IOME, A Toolkit for Distributed and Collaborative Computational Science and Engineering

M. K. Griffiths[1*], D. Savas[1], and C. C. Reyes-Aldasoro[2]

1 Corporate Information and Computing Services, The University of Sheffield, 285 Glossop Rd., Sheffield, S10 2HB, U.K.

2 Cancer Research UK Tumour Microcirculation Group, Department of Oncology, School of Medicine & Biomedical Sciences, The University of Sheffield, Beech Hill Road, Sheffield S10 2RX, U.K.

* Corresponding Author: m.griffiths@sheffield.ac.uk

Abstract

This paper presents a unique, multi-purpose toolkit, enabling researchers to easily develop modelling and analysis applications, which may be run as web services and accessed interactively. The development kit is based on a protocol that uses an XML markup called the "Interactive Object Management Environment Markup Language" (IOME ML). The paper describes the IOME ML and its development kit.

We illustrate the capabilities of IOME with a case study based on a medical image processing application (CAIMAN: CAncer IMage ANalysis), which is wrapped as a web service and accessed through a web browser offering image analysis tools for life scientists. IOME is a toolkit that enables researchers to use a Software as a Service (SAAS) model to enable access to research applications which may exploit resources provided through cloud computing.

The paper concludes with a review of further developments, including refinements to the mark up language and the development of a service factory, enabling a more scalable service provision model through the dynamic invocation of published simulations as IOME web service applications.

1. Introduction

The Internet provides a media-rich communications platform enabling communities to share content. In many cases, the flow of information is still unidirectional with one user uploading data for other users to download. While this model of information exchange has promoted a worldwide knowledge share, there are collaborative situations where a two-way flow of information is required to fully develop knowledge discovery mechanisms, this is partially addressed by

the wide range of tools used for social computing. Recent developments on workflow tools are now enabling researchers from different disciplines to collaborate in multidisciplinary problems by sharing data and running collaborative simulations through web services. However, the implementation of these web services is not always simple, especially when interconnecting between webservers and clusters. Consequently it is found that a high level of technical expertise is required to set up a complete web service application.

Researchers developing computational models would benefit considerably from development kits and tools that would enable them to provide simulations with a range of methods that facilitate collaboration. Many researchers are very good at programming for example using C/C++, Matlab or Fortran and implement algorithms related to their disciplines. These researchers may find it much more challenging to develop distributed applications. This is due to the extra knowledge of communications protocols that is required to develop distributed applications. Distributed applications development tools such as *Message Passing Interface* (MPI) or *Parallel Virtual Machine* (PVM) are popular amongst the scientific community but these can be difficult to use for developing distributed heterogeneous applications running across administrative domains. MPI and PVM belong to the same class of "Single Instruction-Multiple Data" model of computing where the computing task is shared by the coordinated distribution and execution of the same code on multiple servers. Such an approach can not lend itself well to developing heterogeneous independent applications that can communicate across administrative domains. Although PVM was developed with heterogeneity in mind, its emphasis is on the coordination of multiple workers and hence the approach addresses a different set of problems to that presented here. As such, IOME can be viewed as a set of tools for developing potentially "Multiple Instructions-Multiple Data" computing models that can span administrative boundaries. The challenging task of implementing a reliable method of communications between heterogeneous and independent applications running across different software & hardware platforms and administrative domains has been implemented via the adoption of the Simple Object Access Protocol "SOAP" standard. SOAP is used as the carrier of all IOME communication messages. This binds IOME to soap standards and allows future developments based on these standards.

By exploiting a mapping between web services and MPI, it is possible to alleviate some of the difficulties in distributed application development for many researchers . We have investigated a number of techniques researchers may employ for publishing an application as a web accessible service. These methods include the *Gridsphere Portal Development* Application Programming Interface (API) and the portal system known as the Enterprise Accessible Software Application (EASA), both methods attempt to make applications easily accessible from a web browser , . However, these systems have certain restrictions. EASA is run through intranets or users login into a specific website which acts as a gateway for all applications. EASA is very prescriptive in its

usage model and provides a set of predefined multi user community working environments and pattern of working. In contrast IOME provides a set of web service based tools enabling researchers to tailor build and customise service structures to the needs of a specific problem. For the CAIMAN model, it was realised that what was required was a single user interface for accessing a particular service. In essence, IOME provides a tool for resource aggregation by using a mashup approach to development. This paper presents one use-case scenario of IOME that is the publication of modelling and analysis software as a service.

Other diverse collaborative techniques we have investigated include the use of workflow development tools such as the *Taverna* engine or computational steering toolkits such as *Gviz* or the reality grid . Taverna provides an approach for consuming a service whereas IOME provides a method for making an application available as a service and provides the tools for developing suitable clients for a given user community. Since the IOME server is a standalone self contained web server it is easier to administer and maintain than solutions implemented using taverna/condor/AXIS. Although some web services are simple to implement e.g. using AXIS one essentially drops web service onto a container. We have also found that web service wrapping tools such as CARMEN and OPAL require significant expertise and knowledge of java accessible web services. For many of these systems there is a rich number of use cases and this leads to greater system complexity making it more difficult for users with less specialised middleware knowledge to implement a solution. IOME distinguishes itself by mediating the gap between service provision frameworks and message passing frameworks for high end computing such as PVM/MPI. IOME can be viewed as a tool for aggregating services and as such may be used for the generation of mashups.

Our experience has demonstrated that in many cases the complexity of the toolkit, the accessibility to it, and the distributed nature of the problem add to the challenge for researchers to develop a satisfactory mashup or collaborative solution in a timely fashion. Our work with a community of research computing service users indicates that there is a need for a simplified method of communication between a range of client applications running on different platforms. Using IOME, this requirement can be met by providing a general-purpose description language for modelling and analysis problems. We have named the resulting XML based mark up language IOME-ML.

This paper presents a novel, multi-purpose toolkit and protocol called the Interactive Object Management Environment (IOME). The IOME toolkit enables researchers to easily develop simulations and analysis applications that can be run as web services. The simulations can be controlled by client applications such as visualisation tools, web accessible portlets, popular web API tools and other bespoke clients. By using the IOME toolkit, researchers can develop collaborative research tools without requiring an in depth knowledge of the web

service protocols. The IOME toolkit has been developed for a range of popular development tools including Matlab, Scilab, python, php, C++ and FORTRAN. IOME provides researchers with a range of benefits by enabling:

- publication of simulation and analysis applications as web services it also provides tools for building clients,
- data sharing between heterogeneous applications and platforms, which can be running within different administrative domains,
- automated generation of metadata, which may be used for managing distributed data collections and for automated laboratory notebook generation,

The next section gives an outline of application development using IOME; Section 3 presents an application to demonstrate how the IOME toolkit is used in practice. Section 4 provides an overview of the architecture of the IOME toolkit. In the final section we conclude with a summary of our findings and we present further developments required for the IOME toolkit.

2. Application Development Using IOME Design of the IOME Toolkit

The IOME toolkit defines a general mechanism for communication between heterogeneous processes and provides a methodology for implementing "Software as a Service". It uses a client-server model of working with communications taking place using, SOAP, "simple object access protocol" messages. The IOME "Client-Server" model or more precisely the single server multiple-clients model provides the necessary structure for collaboration, simulation and data-sharing tasks that are needed to provide "software as a service".

A distributed application developed using IOME, requires at least two components. These are;

- an IOME server managing, receiving, storing and serving shared sets of data

- one or more IOME clients, querying and updating the data-store kept by the server.

IOME provides a range of methods for adding and removing data items to/from the data store. There are also operations for inspecting, modifying and listing the contents of the data store.

Figure 1 illustrates the architecture of an IOME application. An IOME client provides an application interface that communicates with the remote IOME server.

The use case considered in this paper is that of a researcher providing access to a model or knowledge processing tool through a simplified client that

communicates with the remote service. The application is made available in a "software as a service model" in which a single application instance is made available to multiple clients. A web accessible client application is used to request an analysis application on a remote service in the Cloud.

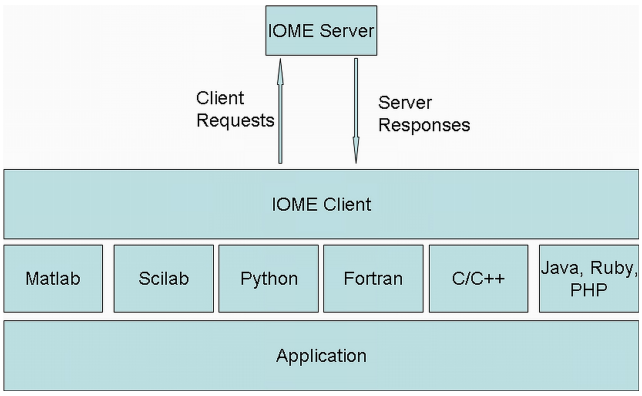


Figure 1

In figure 1, it is very important to note that the 'software' itself is also an IOME client. Thus the IOME server provides the link between the software that is being provided as a service and the clients using that service.

The researcher generates client applications using the various toolboxes that have been provided with IOME. It is then necessary to set up the application so that an IOME server may be started and populated with data. IOME provides the following two applications shown in Table 1.

Application	**Description**
`iogs`	Start the IOME server, make IOME client calls
`iogsproxy`	IOME proxy server

***Table 1. S*uite of applications provided with the IOME toolkit.**

The **iogs** command should be invoked with a number of positional parameters. If it is invoked with less than the expected number of parameters the missing trailing parameters are automatically assigned predefined default values. The first parameter to iogs always specifies the action required from IOME.

For example, initially, the command **iogs initiome** is invoked to start the IOME server so that client applications can make calls to this IOME server. When starting the server, **iogs** input parameters will include the command name, the name of the simulation, the name of a 'XSL' stylesheet file and the port to be used for communications in that order. If the port number is not specified IOME uses the first port it finds available. The XSL file provided on the command line

can be used to automatically transform output XML files into a user presentable form. If no XML file is used it has to be specified as **null**.

When the IOME server is started all the web service methods will be enabled by default. However more precise control of the web service methods can be specified via the use of a configuration file (**iogs.config**).

If the IOME server is started correctly it may subsequently be populated with data by making a sequence of IOME client add-parameter requests, alternatively we can request the server to load an IOME-ML file, the server state may be saved at any point. For IOME servers running inside a cluster we may have to use the **iogsproxy** application, which can be used to relay requests to the actual server.

The toolboxes for generating IOME client applications are listed in table 2, a brief description is also provided in the table. Each toolbox provides the same set of calls to the IOME server. Help for using the Matlab and Scilab toolboxes may be obtained from the respective help utility. Help will also be documented on the *wiki* provided by Google code for the IOME project.

Toolbox	Description
`ioMatlab`	Matlab toolbox using the Matlab web service client capability
`iopython`	python IOME client scripts using the ZSI web service library
`iophp`	php IOME client scripts using the PHP SOAP library
`ioscilab`	Scilab toolbox, which currently uses system calls to make web service client calls using the iogs application

Table 2. IOME Client toolboxes.

For each of the toolboxes there is a number of data definition and manipulation methods including;

- **addparam**
- **setparam**

- **getparam**
- **listparam**
- **deleteparam**
- r**eadsimulation**
- **readlocalsimulation**
- **writesimulation**
- **writelocalsimulation**

These methods may be applied to variables of type *double*, *int*, *string*, *mat* or *vec* (matrix and vector variables respectively), get and set methods for *mat* and *vec* types also require size information about the matrix or vector. There are also get, set, add and delete methods for metadata items. The applications on both client and server side may be used to enable parsing of locally stored IOME-ML data using a local instance of the IOME server. Each toolbox has a parser enabling it to read IOME-ML files by directly parsing data into a generic simulation structure for use by a client application.

In order to create an infrastructure for "software as a service", each toolbox also contains the following job/simulation creation, execution and monitoring utilities;

- **submitsimulation**
- **runsimulation**
- **requestsimulation**
- **runrequestedsimulation**
- **getsimulationresults**
- **simulationstatus**
- **deletesimulation**

Unlike many applications that are able to provide a container of a web service (for example Apache tomcat) the IOME server runs as a standalone web service application. For example to start the IOME server the following command can be used;

iogs initiome iogatest null 50000 localhost 50

This command indicates that IOME will initialise the server hence the ***initiome*** command is the first parameter. The name of the simulation that identifies it uniquely is supplied by the user as the second parameter "in this case **iogatest".** In this example the third parameter is specified as ***null***, this parameter is normally used to specify an XSL file which can be used to transform the IOME mark up files generated by the server into a more readable format. Parameter 4 is the port which the service runs on, in this case **50000**. Parameter 5 is the domain name of the server that is hosting the IOME server, thus **localhost** implies this local machine. The final parameter is the maximum number of simulations that the server will support. When the server starts it generates a text file containing the port on which the service has started

The IOME server can be populated with data using the readsimulation command, for example, if an environment variable **IOME_WSPORT** has been defined and assigned the port number

for example, IOME_WSPORT=5000

then;

iogs readsimulation simfile.xml 0 $IOME_WSPORT localhost

will populate the IOME server with data as defined in the IOME-ML file **simfile.xml**, which we assume is already uploaded to the host on which IOME server is running. The third parameter is **0** which is a reference identifier for a particular task instance hosted by the IOME server, this is normally **0**, **$IOME_WSPORT** contains the port for the service and parameter 5 "localhost" is the host to which the request is made. An IOME-ML file may be read from a local file on the client machine using the **readlocalsimulation** command. There are corresponding methods for writing the simulation i.e. the methods **writesimulation** and **writelocalsimulation**.

Alternatively, as shown below, the IOME server can be populated with data using the addparam method,

iogs addparam int intparam1 10 7 0 $IOME_WSPORT localhost
iogs addparam int intparam2 20 7 0 $IOME_WSPORT localhost
iogs addparam double doubleparam1 3.141 7 0 $IOME_WSPORT localhost

In the example above after specifying the addparam method the next parameter is the data type followed by the parameter name, value and an access flag set to 7. The last three parameters are the server reference identifier, port and host address. The parameters are set using the setparam command to reset intparam1 from 10 to 20 we use.

i**ogs setparam int intparam1 20 0 $IOME_WSPORT localhost**

Executing iogs using the getparam method with the correct parameter type and name returns the value.

i**ogs getparam int intparam1 0 $IOME_WSPORT localhost**

There are additional methods which may be used to make job requests, we first identify the steps required to implement a simulation as a service.

- Installation of the IOME toolkit.
- Complete testing of the user application and identification of user inputs and the required responses

- Prepare a script named iogenericsim.sh which will be executed by the IOME server when the simulation is requested by the web service client.
- Add IOME client calls to the simulation or analysis application to get the parameters from an IOME server, note this server may be different from the IOME server used to request the simulation or analysis application.
- Select a toolkit to build a client that will make the job requests and generate a template IOME-ML file that will be used to make a service request to the IOME server.
- Start the simulation server
- Test the job submission client

It is important to note, that for the scenario identified above there are 2 IOME servers participating in the lifecycle of a job submission. The first server is the main service which is active and awaiting requests to run a particular analysis or simulation. The second, optional server is local to the actual simulation which runs on a separate machine from the main job server this second local instance is used as a parser for reading and writing the contents of the IOME-ML file.

Using the iogs application, the following methods may be called which provide methods for submitting, monitoring the status and receiving job results.

iogs submitsimulation jobfile 0 $IOME_WSPORT localhost
iogs runsimulation jobfile 0 $IOME_WSPORT localhost
iogs requestsimulation jobfile 0 $IOME_WSPORT localhost
iogs runrequestedsimulation taskid $IOME_WSPORT localhost
iogs simulationstatus taskid $IOME_WSPORT localhost
iogs deletesimulation taskid $IOME_WSPORT localhost
iogs getsimulationresults taskid $IOME_WSPORT localhost

The simplest method is the **submitsimulation** method, this is used for the image processing case study the client provides an IOME-ML file containing parameters required for the service. The job is then run by the IOME web service, after completion it is deleted from the server. In this case the job will store or send results to an alternative location the client is expected to provide information enabling that data to be passed to the user making the original job request. For example in the image processing use case the user e-mail was used as one of the job parameters and this was used to e-mail results to the user.

If the **runsimulation** method is used the web service request is made in synchronous mode i.e. the web service client waits for a response from the web service, which in this case is the simulation results, this is satisfactory for a system where the task can be guaranteed to run immediately and the task is not so long that the request timeout is exceeded. This particular kind of request has been tested using the genetic algorithm tutorial example provided with the IOME toolkit, it also demonstrates that this system works well with workflow tools such as the taverna engine.

Making a job request asynchronously using the submit simulation method can make testing difficult this is due to the fact that the server deletes any job files after job completion, in this case, it is best to use the **requestsimulation** method followed by the **runrequestedsimulation** method. In this case the job files are not deleted until the **deletesimulation** method is called. When this sequence of calls is used the user client can retrieve results using the **getsimulationresults** method. For the **getsimulationresults**, **runrequestedsimulation**, **deletesimulation** and **getsimulationstatus** methods the user must provide the correct task identifier, this is returned by the **requestsimulation** method. Another method for monitoring the status of the IOME server is to run a separate process which queries the IOME server and tests status parameters which may be placed on the server, this can be used for the client applications to provide progress information to the user or information to the service administrator. Such a system was easily implemented for the case study. For each of the client toolboxes identified in table 2 the calling method for the respective web service method takes the same form as identified above for the iogs command.

The activity diagram illustrated in figure 2, explains how IOME is used to set up a generic simulation as a web service. For our case study in section 3, the image processing service results are returned to the user using an e-mail message containing the resulting image. In the more general case, results for a job may be returned by the job in response to the job submission request or using a get results request. The results will be contained within an updated IOME-ML file. In this scenario the activity diagram is shown in Fig. 2. The first lane of the activity diagram illustrates the calling sequence made by a user client application these are as follows:

- request a simulation, the IOME web service returns a unique task identifier.
- Run the requested simulation, the IOME web service responds with a status flag.
- Monitor the progress of the simulation by calling get status and
- get simulation results.

In the above steps step 2, 3 and 4 use a task identifier returned by step 1. The results are returned as an IOME-ML file in response to the get simulation results request. The client toolboxes provide functionality for parsing or writing an IOME-ML file in this way data is easily communicated between the client and the IOME web service. Using the IOME tools we have provided an approach for enabling interoperability between heterogeneous applications on different administrative domains this may require the use of secure tunnelling techniques offered by protocols such as *ssh*.

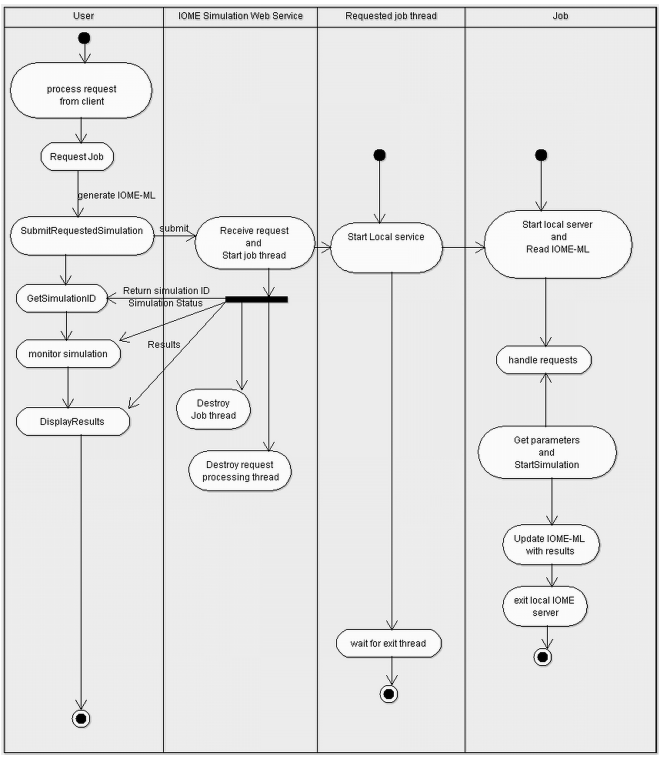


Figure 2 Approximately here

3. An on-line algorithm repository for Cancer Image Analysis

The need for image analysis is growing in many fields and cancer is not an exception. With the advent of new imaging techniques, researchers can visualise physiological and pharmacological processes together with the traditional anatomical images. Yet, once the imaging of subjects has been achieved, sometimes there is a lack of resources to properly analyse and process the data and the wealth of the information contained in images and videos remains to be extracted in the near future. There are many software tools for analysis and processing, which are not restricted to biological images: some have a basic platform to which modules are added (ImageJ, Imaris, AxioVision, Volocity, …), others are highly flexible and powerful and offer high-level programming with a wide variety of toolboxes (Matlab, Scilab,…) and also some graphically-oriented packages (Photoshop, Corel,…) are used to analyse biomedical images. Even when some of these tools are open source and freely available, the user needs to develop an expertise of the software to obtain quantitative results of the images that have been produced, and this is not always simple.

CAIMAN (CAncer IMage ANalysis) is an Image Analysis Internet-based project that combines the strength of open-source web-based scripting languages, the powerful high-level technical computing language MATLAB, and the vast

literature on image analysis and computer vision to provide a user-friendly web-page where any person can upload cancer-related images and execute analysis algorithms and obtain quantitative measurements related to their images. CAIMAN has three algorithms implemented: **measuring cellular migration** for scratch wound assays, **tracing vasculature** using scale-space ridge tracing, and **shading correction** based on a signal envelope estimation retrospective algorithm.

The front-end of CAIMAN is a PHP-based website where the user can access the current image analysis algorithms. The user will select the appropriate webpage and algorithm to apply to the images at the front end and will proceed to upload the image (Fig. 3). The actual components participating during the submission of an image processing task are shown in Fig. 4, it can be seen that these components are the;

- user desktop,
- web server hosting the client used to access the CAIMAN services
- the compute server host awaiting requests sent by the user via the web server
- the compute server worker, which runs the actual processing task and e-mails the result to the user.

The components communicate via the Internet, which may be accessed through an Ethernet or the dedicated network.

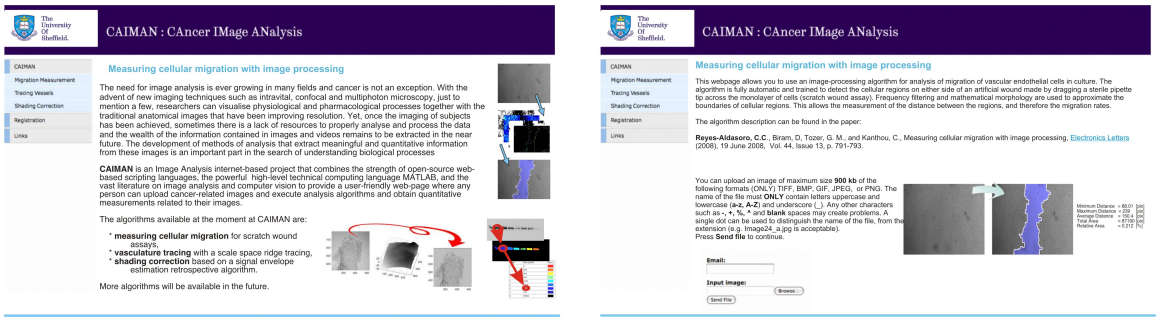


Figure 3 Approximately here

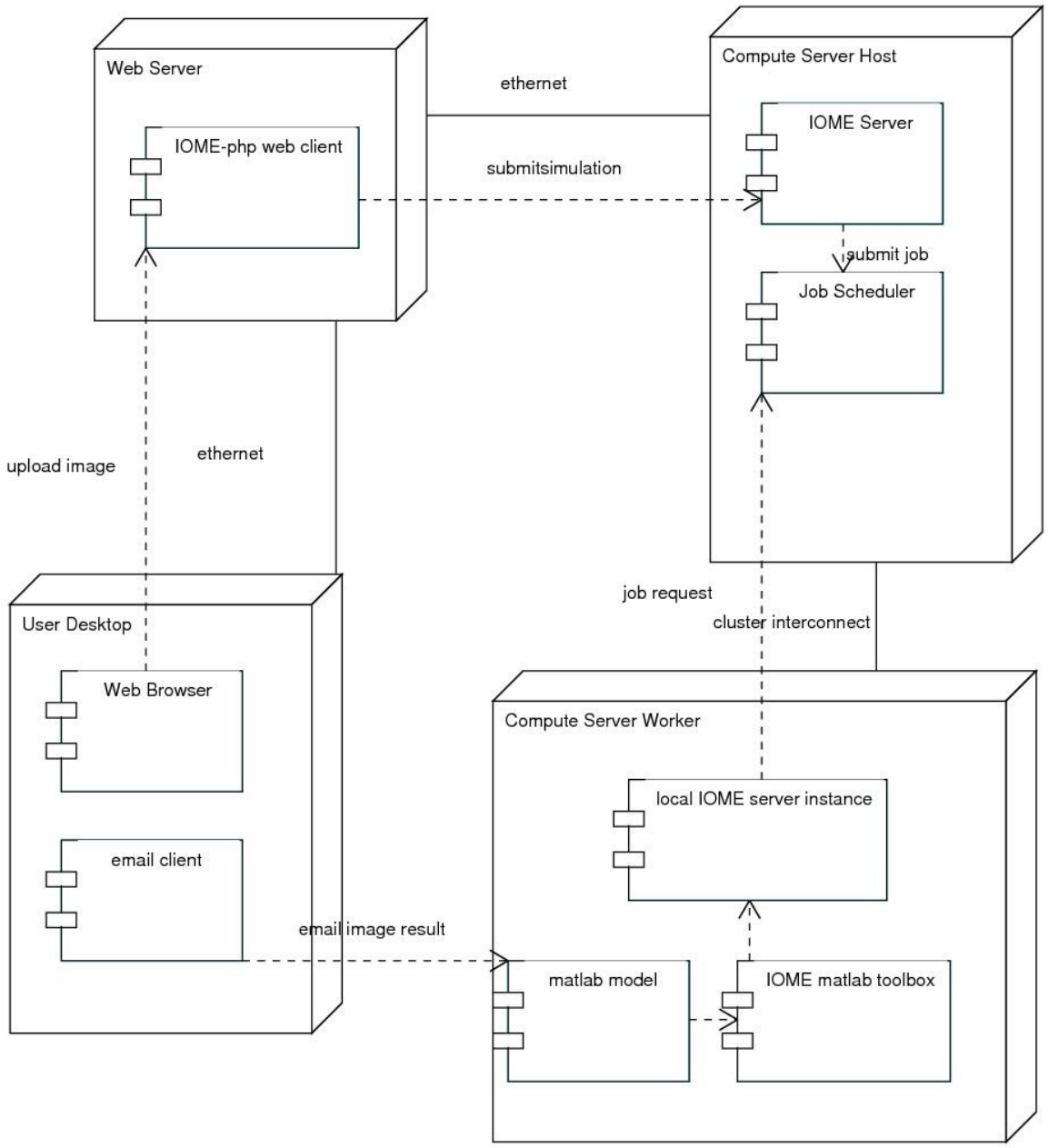


Figure 4 Approximately here

For this project, IOME was used from the beginning, although at the start prototypes were tested using axis and EASA. Once the images have been uploaded to the web server, the IOME php toolbox is used and this provides web service client calls giving access to the image analysis service. This request is made using IOME-ML. For the CAIMAN service the request contains the file name of the image to be analysed, the analysis task to be performed and the e-mail of the recipient for the result. The PHP client uses a template IOME-ML file,

string substitutions from the PHP input form values are made in the IOME-ML file, this file is used in the service request, which is made by calling the **submitsimulation** method.

To set up the CAIMAN service the application developer must start a running instance of the IOME web service (IOME-WS) that will handle potentially multiple incoming requests. The IOME-WS instance runs on iceberg, the head node of the high performance computing service at The University of Sheffield. The compute resource comprises a head node, connected to a farm of execution nodes, accessible to networked computers at Sheffield. On iceberg, a job is queued via the Sun Grid Engine scheduler to the farm of execution nodes where each image will be individually processed according to the algorithm selected on the front-end. We now describe the steps the developer must take in order to set up the IOME-WS instance, this instance will run on the head node of the compute service and may be secured using https. The service runs the job using a generic script file from the developers user account on the compute service. The developer provides an initial IOME-ML simulation file and a script file to run the job, in this case the job will be a Sun grid engine script file that runs the image processing Matlab job. The **submitsimulation** method creates a folder using the job id and time stamp and writes the IOME-ML file for the requested job this new file contains all the required parameters and data to run the job. The simulation is started by running a script called **iogenericsim.sh**, this script file must be provided by the application developer. It is important to note that the request to run the simulation is handled by the IOME-WS, which spawns a separate process thread running the actual simulation. By using the job identifier and the job status for that job, the IOME-WS is able to keep track of all the job requests under its control. In the CAIMAN example the **iogenericsim.sh** job script starts up its own local instance of the IOME server and takes as input the IOME-ML script file generated by the IOME-WS, the script file will also start the Matlab job, this uses the IOME Matlab client calls to obtain the required parameters from the IOME server. An alternative is to utilise the XML parser to extract the job parameters. It was found that using a local IOME server provides an easier method for extracting the job data. In summary, to set up the web service, the application developer performs 3 operations;

- Edit the **iogenericsim** script this will need to start a local IOME server, load the simulation file generated by the request, initiate the Matlab job and finally delete the simulation when it has completed.
- Add IOME client calls to the Matlab script, enabling it to set the correct job parameters.
- Start the IOME web service, which will be running on the head node.

To test the correct operation of the deployed service instead of the **submitsimulation** method, the **requestsimulation** and **runrequestedsimulation** methods were used, this approach does not delete the

working files and allows the developer to diagnose any unforeseen issues. For the CAIMAN service the results of the analysis will be later sent to the user via email. The UML activity diagram shown in Fig. 5 and the process is illustrated in Fig. 6. Each lane of the activity diagram shows the main activities taking place during the lifecycle of an image processing job run using CAIMAN. The lane labelled user includes the activities of the user desktop and the web server. The actual server is represented by two lanes illustrating the CAIMAN web service, which is always handling requests and the thread, which handles the job request for and individual job. The lane labelled job is a scheduled process that runs on one of the nodes of the compute cluster. Images uploaded to the server are restricted according to file size at the client side.

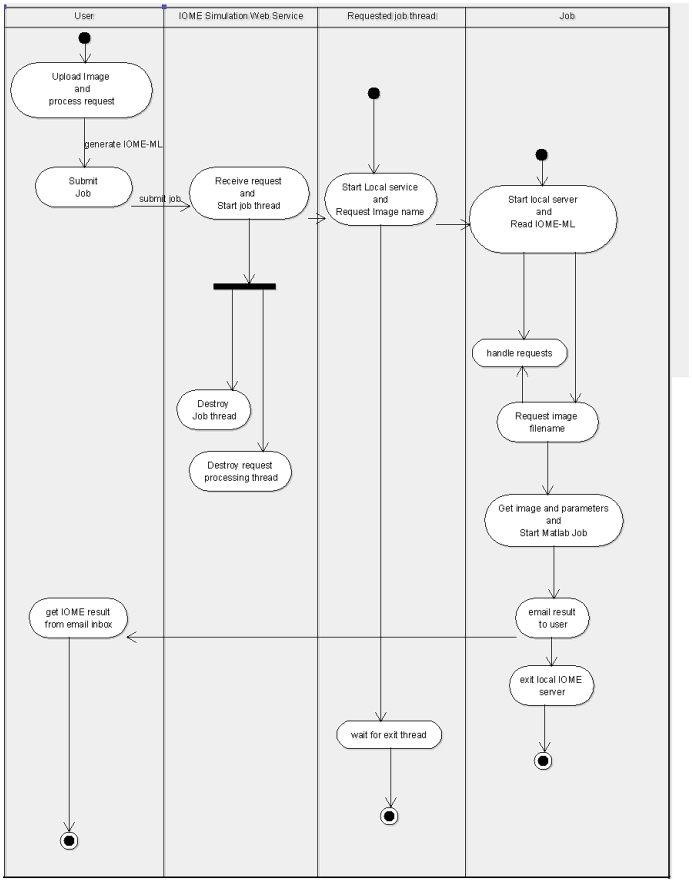


Figure 5 Approximately here

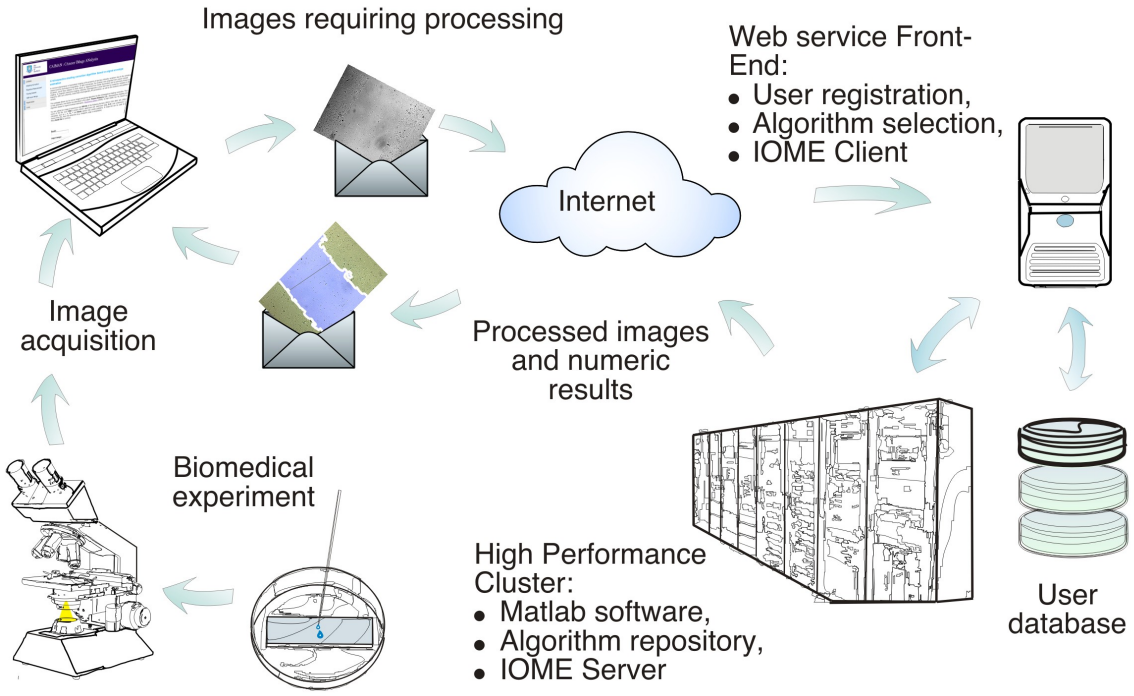


Figure 6 Approximately here

One example of the algorithms available in CAIMAN, measuring cellular migration , will be described to exemplify how IOME was used to run an image-processing algorithm for the analysis of migration of vascular endothelial cells in culture. The algorithm was programmed in Matlab to detect the cellular regions on either side of an artificial 'wound'. This wound was made by dragging a sterile pipette tip across a uniform layer of cells (the monolayer), which had been left to populate entirely a container. This type of measurements are important in the analysis of cellular migration *in vitro*, as it is assumed that the dynamic behaviour of cells *in vitro* is related to *in vivo* processes such as wound healing or the activity of metastatic tumour cells. The algorithm is described in detail in, but briefly: the segmentation algorithm consisted of the following steps: frequency low-pass filtering and thresholding, closing and opening morphological operations to consolidate the cellular regions, approximation of boundaries and calculation of distances. The results consist of an image with the boundaries labelled in white lines, the region of the wound in a different colour and a list of the measurements calculated (Fig. 7).

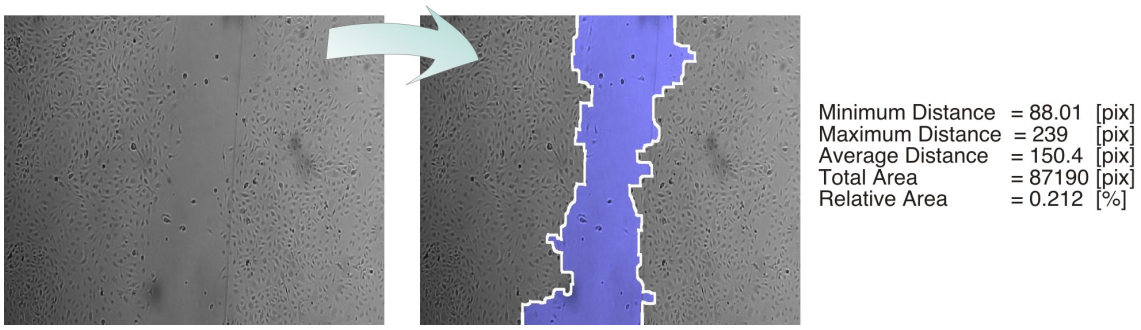


Figure 7 Approximately here

As an indication of the performance of the CAIMAN system, the three algorithms were tested with two groups of five images each; one of approximately 100 kb in size and the other of more than 1 Mb as noted in the following table. The times were recorded from the moment that a user opened the corresponding web page to the time the email was received. The results are summarised in table 3.

	Dimensions (pixels)	Size (kb)	File format	Time ± std (s)
Migration	285 × 203	100	png	62.6 ± 9.6
	1270 × 900	1700	png	81.4 ± 16.7
Tracing	220 × 164	108	bmp	66.2 ± 20.3
	768 × 576	1300	bmp	207.4 ± 14.6
Shading	285 × 203	100	png	59.0 ± 14.1
	1270 × 900	1700	png	65.0 ± 15.6

Table 3. Performance results for the CAIMAN service.

The baseline processing for the images was around one minute, which comprises the time to type the user email, browse to select the image, upload it, process it and email results back to the user. In the cases of migration and shading, there was a small increase in time with the larger images. The tracing algorithm presented a considerable increase with the size of the image. This is due to the algorithm itself; with a small image, few vessels are detected, whilst with a large image, many vessels are detected and the computational complexity increases accordingly.

4. Design of the IOME Toolkit

Up to now we have considered the motivation for using IOME we have also reviewed the way in which it is used. In this section we now focus on the design of the toolkit. IOME uses a protocol based on an XML markup called IOME ML, this is the Interactive Object Management Environment Markup Language. The purpose of the markup language is to,

- provide a framework for describing simulations and models,
- enable interoperability between different simulations with differing data formats and
- enable the movement of data between local, heterogeneous and geographically distributed simulations.

There is an abundance of mark up languages for describing simulations, including mark up languages for systems engineering such as Boeing's Simulation Reference markup language (SRML) . The X-Machines Markup language (XMML) is used to describe and generate simulations of complex systems , XMML is used by the flexible agent modelling environment. More domain specific mark up languages include the Fusion Simulation markup language for magnetohydrodynamics and systems biology markup language (SBML) for describing networks of biological interactions. The SBML was originally inspired by the unified modelling language for engineering object oriented systems. These domain specific markup languages exhibit the required features for describing generic simulations. We encapsulated the most appropriate features within IOME ML.

IOME ML has a collection of XML elements used to describe a simulation, this features a range of elements including,

- *metadata* elements for storing metadata names and properties,
- arrays of parameter elements containing data of different types.

The metadata elements are aggregated by a *metadatalist* element and the parameter elements are aggregated within a *props* element containing a sequence of *prop* elements. The prop elements are used to represent the collection of simulation parameters. A sample of the IOME-ML is shown below

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<iosim filename="iocaimanphp.xml">
   <simulation name="mysim">
         <metadatalist>
            <metadata name=”server” content=”mickey”></metadata>
            <metadata name=”service” content=”caiman”></metadata>
         </metadatalist>
         <props flag="7" name="" numprops="3">
                  <prop flag="7" index="0" name="useremail">
                           <string>%useremail%</string>
                  </prop>
                  <prop flag="7" index="1" name="imagefile">
                           <string>%imagefile%</string>
                  </prop>
                  <prop flag="7" index="2" name="jobtype">
                           <string>%jobtype%</string>
                  </prop>
         </props>
   </simulation>
</iosim>
```

The IOME toolkit was developed as a library of C++ classes, which can be used to parse and write IOME-ML documents. These tools make use of the Apache XML libraries known as Xerces-C and Xalan-C. The IOME toolkit has an **IoXMLSimulation** class, which can be used to create objects for containing the simulation data. The IOME server and client tools use the **IoGenericSimulation** class; this is a child class deriving methods and properties from the **IoXMLSimulation** class. The special feature about the **CIoGenericSimulation** class is that it allows dynamic creation of the data structures used to describe a simulation. The main application in the IOME toolkit is the **iogs** application; the iogs application is based on a realization of the **CIoGenericSimulation** and uses methods from the gsoap toolkit to enable client-server communications using SOAP 1.1. The **iogs** application is used to start the IOME server and can be used to make client requests to an existing service. For scenarios where the IOME server is used to run simulations as web accessible services the server generates multiple simulation instances. The IOME toolkit provides a Web service description file, which can be used to develop client applications. To make web service client application development easier, client development tools are provided for the languages described in section 2. The main classes are illustrated by the class diagram shown in Fig. 8. It illustrates the main classes used to contain the simulation parameters and metadata. Also shown are the classes used to parse and generate the IOME-ML from the simulation itself.

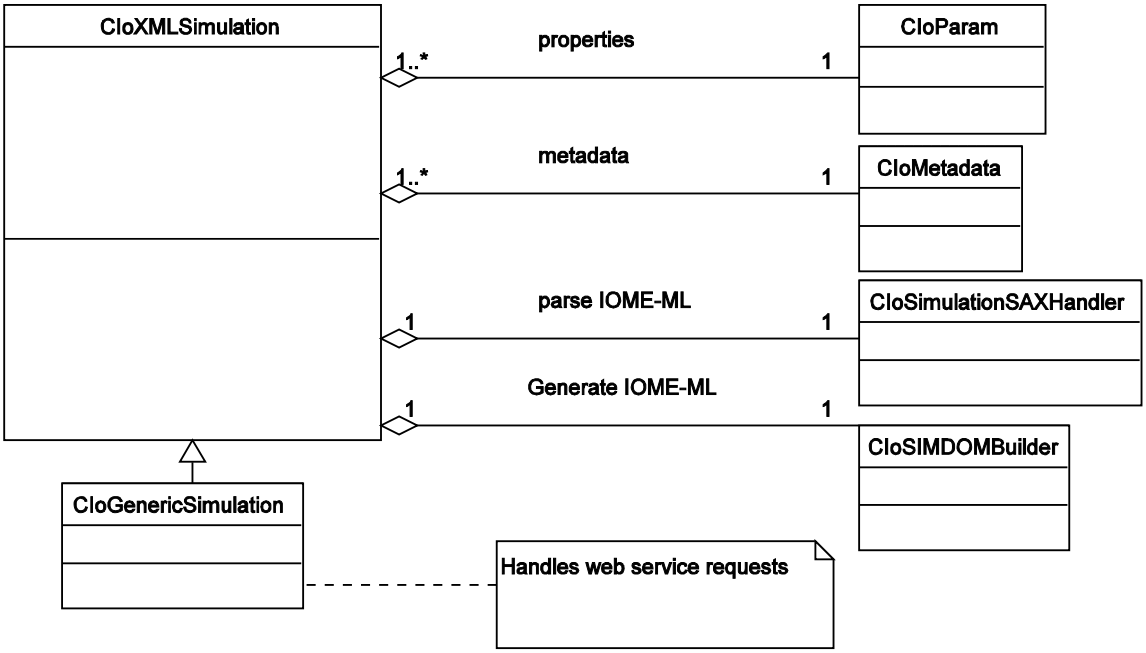
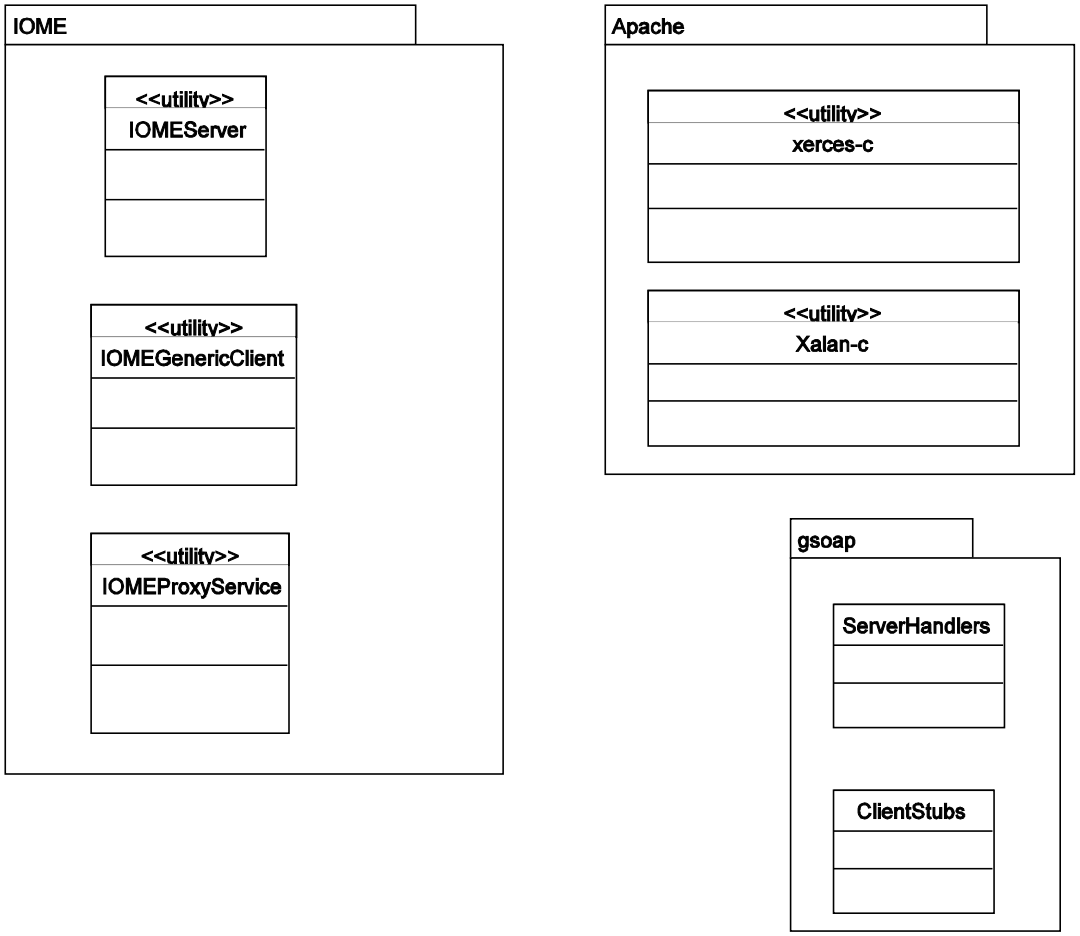


Figure 8 Approximately here

The utilities and client libraries, which make up the Interactive Object Management environment, are shown in Fig. 9, also illustrated are the library packages upon, which the toolkits and IOME utilities are dependent. The main IOME utilities shown are;

- IOME server run using the iogs application
- IOME generic client, using the iogs application and the IOME libraries
- Proxy service, an application enabling requests to be redirected from the proxy server to the actual IOME server


IOME
<<utility>>
IOMEServer
<<utility>>
IOMEGenericClient
<<utility>>
IOMEProxyService
Apache
<<utility>>
xerces-c
<<utility>>
Xalan-c
gsoap
ServerHandlers
ClientStubs

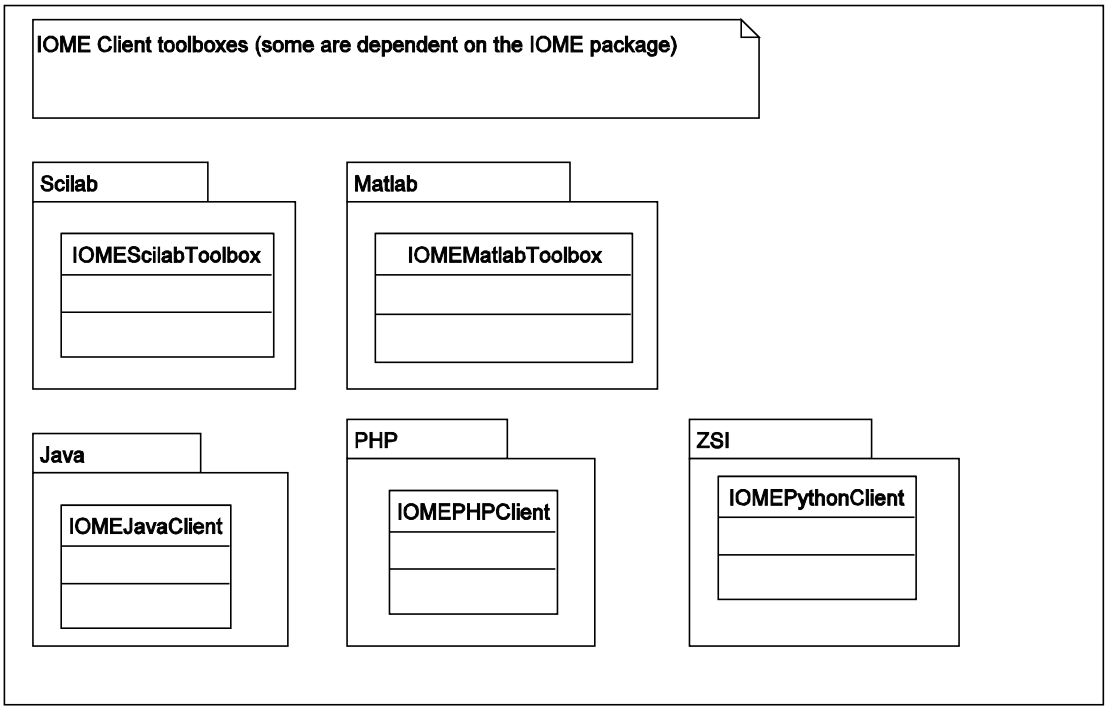

IOME Client toolboxes (some are dependent on the IOME package)
Scilab
IOMEScilabToolbox
Matlab
IOMEMatlabToolbox
Java
IOMEJavaClient
PHP
IOMEPHPClient
ZSI
IOMEPythonClient

Figure 9 Approximately here

Clients can be developed for applications using a range of different development tools for example;

- C/C++,
- Python,
- Fortran,
- Matlab,
- PHP and
- Scilab.

The IOME server application was developed using the well known web service software development kit called gsoap . A range of tools are available for developing web service clients for C/C++ and FORTRAN. For Matlab, php and python the client toolbox was developed using the;

- ZSI library for python ,
- web service tools provided with Matlab and
- php soap plugin.

By using a web service toolkit such as gsoap, IOME provides a high performance in handling and requesting SOAP requests . Performance results for the service were presented in the section describing the CAIMAN use case study. The factors determining the performance of a simulation published using IOME are

- web service request
- number of requests handled per hour
- data transfer (the CAIMAN case study is dependent on the image sizes)
- job request time for processing job request by scheduler (dependent on queue size)
- time for running the job which is dependent on the system specification (for example cpu and memory)

Tests using the CAIMAN project have indicated that the performance and reliability of the IOME server are within acceptable levels for a moderately sized active community of researchers. Fault tolerance is built into the IOME server by handling each request using an individual thread of execution the service request handlers also make extensive use of exception handling. In order to capture fault cases unsuccessfully handled by the server we have made use of a separate service monitor application, which tests the IOME server. This is a short script which uses the **getparam** method to collect status information from the server. If it fails to get the correct status the service monitor application e-mails an appropriate message to the service administrator. The status information may also include information about compute server loads, for example the job queue size for any scheduling system employed by the IOME service. If a failure of the IOME server occurs during a simulation run, it is

normal for that simulation to run to completion, some failure modes may be dependent on the scheduler implementation that has been adopted, however with a service utilising multiple compute workers in a cluster it is not unreasonable to expect a greater level of service resilience. One possible service configuration is to allow the IOME server and the actual service tasks to run on the same compute server for some problems it might be expected that this scenario would have a reduced reliability. In general the favourable configuration that should be utilised is to make use of two compute servers, one server handles job requests from the clients and submits tasks to a scheduler. The second server is a worker which is used to run possible multiple compute intensive simulation instances.

The IOME toolkit was designed to handle a number of use cases for collaborative research, the following scenarios have been considered.

- Publishing a simulation as a web accessible service,
- Sharing data between different client applications running in different locations - this can be used for collaborative working or in a computational steering scenario and
- generate simulation metadata, which may be used to manage distributed data collections and/or for automated laboratory notebook generation.

This paper has considered the first use case i.e. the publication of applications as a web service.

5. Conclusion and Future Developments

The CAIMAN case study presented here demonstrate that researchers using IOME can easily develop modelling and analysis applications that are able to communicate useful data and results between geographically separated researchers. It has been shown that researchers can develop practical web interfaces, which allow collaboration partners to run simulation and analysis applications . The main benefit is that researchers can set up these models without the need for a detailed understanding of web services and protocols such as SOAP. The toolkit may be used as a method of aggregating web content and as such may be viewed as a means for generating a mashup.

The current implementation of IOME assumes a service provision model for which the services are static and stateless. In a service model with multiple applications a more appropriate approach is to make use of a stateful service model. In this model we have a continuously running service factory and registry that can start an instance of an IOME service. It is therefore necessary to provide a system enabling the dynamic invocation of web services and a registry service providing information about the currently available services. The service must

enable a researcher to make requests, which results in an application being published by the service manager and registers the service as available. Researchers developing workflow applications must be able to find information about available services and how they are called. The final requirement is therefore an application request mechanism that a user application must make, this service requester would query the service registry, determine the state of the service. If necessary the registered service would be created dynamically and would return an end point for the invoked service. This application request mechanism would then use the returned end point to request the actual service. Much effort has already been invested in developing the standards to resolve such dynamic service provision models and this is one of the reasons for modern standards such as the web services resource framework . Tools such as *dynasoar* are available for dynamic creation of web services and registry services such as *grimoires* are available for discovering information about services and workflows .

The main method for securing IOME web services has been to run an IOME web service on a dedicated network, such as a cluster. Users accessing such a cluster remotely may use secure shell enabled tunnelling. With a view to enhancing the current security arrangements it is proposed that the following tools are:

- use secure sockets layer plugin with gsoap
- use the gsoap wsse plugin, i.e. the WS-Security standard
- a further possibility for enabling access to globus GSI authenticated facilities is to ensure the secure availability of web services and clients provided grid facilities using X-509 based digital certificates for authentication.

The tools presented here may be employed to build a range of collaborative applications, running across heterogeneous platforms. It has been seen that the IOME toolkit provides the benefits listed below:

- allows researchers to publish a simulation or analysis application as a web accessible service.
- The API may be employed to generate a crowd computing application, alternatively the web service client may be used to request work from the IOME server in a cycle scavenging scenario.
- The IOME toolkit allows data sharing between client applications running in different locations for collaborative working or in a computational steering scenario,
- it also enables the automated generation of simulation metadata, which may be used to manage distributed data collections.
- The IOME-ML has provided a useful model for describing parametric data used by a diverse range of simulations,

Experience with the CAIMAN project has demonstrated that the IOME toolkit provides a novel approach for implementing software as a service and to exploit

compute resources in the cloud. By making simulations and analysis accessible as web services, work flow engines such as *Taverna* provide an easy programming interface and enable different applications to be merged into a single workflow. This approach to application development provides a possible approach for research teams to readily solve multidisciplinary optimisations problems. Multidisciplinary optimisation problems will typically explore large parametric spaces and have the potential to consume huge quantities of computational clock cycles. If such a service is to be made available, the computation facility would need to provide a service for hosting and managing the simulation services.

Because of its adoption of a user lead design approach the development of applications for collaborative research using IOME is likely to result in tooling that accurately captures user requirements, this approach is essential for the flexible development of mashups or Virtual Research environments. By enabling interoperability between a range of different components and API's, IOME makes it practical to integrate research applications with modern social or collaborative computing systems.

6. References

1. Puppin, D., N. Tonellotto, and D. Laforenza, *How to run scientific applications over Web services.* 2005 International Conference on Parallel Processing Workshops, Proceedings, 2005: p. 29-33

2. Novotny, J., M. Russell, and O. Wehrens, *GridSphere: a portal framework for building collaborations.* Concurrency and Computation-Practice & Experience, 2004. **16**(5): p. 503-513.

3. Griffiths. *The Grid Application Portal for Glass Technology. (GAP-GT)* in *e-Science All Hands Meeting*. 2004. Nottingham.

4. EASA. *EASA and Service Oriented Architecture (SOA)*. 2007 [cited; Available from: http://www.easasoftware.com/articles/EASA_Service_Oriented_Architecture.pdf.

5. Hull, D., et al., *Taverna: a tool for building and running workflows of services.* Nucleic Acids Research, 2006. **34**: p. W729-W732.

6. Brodlie, K.W., et al., *Distributed and collaborative visualization.* Computer Graphics Forum, 2004. **23**(2): p. 223-251.

7. Brooke, J.M., *Computational Steering in RealityGrid*, in *e-Science All Hands Meeting*, S.J. Cox, Editor. 2003: Nottingham.

8. Avar, A. *The AXIS Project*. 2003 [cited; Available from: http://ws.apache.org/axis/.

9. Krishnan, S., *Opal: SimpleWeb Services Wrappers for Scientific Applications*, in *IEEE International Conference on Web Services (ICWS'06)*. 2006.

10. Reyes-Aldasoro, C.C., et al., *CAIMAN: An online algorithm repository for Cancer Image Analysis.* Comput. Meth. Prog. Bio., 2010. **in press, doi:10.1016/j.cmpb.2010.07.007**.

11. Reyes-Aldasoro, C.C., et al., *Measuring cellular migration with image processing.* Electron. Lett., 2008. **44**(13): p. 791-793.

12. Reichenthal, S.W. *SRML – Simulation Reference Markup Language*. W3C Note 18 December 2002 2002 [cited; Available from: http://www.w3.org/TR/SRML/.

13. Holcombe, M. *A General Framework for Agent-based Modelling of Complex Systems*. in *The European Conference on Complex Systems*. 2006. University of Oxford.

14. Shasharina, S.G. and C. Li, *FSML: Fusion Simulation Markup Language for interoperability of data and analysis tools.* CLADE 2005: Challenges of Large Applications in Distributed Environments, Proceedings, 2005: p. 115-121.

15. Finney, A., *Software Infrastructure for Effective Communication and Reuse of Computational Models*, in *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*, Z. Szallasi, Editor. 2006, MIT Press. p. 355-378.

16. van Engelen, R.A. and K.A. Gallivan, *The gSOAP toolkit for web services and peer-to-peer computing networks.* Ccgrid 2002: 2nd IEEE/Acm International Symposium on Cluster Computing and the Grid, Proceedings, 2002: p. 128-135.

17. Joukl, H. *Interoperable Python ZSI WSDL/SOAP Web Services tutorial*. 2008 [cited; Available from: http://pywebsvcs.sourceforge.net/holger.pdf.

18. De Roure, D., C. Goble, and R. Stevens, *Designing the (my)Experiment Virtual Research Environment for the social sharing of workflows.* E-Science 2007: Third IEEE International Conference on E-Science and Grid Computing, Proceedings, 2007: p. 603-610

19. Banks, T. *Web Services Resource Framework (WSRF) – Primer v1.2*. 2006 [cited; Available from: http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf.

20. Watson, P., et al., *Dynamically deploying Web Services on a Grid using Dynasoar.* Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Proceedings, 2006: p. 151-158.

21. Moreau, L. *GRIMOIRES, a UDDI-compatible web service registry with added capabilities*. 2008 [cited; Available from: http://www.grimoires.org/.

7. Acknowledgements

The authors acknowledge the EPSRC and Cancer Research UK for funding: WRG Phase III: The White Rose Grid e-Science Centre EPSRC reference EP/F057644/1.

Figure Legends

Fig. 1. The IOME Client-Server Application Interface Model

Fig. 2. Activity Diagram for Running a Generic Simulation as an IOME Web Service.

Fig. 3. CAIMAN home page and a specific algorithm webpage.

Fig. 4. Component diagram illustrating how CAIMAN uses IOME to provide an Image Processing Service.

Fig. 5. Activities taking place through the lifecycle of an image processing task using CAIMAN.

Fig. 6 Graphical description of CAIMAN.

Fig. 7 Measurement of cellular migration example. The raw image is processed to detect the boundaries of the cellular regions.

Fig. 8 Class Diagram for the Interactive Object Management Environment.

Fig. 9 Packages, Libraries and Utilities Available with IOME