

# UK Engineering Task Force Evaluation of the OMII Middleware v1.0

Dave Berry (NeSC), Jonathan Giddy (WeSC), Mark Hewitt (NEReSC) and  
Tim Parkinson (SeSC)

Version 1.0, 27th July 2005

## 1 Introduction

The UK's Engineering Task Force (ETF) is evaluating several Grid middleware solutions in order to take a view on their deployability on the resources within the National Grid Service (NGS) and those of the wider UK e-Science community. These systems include those from GridSystems, the Open Middleware Infrastructure Institute (OMII), the EGEE project and the Globus Alliance. This report presents the results of the evaluation of the OMII system.

The main purpose of this evaluation is to examine the strengths and weaknesses of the system and identify the issues that need to be considered before deploying it in a production environment. Companion documents will report on the evaluation of the other systems, using the same set of criteria.

## 2 General Information

The evaluation was carried out using the first public release, known as OMII\_1 version 1.0. The system provides support for running Web Services remotely and for integrating applications into a service provider for remote execution.

### 2.1 Provider

OMII\_1 was released in December 2004 by the Open Middleware Infrastructure Institute (OMII), which is based at Southampton University. The OMII is funded by the UK research councils. It was created in 2004 and is currently funded until 2007. The expectation is that funding will continue beyond that, although this is subject to variations in funding policy.

The authorship and copyright of most of the significant source code in the OMII\_1 release is from the Southampton University IT Innovation Centre, not OMII themselves.

Their website states that "All OMII software releases are tested to a published and documented level."

Currently the system does not have a widely installed base. Active users include this ETF evaluation team, members of the OMII's "managed programme partners" and a team of IBM engineers from Hursley. OMII report approximately 200 registered users from 70 institutions, but it is not clear how many of these are active.

### 2.2 Licensing

OMII\_1 is distributed under a modified BSD licence. The system also includes third party software that is subject to separate licences, mainly Apache or LGPL. These licences are non-restrictive and permit re-distribution and re-packaging even in derived works intended for commercial use.

## 2.3 Supported platforms.

OMII\_1 1.0 only supports SuSE Linux 9.0 for the server (not 9.1 or 9.2). Also, it requires a particular build of JDK1.4.2 that is older than the current Sun release and which has to be obtained from the Sun archive download site. It uses PostgreSQL, which is included in the download and installed through the OMII installer

OMII\_1 version 1.2 has since been released. It provides support for Redhat Linux Enterprise 3, but still requires the older version of the Java SDK.

The evaluation was conducted at the Welsh e-Science Centre, the Southampton e-Science Centre and the Newcastle e-Science Centre. The machines at each site are as follows.

### Southampton Configuration

The gateway machine is an Intel 32-bit PC running SuSE 9.0 ([magrat.e-science.soton.ac.uk](http://magrat.e-science.soton.ac.uk)). This machine is also configured as a Condor submission node. Client testing was carried out on an Intel PC running Windows XP SP2, browser IE 6.0.

### Cardiff Configuration

The gateway machine is an Intel 32-bit PC running SuSE 9.0 ([ygrid12.grid.cf.ac.uk](http://ygrid12.grid.cf.ac.uk)). Client testing was carried out on same machine.

### Newcastle Configuration

The gateway machine is a 1.5GHz Intel Pentium III machine running SuSE 9.0 ([accessgrid02.ncl.ac.uk](http://accessgrid02.ncl.ac.uk)). The machine is dedicated for the purpose of testing the OMII software. Further deployment was restricted due to SuSE 9.0 not being a supported platform at Newcastle.

## 2.4 Support

OMII provided us with e-mail support with a ticket-tracking system. Since we began this evaluation they have introduced an on-line support page which makes it possible to view the progress of tickets. This was introduced with OMII\_1 release 1.2 – see <http://helpdesk.grid-support.ac.uk/omii.html>. The support information is not divided into separate areas for end-users, systems administrators and developers – all support is handled by a single mechanism. Commercial support is not available.

Experience with support so far has been good, with most tickets responded to before the end of the working day.

## 3 Systems Management

### 3.1 Documentation for System Managers

There is a single documentation set that ships with both the server and client applications. This is available in HTML format. Originally it was only available for browsing online after logging into the downloads page. More recently it has been made available at <http://helpdesk.grid-support.ac.uk/omii.html>.

The documentation has an index and also has a keyword search facility.

The documentation is generally accurate but is not always comprehensive. Several pages claim only to “discuss the possibilities in broad terms”. A few parts of the documentation have been wrong; for example errors in the description of the process

for converting an installation from OMII test certificates to UK e-Science certificates caused us delays. These errors have now been fixed.

There is a “quick start” guide for the client installation in the documentation. The “quick start” for the server installation is essentially inside the README file.

### 3.2 Server Deployment

The OMII\_1 installation is self-contained. It includes the necessary software components such as Apache Tomcat and PostgreSQL. Provided that all of the pre-requisites are present then the server installation process is straightforward. However – as with many middleware stacks – if something goes wrong (due normally to a missing pre-requisite) then the installation engineer needs to know about RPM's, shell scripts, Perl, Ant, and a bit of Java to work out what is happening.

The OMII\_1 installation is required to be installed as root. However, the install directory is owned by one normal user and the system is run by another normal user. In OMII\_2, the installer does not require root privileges at all.

For discussion of user management, see section 3.4.

The resource administration and account administration has a reasonably easy web interface.

The server software only requires at most two ports – one http and one https. These have sensible default values (18080, 18443) and can be changed by the administrator by simply editing a Tomcat config file. Client machines do not need any changes to their firewall, as OMII servers do not call back to the clients. The sites have successfully run remote web services across campus firewalls with and without SSL.

The original documentation did not describe these ports. This has been added to the documentation in more recent releases, so that it can be passed to firewall administrators. In a similar vein, it would be useful if the documentation reminded the reader that any routers on your network should also allow incoming connections on this port.

The Southampton OMII server has been up since October 2004 and hasn't crashed (but has been deliberately restarted from time to time). However, it hasn't been loaded significantly so we can't really make a judgement as to stability.

Newcastle had initial problems with the install scripts due to the OMII software not taking into account the environment where web access is only permitted through an http proxy. This was resolved initially for the base services installation but other parts of the installation required the installer to obtain a file from a web page rather than automatically through the script.

The environment needs to be fully set up for “UK English” otherwise the installation fails reporting an error with the format of the dates. Changes to environment variables are needed to make the installation work. OMII included this information in their FAQ once they were made aware of the problem.

Newcastle's services have been up for several months without any known problems but have not been extensively load tested.

### 3.3 Client Deployment

A command line and Java toolkit client is available (called ‘ogre’) which allows users to invoke services on an OMII server that trusts them. It consists of nearly 11MB of

Java libraries and 30MB of Ant with the entire 10MB HTML document (describing the server and services installation too).

It seems that the Ant distribution is shipped purely to provide a scripting environment for the example programs and could be deleted once they've been run. Alternatively they could be provided as shell and batch scripts, or perhaps Perl scripts, in the first place. The Client documentation could have been extracted from the whole doc set and shipped alone.

The Client is self-contained and only requires a 1.4.2 JDK. Although OMII only support SuSE Linux and Windows XP SP1 as clients it seems to us that it should work on any machine that has a 1.4.2 JDK. If the client machine runs a personal firewall and that firewall is set to maximum security then the user must allow outgoing connections over port 18080 (and 18443 for SSL).

A user that knows how to put JAVA\_HOME on a PATH can install and run the client applications. We have not tested the system with real end-users.

### 3.4 Account Management

OMII\_1 provides a prototype account management web application. However, this does not allow an administrator to create a new account from scratch. Either the user must request a new account with the `ogre_client` or the administrator must use `ogre_client` to request an account on the user's behalf using the user's DN.

The system should generate better notifications during this process. When a user applies for an account, the administrator is not notified by the server, and when an administrator approves an account request, the user is not notified by the server.

It is not currently possible to use certificates from more than one Certificate Authority. OMII\_1 doesn't provide VO management tools per se, although it is possible to add new users into your account using the delegation mechanism (see below).

Accounts can only be modified (if not created) by the administrator through the web interface. There is no field for time-limiting accounts, only for suspend/deny/close operations.

OMII\_1 provides a Process-Based Access Control (PBAC) system. Services hosted by OMII\_1 may optionally be configured to use PBAC (the sample services do so). PBAC assigns a cost to use a service, in notional funds. It requires that users have sufficient funds to access the service and decrements their account accordingly. It tracks the funds remaining to each user. The system also records per-service usage information in the database but this is not presented to the administrator or user.

The system manager may set allocation limits and the work (per job) limits for each user. We found that these did not affect the behaviour of the system as we expected. In most cases jobs ran despite these limits. Appendix 1 has full details of our experiments.

### 3.5 Reliability

OMII\_1 does not include high-availability features. We haven't tested it under high loads.

### 3.6 Distributed management

The server Resource Administrator (RA) can add machines to its list of worker nodes and can add Application Suites / Applications to its list of services and can map Application Suites to machines. The RA can price and limit file transfer to and from the server independently.

The RA can set relative performance characteristics of each machine that s/he knows about. This modifies the pricing of the service. 'Faster' machines cost more per unit; the difference is seen at tender time.

### 3.7 External Services

Web Services and batch applications can be deployed on the server with no modifications. The deployer has to provide wrapper scripts and modify some configuration files. This requires a fair amount of effort.

Newcastle attempted to configure the OMII services to run using a Condor installation which was already present on the test hardware as part of the Newcastle infrastructure.

The work involved modifying the deploy wrappers for the GRIA test application to call the local Condor installation. This procedure was clearly outlined in the documentation. It was less clear as to how to make the system use Condor as its default job manager. However it was a straightforward matter to run the test application using the Condor wrappers.

There were also wrappers pre-installed for PBS, from the look of these it should be possible for someone who is reasonably familiar with the system to make custom wrappers for any particular job manager or queue system.

Unfortunately the Newcastle installation was not able to be fully tested. The OMII application correctly created the submission script and then submitted that to Condor. However Condor then replied that it refused to run jobs as root. This is correct behaviour from Condor and just reflected an incompatibility arising from the way Newcastle had chosen to install OMII. Further progress was not possible in the time constraints, however everything looked like it should run correctly.

### 3.8 Scalability

We have not tested the effect of scalability on performance.

The use of service pricing should provide an incentive for other sites to provide their services to users and a mechanism to ensure fair sharing among different users, two major issues with the scalability of many infrastructures. However, there is currently no mechanism for discovery of services, which limits the ability of a user to scale up their computations to using many sites.

The account management process, while simpler than a user account-based mechanism, still requires the interaction of an administrator for each user. This may significantly affect the number of users a single site can handle. An alternative is to configure a site's OMII to auto-approve all new accounts. This removes the requirement for administrator interaction but bypasses the PBAC system, essentially turning the machine into a free service.

## 4 User Experience

### 4.1 Documentation for Users

There is a single documentation set which contains a mix of design documentation, administrator documentation, and user documentation. Page one of the original documentation recognised the different classes of reader but the documentation did not preface or colour code the various chapters with “This should be read by ...”. In a more recent release, the first page now directs different classes of reader/user to different parts of the documentation.

### 4.2 Joining the Grid

The user must install the client app (*ogre*) and request an account on the server machine. As previously mentioned, unless user continuously runs *ogre\_client* check-accounts then s/he must rely on the administrator to send a courtesy e-mail by hand when this is done. The user is then a *budget holder* for an account and may request *allocations* from the server which represent pre-paid chunks of “project money”.

The *allocation* can be spent either by the budget holder running *jobs* or *applications* (which consume CPU time, storage, and bandwidth as priced by the server resource administrator) or else the budget holder can delegate (share) the allocation(s) with colleagues.

Communication with the server is authenticated purely by X509 certificates. The server must be configured to honour the CA that signs the certificates. At the time of writing, a given OMII server can only honour a single CA. OMII intend to support multiple CAs in a future release. This can be worked round by running multiple instances of the server on different ports each using a different CA in their keystores.

A non-budget holder user can use OMII services via *delegation*. To achieve this non-programmatically, the user must transmit their certificate (presumably just the public part, although this is not stated in the documentation) to the budget holder who must import it into his/her client keystore. Then the budget holder can use the

```
ogre_client browse
```

command to select the allocation s/he wants to share and use the Enable Access function, selecting the subordinate user’s certificate from the list. Then the budget holder must extract the allocation description fragment from the OMII client’s **client.state** file and transmit it to the subordinate user who in turn must merge the fragment into their own **client.state** (FileStateRepository) and subsequently run jobs against it.

[A backup should be kept of the client.state file at this point in case the Forget Conversation function is used for some reason – without the backup there is no way to recover a users active allocation conversation from the server. Perhaps an

```
ogre_client refresh Account.xml
```

command would fix this scenario.]

This procedure is complicated and relies on manual operations so is therefore error prone. But in practice, client application front ends would perform these functions programmatically via the Java API.

### 4.3 Legacy Application Integration

A sample application was integrated into the OMII server. Details of the process are described in Appendix 1.

The user or developer of a legacy application must:

- a) install the application on the server;
- b) write one to three wrapper scripts to translate the OMII invocation into an execution instance. The function of these scripts can be quite complex. The shipped example runs to nearly 300 lines of Perl.
- c) have a server administrator modify a properties file hidden deep in the OMII server file hierarchy and add the application to the Resources of the server.

The only reliable way for an application wrapper developer to test their work is to have a local installation of the OMII server (or to have admin access to a non-production server plus the ra password). The integration requires intervention by both the server manager and the resource administrator which could be two different people.

The resource manager information is saved in the database and should be preserved across upgrades. However the `jobservice.properties` file that contains the definition of the wrapper scripts will be lost upon upgrade unless its contents are reapplied.

### 4.4 Migration between platforms

We have not tried to port services from GT4 to OMII\_1 or vice versa.

### 4.5 Usability

The client can poll the server to find out whether a job is started/running/finished. Asynchronous notification is not provided in this release. More detailed status tracking is up to the application to provide. Application wrappers are required to write status codes for the client to poll.

### 4.6 Security from the User's Perspective

Like many other 'grid' type systems OMII operates using X.509 certificates at the core of its security infrastructure. If the user already has a certificate and is familiar with their operation then getting access to / deploying OMII services is straight forward. Otherwise they will have to go through a certificate application process which will vary according to which certificates are accepted by the OMII server.

On installation of both the server and the client, OMII installs temporary certificates from a special OMII Certificate Authority. This simplifies setup for people trying out the OMII middleware, who do not need to wait for the issue of a properly authenticated certificate. However, the user needs to be informed that the certificate is being issued and that this certificate provides only low value authentication, in order to educate them about the need for secure certificates. The installation also needs to check for existing certificates to prevent external certificates being destroyed by a user reinstalling the middleware.

### 4.7 Verification

Application verification is via a status code returned by the application, or more precisely the application wrapper script. The framework returns the code given to it

by the wrapper. Any status code to diagnose spurious results must either be issued on the server side by an intelligent wrapper script or on the client side by checking the results.

Verification of the environment is not applicable because the execution model is that applications are deployed on the server and invoked remotely, as opposed to submitting arbitrary executables to be run.

If the OMII server resource administrator advertises an application then the application can be tendered for. There is no way to check the consistency of the server's advertised services, e.g. whether the properties files are set correctly or whether the actual programs are installed. Clients only discover problems upon job execution (in which case their allocation is spent and the server keeps the money).

## **5 Developer Experience**

### **5.1 Documentation for Developers**

See above for a description of the documentation. The documentation would benefit from more examples for developers.

### **5.2 Languages and Tool Support**

OMII uses standard Web Services and therefore any WS development environment is sufficient for writing services. To deploy a non-PBAC web service is the same as with any Tomcat/Axis container.

To deploy a legacy application as an OMII Application within an Application Suite., someone has to write complicated wrapper scripts and change several configuration files. No specific support is provided for this and it can be a tricky and time-consuming process.. There may also be some extra effort required to support secure services.

### **5.3 New Services**

Developing and deploying a service is documented in two places: in the Base and Extensions section, where a provided TestService is deployed; and in the Services section, in the Developing Services subsection. Both sections provide a simple example of a standard insecure service, followed by an example of checking authentication and authorisation details.

The Services section also provides a suggested model for wrapping a web service implementation in order to enable it in OMII. The deployment of services is fairly simple, particularly to developers already familiar with the standard Apache Axis model.

However, the example code in the Services section is incomplete, which makes it difficult to work out the exact structure of the OMII Wrapper class for a new service. For example, the required imports are not specified and there is no class hierarchy reference in the documentation. Full example code for the Base and Extensions section of the documentation is provided in the Appendices of the documentation, but this is not referenced from the Services documentation.

In addition, the provided example does not use the suggested two-class Implementation and Wrapper model, so when a developer wishes to expose their developed Implementation class, there is no clear example to follow in the

documentation. Despite the presence of reference sections on Axis and Web Services there is no example of converting a service implementation developed using WSDL2Java to an OMII service. There is no example of initialising and using the Implementation from the Wrapper.

Also, the documentation and the example in the Services section both use a method of initialising the Wrapper class that does not work in the current implementation. The documented constructor passes the ServiceContext parameter to its superclass (EScienceService). Actually, the superclass constructor does not accept any parameters, and the only way to set the service class is using the setServiceContext method.

We did not test the authentication code from the second section.

## **6 Technical**

### **6.1 Architecture**

OMII\_1 uses Web Services and maps to a Service-Oriented Architecture.

### **6.2 Standards & Specifications**

OMII\_1 is currently built on Axis 1.2 which implies SOAP 1.1 and WSDL 1.1. In future, OMII's managed programme plans to add support for UDDI, BPEL, WS-RM, (Reliable Messaging), WSRF and other WS standards.

### **6.3 Security**

OMII\_1 includes a partial implementation of WS\_Security (for details , see <http://wssecit.sourceforge.net>). OMII's implementation of WS-Security only provides message signing, not encryption. Encrypted traffic is only possible by setting the supplied Tomcat to listen on https, which you can do with a Vanilla Tomcat. Users' allocation and account details are stored in plain text on the client, so your details may be as safe or as vulnerable as your laptop.

If the resource allocation system allows a user access, that user can access any application on the server. The only way to have more fine-grained control is to run multiple containers.

The default installation runs over http rather than https and uses temporary OMII certificates. So (by default) all packets are plaintext unless encrypted by the application and users shouldn't trust the message signatures as much as an installation using certificates from the UK e-Science CA.

We are not aware of a security audit of the system. The system has not been widely deployed. The system does not have a documented auditing system. We do not feel qualified to answer the more detailed security questions.

### **6.4 Industrial Support**

OMII\_1 1.0 only runs on SuSE 9.0, a particular release of PostgreSQL and a specific build of the JDK. This severely restricts its suitability for deployment on commercial hosting systems. The more recent 1.2 release also supports Red Hat Enterprise Linux. The limitation to a specific build of the JDK will be removed in a later release. However, it still seems some way off releasing a general purpose version which will work with any Linux platform.

To use such industrial strength hosting environments, a middleware supplier has to cope with different underlying operating systems and database systems. A commercial web hosting service provider will expect to upgrade operating systems, databases, and JDK/JVM's from time to time to add functionality, improve security etc. They will expect hosted software to be upwards compatible.

## 6.5 Capability & Functionality

OMII\_1 allows certified users or their delegates to execute a service that is pre-registered on the server. This is implemented by either the OMII default job manager (which uses 'at') or the OMII-supplied but untested and uncertified links to Condor or PBS. Users or their delegates may also upload files to and download files from persistent storage on the server and allow access to other certified users. This uses SOAP With Attachments (SwA). OMII encountered some performance problems with the Axis implementation of SwA and have contributed a patch back to Apache; we don't know whether Apache have accepted this patch.

OMII\_1 provides an account service that provides basic resource is a combined authentication and authorisation mechanism. Users require an account and a budget to access any resources and to consume resources they need a pre-paid allocation from their budget. OMII and user supplied services can be grouped and assigned to machines and priced per CPU second. On a server by server basis, bandwidth can be priced and limited.

Thus OMII\_1 addresses simple use cases where one site makes a service or application available to other sites. By design, it does not support more complex scenarios such as workflows, data federation or replication, etc. Nor does it provide registries, brokering services, or monitoring services. Future releases may support some of these use cases, perhaps via integration with services from other suppliers.

## 7 Conclusions

OMII\_1 supports secure job submission to remote sites, which can be configured to run the jobs using Condor or PBS. The installation of the OMII\_1 software is reasonably robust and straightforward. Applications must be pre-deployed on the server and this can be a tricky and time-consuming process.

OMII\_1 uses web service standards and supports the invocation of both web service and native applications. It is well-documented and seems to be robust, although configuration information can be lost while upgrading to a new release.

OMII\_1 only runs on SuSE 9.0, includes a specific version of PostgreSQL and requires a specific build of the JDK. More recent releases also support Red Hat Enterprise Linux and the limitation to a specific build of the JDK will be removed in a later release.

The OMII server requires at most two incoming ports: one http port and one https port. All others can be closed. It is not necessary to configure outgoing server port ranges or incoming client port ranges as the OMII server does not perform callbacks.

The account service is a novel feature of this system. It is clear that account management will be needed in a production Grid and steps in this direction are welcome. However, there still seem to be some problems with the OMII\_1 implementation, as documented in Appendix 1. Also, in practical terms this is also a point that could hinder integration with services from other sources, which are

unlikely to support this system. We do not know whether the OMII\_1 accounting service is aligned with developments in standards organisations.

Overall OMII\_1 performs as claimed by the developers. The functionality available is, however, limited and it can not provide a complete Grid solution. A consequence is that OMII\_1 will necessarily have to interoperate with other grid middleware and at present it is unclear how this interoperability will be achieved on the National Grid Service. Some steps are being taken in this direction. For example, future releases of OMII will include OGSA-DAI and the UK e-Science “Integrative Biology” project is pursuing integration with SRB & MyProxy. Our evaluation has not addressed issues such as integration with Grid information systems, data replication systems, or other Grid components.

## Appendix 1 – Hypercubes Application Integration

### Description

The Hypercubes application was provided by the GeodiseLab project. It is a Python script that calculates a Design of Experiments multi-dimensional point sample.

Invocation is by:

```
python <path-to>/hypercubes.py <npoints> <ndims> [ <outfile> ]
```

where `npoints` and `ndims` are non-negative integers and `outfile` is an optional output file (stdout if omitted).

The output is `npoints` lines each containing `ndims` floating point numbers between 0.0 and 1.0.

Runtime grows rapidly as `npoints` and especially `ndims` are increased. Output size is about  $15.25 * npoints * ndims$  bytes. Sample runs directly from the shell command line are:

Run Name	npoints	ndims	CPU time	Output Size
small	3	2	negligible	93 bytes
medium	1500	2	0.6s	46 KB
large	5000	2	8.5s	155 KB
extralarge	15000	4	2:00	915 KB
huge	30000	8	1:06:46	3.6 MB
monster	25000	50	killed after 2:30:00	est 19 MB

and elapsed runtimes are usually of the order of twice the CPU time on a lightly loaded machine.

### Server Integration Process

This consists of the following activities which are all described in the OMII documentation:

#### 1 Install Application on server

This consists merely of placing the script in a directory that is searchable by `omii_tomcat_user` and making the script executable by `omii_tomcat_user` and making sure it runs (and ensuring the python RPM is installed).

I put the script in `/opt/geodise/DemoApps/HyperCubes/bin/hypercubes.py` with universal search and execute permission.

#### 2 Write application wrapper scripts

Following the OMII documentation in Services > Job Service > Adding a new application I wrote a basic Bourne Shell *Start Application wrapper script* that interprets its first input and output options and ignores all others.

The input parameter value is treated as the (path)name of an uncompressed file that whose contents are embedded in the python command. The output parameter value is interpreted as the (path)name of an uncompressed output file that is used verbatim as the third argument of the script. There is no attempt to validate the input data nor to do any sort of security checks. I even skipped the exit status step, claiming success whatever happens.

The Application Status and Application Termination tasks are optional according to the OMII doc so I didn't implement them.

A full set of production scripts for an application would be a non-trivial task. The list of requirements and responsibilities of the wrapper writer in the OMII documentation is quite onerous. Although I've used shell-script here, for portability a real implementation would have to be written in Perl or Python or TCL or some other language universally available on Linux, Unix, and Windows.

The full script is listed here:

```
#!/bin/sh
#
#
# OMII Application Wrapper for hypercubes app.
#
HCHOME=/opt/geodise/DemoApps/HyperCubes
touch .app_started
#
# NPOINTS is number of NDIMS-dimensional points in the
# OUTPUT_FILE
# Generate a 4 dimensional 512 point sample to stdout.
# Now we must parse the args.
# The -i option tells us the location of an uncompressed file that contains
# one line with NPOINTS and NDIMS on it - we don't validate that, leave it to the app.
# The -o option tells us the location to generate the (uncompressed) output to.
#
while [ $# -gt 0 ]; do
    case $1 in
        "-i")
            shift
            INPUT_PARAM_FILE=$1
            shift
            #echo "Input File is ${INPUT_PARAM_FILE}"
            ;;
        "-o")
            shift
            OUTPUT_FILE=$1
            shift
            #echo "Output File is ${OUTPUT_FILE}"
            ;;
        *)
            shift
            ;;
    esac
done
python ${HCHOME}/bin/hypercubes.py `cat ${INPUT_PARAM_FILE}` ${OUTPUT_FILE} >.stdout
2>.stderr
# echo $? > ../.app_exit_code
echo 0 > ../.app_exit_code
# cd ..
touch .app_ended
# if [ "`cat .app_exit_code" != "0" ] ; then
#     echo FAIL > .app_exit_status
# else
#     echo SUCCEEDED > .app_exit_status
# fi
```

### 3 Configure application wrapper into server properties

Chose the Application URI <http://geodise.org/DemoApps/HyperCubes> because the Geodise project own the domain name and because they may have other apps they could donate in future.

As instructed by the documentation I appended one line to:

`$OMII_HOME/jakarta-tomcat-5.0.25/webapps/axis/WEB-INF/classes/jobservice.properties`  
that says:

```
jobservice.appscript.run.http\://geodise.org/DemoApps/HyperCubes=/opt/geodise/DemoApps/HyperCubes/OMIIWrappers/start.sh
```

## 4 Add Application Suite and Application into resource manager with pricing.

Notes:

- The writing of a set of robust production application wrappers is a big task.
- The only reliable way for an application wrapper developer to test their work is to have a local installation of the OMII server (or to have admin access to a non-production server plus the ra password).
- The integration requires intervention by both the server manager and the resource administrator which could be two different people.
- The resource manager information is saved in the database and should be preserved across upgrades.
- However the `jobservice.properties` file that contains the definition of the wrapper scripts will be lost upon upgrade unless its contents are reapplied.

### Client Execution

I followed the Command Line Tutorial steps in the OMII documentation using the `ogre_client` utility with a previously opened account that has lots of credit. The input file contains one line consisting of `npoints ndims` as described above. The `Requirements.xml` and `Work.xml` files had large, non-binding, non-conflicting values (for the *extralarge* job all bytes type limits were of the order of 5 or 7 MB and CPU limits were several thousand seconds).

The execution loop goes like this:

```
# Allocation for the runs
ogre_client tender hcAccount.xml hcReq.xml hcallocname
# select the only one on offer
# then repeat the following 3 lines for each execution
ogre_client upload hcallocname xlInput.dat hcinput
ogre_client run http://geodise.org/DemoApps/HyperCubes/ \
hcWork.xml --input hcinput --output hcoutput
ogre_client download hcoutput hcxlOutput.txt
# give up when you have no more runs or your allocation is exhausted.
# get a refund for any unused resources
ogre_client finish hcallocname
```

At each point in the cycle I could perform an `ogre_client browse` and `Get Statement` to see the charges to my account for each job. These were all in line with the CPU pricing for the application suite and the bandwidth charges for upload and download.

The initial runs were done with the *small* application (3x2) and all subsequent runs used the *extralarge* configuration (15000x4).

Once I'd got the hang of it, all runs completed successfully and the downloaded data was as expected. But see the next section on resource limits for the problems.

Notes:

- If I merely perform a `tender` followed by a `finish` operation with my allocation then the server seems to charge me 10% of the allocation in Euros for the privilege of doing nothing. This pricing policy is not published anywhere on the RA screen.
- The server maintains up to four copies of each input data file. One in the Data Conversation datastore (it's persistent because it could be shared among allocation delegates); one secret copy in the Tomcat `webapps/axis/WEB-INF/attachments` directory; one copy in the job workspace `sharedzips`

- directory; and potentially an unpacked copy in the job work area (created by the application wrapper).
- The server maintains up to four copies of each output data file: the raw output in the job work area (created by the application); one copy in the job workspace sharedzips directory (created by the app wrapper); one secret copy in the Tomcat webapps/axis/WEB-INF/attachments directory; one final copy in the Data Conversation datastore (which persists until the allocation is Finished).
  - All this copying could add up to a large overhead on a server that handles jobs with much larger datasets than the ones I was using. Also, the administrators would have to allow for a generous (four times expected running capacity) amount of extra disc space.
  - The data on the server associated with data conversations (job inputs and outputs) is in general deleted upon allocation being Finished. However, a copy of an input that has been used by a Job is retained in the axis attachments directory (whereas a created output, or an uploaded input that wasn't used is deleted from attachments). This is a surely a resource leak?
  - Upon Allocation Finish, the job workspace in datastore is not deleted. A copy of the job log and all its input and output data seems to be retained by the server forever (until the disc is full and a cleanup is forced at any rate). Also the file permissions on the datastore job workspace allow universal read and execute/search. This can't be right. The datastore directory should be only readable by omii\_tomcat\_user. (Perhaps there is a configuration option to make job workspaces be deleted upon allocation Finish – if so it should be set by default).
  - The client (*ogre\_client run*) receives no feedback at all in case of error due to resource overusage. The only way to diagnose resource overuse seems to be to read the server logs which may not be possible for a remote client.
  - The *ogre\_client upload* command requires an allocation name but the *ogre\_client download* does not (and won't take one). The download command should require an allocation name (I may have more than one active allocation) to make it consistent with upload, run, start etc.
  - The *ogre\_client finish* command does not require an allocation name. I may have more than one active allocation in a state repository. It should require an allocation name to make it consistent with upload, run, start etc.

## Exploring Resource Limits

I explored the effect of the Allocation limits and the Work (per job) limits.

1 and 2) Work limits: Using the *extralarge* job I reduced in turn the work/std-CPU-seconds to 20 and the max-output-volume value to 500,000. I expected the first run to fail due to overuse of CPU and the second run to fail due to too much output or at least to fail to download the results. However both times, the job ran to completion and the output was downloaded in full. It looks as though the default job manager does not enforce this limit.

3) Requirements (allocation) CPU limit. Using the *extralarge* job I reduced the max-work/std-CPU-seconds to 30 (the job uses 120s). But the job ran to completion and the output was downloaded in full. It looks like the default job manager does not enforce this limit.

4) Requirements data storage. Using the *extralarge* job I reduced the allocation's max-store-data-volume to 500K. Although the job ran to completion the download was blocked and the client gave the correct error message.

5) Requirements download limit. I reduced the max-download-data-volume limit to 500,000. I expected the run to complete but the download to fail. But the job ran to completion but the download completed in full. Have I misunderstood the function of that limit and it's a limit on bytes/second or does it just not work?

6) Requirements upload limit. With all other limits back to large values and using the *extralarge* job I reduced the max-upload-data-volume limit to 100000 ( safe in the knowledge that my input was only a dozen bytes.

However this led to quite bizarre behaviour. The upload succeeded (of course) and the job started but *ogre\_client* run just kept on reporting that the job oscillated between 'submitted (FINISHED)' and 'output-staging-in-progress(FINISHED) and would not complete.

Looking on the server I saw that the Python script had terminated and that the output was there in the stagedzips output but that there was no output in the output data conversation datastore directory. The server was continually copying the output file from the stagedzips area into the axis attachments directory then issuing a Data Staging error to the catalina.out and services.log file saying that the transfer breached allocation limits (an Upload limit). Then it would wait about one minute then delete the attachments copy then try again.

When I killed the client and Finished the allocation it had no effect on the server's behaviour except to add a bunch of errors to the process complaining that the conversations no longer existed. The server was still stuck in the copy loop.

I stopped and restarted the server. It remembered what it was doing and carried on copying to attachments and complaining to the log file.

The only way to stop it eventually was to stop the server, delete everything in the webapps/axis/WEB-INF/attachements directory, delete every workspace in the datastore directory and delete the jakarta-xxxx/work/Catalina directory then restart. After a few log messages complaining about things no longer existing it settled down again.

Clearly the upload limit doesn't do what I think it does (or does something else too). The client has no idea this is going on. Also, once I've done all the deletions and Finishing, the allocation is kept by the bank and no refund is issued. The *acct\_admin* has to do a manual refund.