. However, in the initial phase of the project there is no need to employ services other than Naming and Security. On the other hand, if the generic environment evolves in time it might become necessary to integrate other CORBA services as well. A list of some suitable candidates follows:

- The Concurrency Control Service is intended to co-ordinate clients, which are supposed to access the same resources. For example, conflicts arising due to multiple accesses of the same file could be avoided.
- The Event Service allows components on the bus to dynamically register or unregister their interest in specific events. In SimBio it could be employed to communicate with graphical application using asynchronous events.
- The Licensing Service provides operations for metering the use of components to ensure a compensation for their use. It supports charging per session, per instance creation, and per site.
- The Trading Service is an extension of the Naming Service. Here, servers have the possibility to attach properties to their exported services, which can be later used as a selection criterion. In a later stage of the SimBio environment more that one tool might be available for a specific task. This service could decide from the specification given by the client, which tool is best suited to fulfil a request. Maybe we can say that this service offers “Yellow Pages” for objects.

Schematic Overview of SimBio

The ordinary scenario of the generic environment is depicted in the following figure. A client application runs on a local user machine and requests SimBio services residing on remote architectures. The user machine is connected to the SimBio sites via the CORBA bus.

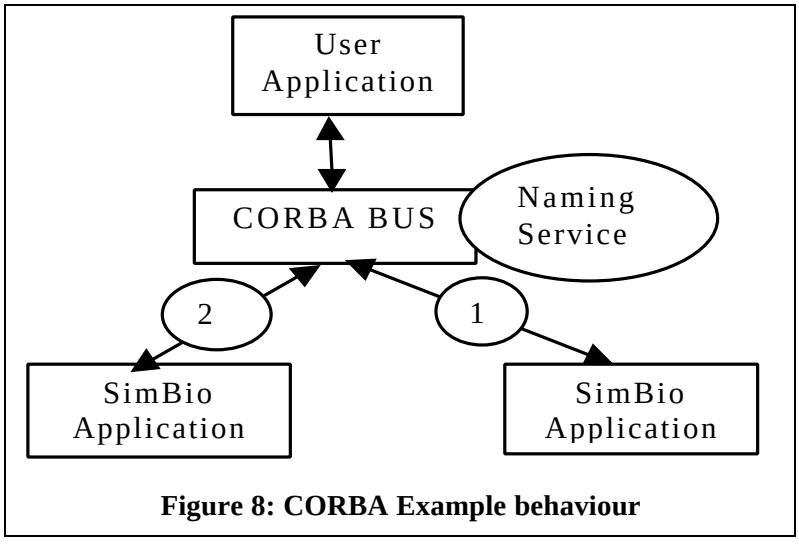


Figure 8: CORBA Example behaviour

It is important to note that the connection provided by the CORBA bus is not limited to user and SimBio site only. Additionally, all SimBio sites are linked with each other allowing a flexible exchange of information between all components.

Application Manager Definition

Because we do not have the sources of all SimBio codes available, the “Functional Schema of CORBA” has to be adapted accordingly. Thus, instead of developing a CORBA binding for each application, we decided to implement CORBA server objects that act as simple wrappers for the

corresponding SimBio applications. We call such a target object an Application Manager (AM). The main advantages of this approach are that we do not have to change the application itself and the time needed for writing an AM is comparable small. From the functional point of view we can define a generic AM (see listing I.) that is able to drive both already known applications and future ones. Here we agree on a set of functions, e.g., Start, Stop, GetResult, etc. , which is sufficient for controlling an application. If it turns out that additional routines are required, they can be easily integrated.

Name:	**Generic SimBio**
Syntax:	Genericcool <parameters as string>

IDL Definition (Prototype):	#ifndef SIMBIOGEN_IDL
	#define SIMBIOGEN_IDL
	
	// **
	// * CORBA IDL FOR SIMBIO GENERIC INTERFACE (ORBACUS V3.3.1) *
	// *---*
	// * Last Update : 01.09.2000 *
	// *---*
	// * (C) by ESI SA, Rungis, France *
	// *---*
	// * Contact : Christophe Janvrais / cj@esi.fr *
	// **
	
	interface SimBioGeneric
	{
	void Start(in string inputParameters, in string inputfile, in string outputFile, out short rtn);
	void JobPause(out short rtn);
	void JobStop(out short rtn);
	void GetResult(out string resline, out short rtn);
	void GetResultFile(out string outputFile, out short rtn);
	void ReStart(out short rtn);
	};
	
	#endif

Listing 1: IDL for Generic AM

SimBio Setup

Naming Service

Before a user application is able to access the required services, all servers have to register with a particular naming service running on a dedicated machine the project partners agree upon. Due to the fact that the corresponding (naming) object is persistent, it is sufficient to distribute the IOR to the different sites only once, since the IOR remains identical on every new server invocation. If a server will no longer provide services to the environment, it will explicitly de-register itself from the naming service. Obviously, the servers have to follow a particular naming convention to allow clients an unique identification of the target objects.

Client/Server Dependencies

A user application (client) traverses the list of all available names and retrieves the corresponding reference for the appropriate target object. In those cases, where a SimBio application needs certain input from another SimBio application, it will contact the naming service in order to get access to the remote service. For example, if a user requests a mesh generation of certain segmentation data, the meshing server has to request this file from the segmentation server, if it is not already (maybe from a previous request) locally available. On the other side, the segmentation server requires the corresponding raw data as input for its computation. Similar to the meshing server, it has to request this data from another server, i.e., the raw data server. Thus a single user request causes a dynamic chain of client/server dependencies. Dynamic in this context means, that due to the storage and/or removal of intermediate results the number of dependencies may vary for the same user request.

Security Impact

Even though the security related issues do not play a central role in the initial phase of the project, it should be mentioned that the currently employed CORBA implementation is **not suitable** for handling the above mentioned client/server chain. The restriction to CSI level 0 does not cover the delegation of user permissions to intermediate objects (see section "Security Service). However, since commercial CORBA security services are available already, which support CSI level 1, this problem can be considered as not critical.

Performance

From the development of a CORBA prototype that is comprised of three components, we learned that one major performance factor is determined by efficiently handling the binary data exchange. Since the sizes of the Vista based files are significantly larger than 1 Mbyte, this could easily become the bottleneck of the distributed computation. After some tuning of the corresponding code sections we could prove that the bandwidth on basis of IIOP is practically identical to the *ftp* counterpart. Thus, we can claim to avoid introducing further overhead here. Obviously, performance gains can be made whenever re-usage of generated data is possible. The drawback is, however, additional memory consumption for storing intermediate results. In a (future) commercial environment topics like structure and maintenance of those local databases have to be addressed carefully.

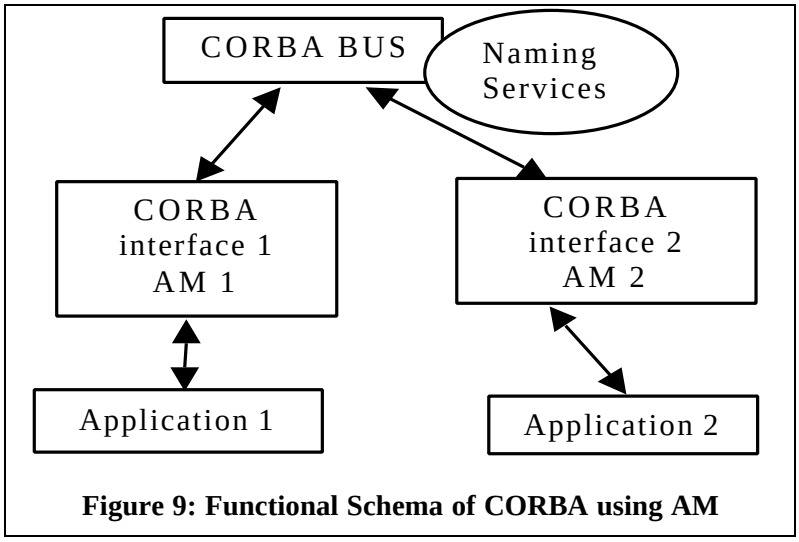


Figure 9: Functional Schema of CORBA using AM

Native CORBA Binding

An alternative to the above described AM method would be the provision of an actual CORBA binding for each SimBio application. Even though this is the most elegant solution for the object oriented distributed environment, the major drawback is the required re-coding effort in order to make each application CORBA compliant. The lack of source code availability of some codes is another limiting factor here. However, the CORBA approach allows us to migrate one application after the other to the CORBA world, if the previously mentioned problems are solved. Thus, the CORBA approach itself can be considered as dynamically evolving.

Integration of Pam-Safe

To be able to communicate with other SimBio applications ESI decided to define the following functionality:

- Print Information about the current cycle (only available at the end of the current cycle).
- Plot the current Geometry (only available at the end of the current cycle).
- Create a restart File.
- Quit (no restart file is created).
- Stop (a restart file is created).

Definition of the PAM-SAFEÔ CORBA Object

```
#ifndef PAMJOB_IDL
#define PAMJOB_IDL
// ********************************************************************************
// *           CORBA IDL FOR SIMBIO PAM-SAFE (ORBACUS V3.3.1)            *
// *--------------------------------------------------------------------*
// *  Last Update : 01.09.2000                                          *
// *--------------------------------------------------------------------*
// * (C) by ESI SA, Rungis, France                                       *
// *--------------------------------------------------------------------*
// *  Contact : Christophe Janvrais / cj@esi.fr                          *
// ********************************************************************************

typedef sequence<float>   floatseq;
typedef sequence<long>    longseq;
typedef sequence<string>  stringseq;

// Prnt
struct PamPrnt {
  long   cycle;          // number of the current cycle
  float  time;           // time at the end of the current cycle
  float  time_step;      // time step for the current cycle
  long   type_elem;      // type of element where this time_step occurred
  long   num_elem;       // number of the element where this time_step occurred
  float  dtmin_fac;      // element reduction factor
  long   dtmin_nb;       // number of element adjusted due to the dt_min factor
  float  kine_enrg;      // current kinetic energy
  float  inter_enrg;     // current intern energy
  float  init_enrg;      // initial total energy
  float  tot_enrg;       // current total energy
  float  current_speed;  // current execution speed (in ms/element/cycle)
  float  remain_time;    // estimation (in s) of the remaining execution time
  float  ad_mass;
};

// PamVar
struct PamVar {          // This structure used in input contains the flags
  longseq VarGlobal;     // to get the variables. The order is set according
  longseq VarNode ;      // to the GetTitles results. (0 <= FALSE, other = TRUE)
  longseq VarSolid;      // In this structure in output are stored the number of entity wrote per variables.
  longseq VarBeam;       // -> for Beam, Bar, Spotweld, etc... (1D Elements)
  longseq VarTool;
  longseq VarShell;      // -> for Shell, Membrane
};

// State
struct PamState {
  float    time;
  floatseq coords;
  floatseq var;
};

// Exception
exception PamJobException {  // Exceptions return a number and a string to explain the
  long   error_id;           // type of error
  string description;
};

interface PamJob
{
// Get Initial Geometry
  void      cGetParams(out long cNumNodes,
                       out long cNumBeams,
                       out long cNumShells,
                       out long cNumTools,
                       out long cNumSolids) raises (PamJobException);
  long      cGetNumMaterials()             raises (PamJobException);
```

```
  floatseq  cGetCoordsInitState()          raises (PamJobException);

  longseq   cGetMatLabels()               raises (PamJobException);
  longseq   cGetNodeLabels()              raises (PamJobException);
  longseq   cGetBeamLabels()              raises (PamJobException);
  longseq   cGetShellLabels()             raises (PamJobException);
  longseq   cGetToolLabels()              raises (PamJobException);
  longseq   cGetSolidLabels()             raises (PamJobException);

  longseq   cGetBeams()                   raises (PamJobException);
  longseq   cGetShells()                  raises (PamJobException);
  longseq   cGetTools()                   raises (PamJobException);
  longseq   cGetSolids()                  raises (PamJobException);

  stringseq cGetTitles(in char cType)      raises (PamJobException);
  // cGetTitles return the names of the variable available in the DSY file for the type cType

// Get Dsy States
  long      cGetNumofDsyStates()          raises (PamJobException);
  long      cGetNumCurrentDsyState();
  PamState  cGet1DsyState(inout PamVar cVar, in boolean cCoor, in long cState)
                                  raises (PamJobException);
  // cGet1DsyState return the DSY state number 'cState' if exists with
  // variable (if asked) and coordinates (if cCoor is TRUE)

// Job Management (Current State)
  void      cPrnt(in boolean cWait_for, inout PamPrnt cPrnt)
                                  raises (PamJobException);
  // cPrnt fills the PamPrnt structure according to the current state.
  //
  //  in : cWait_for = TRUE  -> wait for the end of a cycle (5-10 s on a Origin 200)
  //                 = FALSE -> do not wait for the end of the cycle
  //                         and return the status of the last asked cycle
  //                         or the last status writted in the output file (stdout)
  //                         (In this case current speed and remain_time are null).

  void      cSendQuit()                   raises (PamJobException);
  // cSendQuit send a signal to terminate the job

  void      cSendDump()                   raises (PamJobException);
  // cSendDump asked to the job to write a restart file at the end of the job

  void      cSendStop()                   raises (PamJobException);
  // cSendStop send a signal to terminate the job and to write a restart file.

  PamState  cGetPlot (inout PamVar cVar)    raises (PamJobException);
  // cGetPlot returns the current state with variables (if asked : cVar != NIL or cVar not empty)
  // this function wait for the end of the current cycle.
};
#endif
```

Development Plan

The development of a CORBA based SimBio environment should proceed in the following stages:

1. Development of Application Manager for SimBio tool supporting basic functionality only.
2. Integration of advanced features to the Application Managers. Some applications require more specific functionality for control.
3. Incorporation of the available security package to the implementation.
4. Deriving the definition of a Generic Application Manager. Could be used for SimBio components and above all for future external codes.
5. Optional: Provision of a GUI for selected SimBio components.
6. Optional: Provision of a global GUI that users could employ to visualize the status of available services.
7. Optional: Provision of a global help menu. User will get in depth information on both the underlying principle and user interaction of each component.

Technical Choices

The entire concept presented here is based on the assumption that at every participating site of the generic environment, we are allowed to run the CORBA software.

In order to minimise portability problems resulting from the usage of different ORBs we should agree on a single public domain implementation. Good candidates are ORBacus and MICO. Both products support a wide variety of UNIX-systems as well as Windows NT. The authors of this deliverable are currently using the ORBacus implementation.

Summary

In this deliverable we have investigated two different approaches for implementing the component interactions within the SimBio environment and for allowing the user to control his or her environment via a graphical user interface. The first approach was based upon the use of standard Web technologies, while the second approach was CORBA based.

CORBA based Approach

Advantages

With its roots stretching back more than a decade, CORBA is a mature technology, familiar to many object-oriented software developers. Its architecture has been thoroughly tested and proven to be very reliable. There is also a large body of commercial software one can tap to extend the functionality of a CORBA based system.

For SimBio, CORBA offers the advantage of being able to move to a tight coupling mode in the future. At present, the applications are only loosely coupled through file exchange. This decision was taken because the individual applications need tens to hundreds of minutes of wall-clock time on present computer systems to complete their respective tasks. With advances in computer hardware, the execution time for these applications might be reduced at some point in the future to just a few seconds. In that case, a CORBA based system would be in the position of providing a tight coupling via method invocation.

Disadvantages

CORBA was developed for object-oriented, distributed computing in an Intranet environment. The first ORBs could not even intercommunicate because the designers saw no need for more than one ORB on a company's Intranet. Although CORBA has since been extended for use on the Internet, it is by no means comfortable in a Web environment as evidence by its use of its own communication protocol, namely IIOP instead of the web standard, HTTP. This has the affect of making it difficult to access remote resources via a standard URL in a transparent manner. (For the original SimBio applications one can design around this problem with little loss in flexibility).

Also, as noted above, CORBA is most effective when it is integrated into an object's source code enabling other objects to directly reference the "corbatised" object's methods. In SimBio, a tight coupling between applications is not required at present and therefore most of them will not be "corbatised", i.e., they will not implement CORBA-callable methods. Instead, it is planned to pass information from one application to another using simple file transfer. Under such a scenario, CORBA is overkill and will add unnecessary bulk to the software.

Web based Approach

Advantages

As mentioned above, a tight coupling between applications will not be required in the near future within the usage scenarios discussed at the beginning of this report. Under these circumstances, a Web-based approach using standard HTTP servers and browsers is an attractive alternative in terms of convenience and portability.

The convenience lies in providing the end-user who is not at his or her usual place of business with the ability to still reach the SimBio environment by pointing any browser at the nearest SimBio portal. The portability arises by implementing the entire environment in Java.

Disadvantages

Within a loose coupling environment, the inefficiencies of Java are not very prominent, nevertheless it must be pointed out that a Java implementation may be over an order of magnitude slower than a pure C/C++ implementation.

With the convenience of allowing users to access the system from any Internet host, comes the problem of securing such a system. As discussed in detail above, these problems are far more severe than those encountered in a CORBA based approach where it is relatively simple to control access. For this reason it is not clear that Web-based systems like the one proposed here could ever be made secure enough to find acceptance in the medical research community let alone in a clinical setting.

Combining the Two Solutions

The World Wide Web Consortium (W3C) has circulated several proposals for introducing CORBA like functionality into HTTP servers; these include WebBroker, WIDL and SOAP. To date, none of these have generated more than passing interest, hence, there is little incentive for us to invest effort in a technology which might not be supported by the time the project is completed.

On the other side, there is also the possibility of defining Interfaces for Web browsers using CORBA Applets. There are two CORBA extensions that could provide this functionality: CorbaScript and CorbaWeb. CorbaWeb is an environment that allows integration of Corba Object inside a Web browser, while CorbaScript is a scripting language dedicated to Corba that allow easy access to Corba objects defined in the Corba bus. Naturally, both of these methods suffer from the same types of security problems that plague a pure Web system.

The basic problem with combining the two approaches is one of complexity. If parts of the system are implemented in CORBA and parts using Web technology, then the combined system will consists of Web servers and CORBA servers. It is not clear whether or not such a complex software system would be possible to maintain without enormous effort.

Conclusion

As can be seen from the above discussion, both of the approaches investigated here have advantages and disadvantages. Whereby it seems that based upon our present knowledge, the Web based approach entails more security problems than a CORBA based approach. Furthermore, a Web approach using Java as the programming language is expected to show significant performance degradation compared to a C/C++ implementation of a CORBA based environment. Finally, a Web approach, by eliminating the possibility of introducing tight coupling at a future date, may unnecessarily limit the adaptability of the SimBio environment to changing hard and software configurations. Nevertheless, before making a final decision on which approach to pursue, we will first consult our project partners to be sure that there are no determining factors we might have overlooked which could play a role in our decision making processes.