

The CAS Server – User Manual

Contents

1	Introduction.....	2
2	Installation Steps.....	5
2.1	Prerequisites.....	5
2.2	Package Content.....	6
2.3	Installing the Packages.....	6
2.4	Running the CAS Server.....	7
2.5	CAS Server Configurations	8
2.5.1	Configuration File.....	8
2.6	A Simple Client.....	10
3	Local Registry Administration Application.....	10
3.1	Adding a New Execution Node to the CAS Server	10
3.2	Adding a New CAS to the CAS Server	11
3.3	Adding a Function to the CAS Server Registry	13
3.4	Adding an OpenMath Symbol to the CAS Server Registry	14
3.5	Adding Remote Registries	15
3.6	Linking a Function to a CAS	16
3.7	Linking an OpenMath Symbol to a CAS and Main Registry Publication... 18	
3.8	Consulting the CAS Server Registry	19
3.9	Advertising messages to remote registries.....	19
3.9.1	Changes update message structure.....	20
4	Interfaces of CAS Server Service	22
4.1	Task Management Interface.....	22
4.1.1	Grid Factory Service Interface.....	22
4.1.2	Submitting a New Task and Task Management	26
4.1.3	Execution Management Interface	28
4.2	OpenMath Resolver Interface	29
4.2.1	Reference resolving internals.....	31
4.2.2	Service Interfaces Participating in OpenMath Symbol List Retrieval..... 34	
4.2.3	Messages for Retrieving the List of OpenMath Symbols.....	35
4.2.4	Service Interfaces Participating in Target Node Retrieval	36
4.2.5	Messages to Retrieve the Targeted Nodes	37
4.2.6	Download of Result Files.....	38
	References.....	40

Introduction

This document presents the capabilities and design elements of the CAS Server software component, developed in the framework of the European project SCIENCE and contains detailed installation and configuration instructions. The target audience of this document is,

on one hand, computer algebra specialists that may use this software to expose their own work using the Grid services technologies. On the other hand, this documentation is intended to offer required technical details that would enable system administrators to implement complicated symbolic computation distributed infrastructures by installing this software on computational remote computational nodes, and integrating them in a single computational architecture. While the CAS Server may be used in conjunction with the simple client provided with the installation software package, more details are given for more advanced users that may want to implement more complex infrastructures or custom clients.

Several reasons motivate the need for the CAS Server component. The first one is interoperability in the context of distributed architectures. Symbolic computation is mostly done using Computer Algebra Systems (CASs). These systems are standalone software applications or software packages that can be integrated with existing CASs. There are two main categories of such systems:

General purpose systems, designed to cover a large area of functionality required by symbolic computing. While these systems offer a broad functionality, they may not offer the best computation solutions to solve particular problems.

Special purpose systems that offer solution for particular areas of the symbolic computing. These systems are especially designed to solve problems from a particular area of interest. Their efficiency resides in the data representation model used and the specially tailored algorithms.

Integration with distributed systems requires, at least, network interconnection capabilities which most of the current CASs do not provide. Communication with sibling instances or instances of other CAS systems may allow computer algebra specialists to distribute computation and thus, use available computational resources optimally. Communication capability is proven as an important feature for symbolic computing due to the nature of symbolic computing:

Symbolic computing demands huge amounts of computing resources such as internal memory, storage space and high number of computing cycles. When these resources are not available on a single system, a common solution is to implement a distributed architecture that is able to offer the required computational resources.

Symbolic computing problems may be sometimes decomposed in smaller problems that can be solved separately. Using a distributed architecture to solve such problems may dramatically decrease the time required to obtain solutions.

Having in mind the above requirements, CAS Server is a software package that offers the ability to expose the functionality implemented by CASs using standard technologies currently used to implement interoperable distributed systems, i.e. Web Services. The latest developments in the Grid Services technology and its advantages recommend Grid Services (Web services that comply with the WSRF standard) as the best technology to use for the CAS Server component communication interface.

In order to make sure that the CAS Server is versatile enough to meet the expectations and requirements of the symbolic computing specialists, several main objectives were considered throughout the CAS Server design and implementation process:

Offer a standard interface that enables exposing capabilities of any CAS - These capabilities may be accessed by a remote client by passing meaningful self explanatory calls that must be understood and interpreted by the CAS. Depending on the capabilities of the underlying CAS, the CAS Server offers two exposing flavors:

1. Functionality made accessible by parsing and executing SCSCP [1] compliant calls. The SCSCP is a communication protocol specially designed for CAS to CAS communication. All messages exchanged using SCSCP are encoded using OpenMath [2] and have a specific format. The CAS Server is able to receive SCSCP compliant messages as

XML documents that are fed as plain strings to the CAS Server, by calling the correct operation of the CAS Server interface. The calls are parsed to extract useful meta-information and then are forwarded to the target CAS.

2. Functionality accessible through remote function invocation. This is an alternative suitable for the case when the CAS does not support SCSCP and OpenMath. Most of the CAS functionality is available by calling previously implemented functions grouped in function packages. These functions are either supplied by the standard libraries of the CAS or they are described by the CAS user.

The goal of the CAS Server is to enable users to expose such functions in order to make them available for remote invocation. When calling such functions remotely, the client of the CAS Server must supply in this case a correct function name and the list of its arguments. We must note that the only allowed type for input parameters and result type is string. For any other required type, the client must map/transform values to their string representation. The result obtained by calling the target function is the exact string returned by the CAS, and thus, the client is responsible for parsing the string result and for extracting the useful information. The CAS Server is an interface that offers a bridge between the client and the CAS through the interface of the Grid service. An important difference between this type of call and a call submitted through the command line interface of the target CAS is persistence. When working locally with a CAS such as GAP [3], using the command line interface, calling a function may lead to initialized variables that are stored in the memory of the CAS. A subsequent call in command line could potentially use the initialized values. This interactive behaviour is not available between two different calls to a CAS Server without additional intermediary steps that would store and resume a certain state. The calls to the CAS Server must be self explanatory and self contained.

Result notification call - Most of the symbolic computations require a long time to complete. For this reason, a non blocking call mechanism to the CAS Server is a better choice than a blocking call that waits for a certain task to be completed. The CAS Server makes sure to call a partner service ready to gather results as soon as they are obtained. The call back mechanism is activated in three situations: when the execution ended normally and a result is obtained, when the execution ends due to an encountered error, and when the result is manually set using a special task management mechanism implemented by the CAS Server.

Allow multiple CASs to be exposed through a single interface - It is very common that on a certain machine several CASs or versions of the same CAS exist. CAS Server allows exposure of several CASs using the same interface. This is achieved by publishing the CAS and, for a certain CASs, the methods/symbols that the CAS's instances are able to interpret/execute. When dealing with clusters, it is a rule of thumb that only one machine, usually the head node, is exposed and accessible for remote invocations. The rest of the cluster's nodes are able to communicate among them using local network calls, while communication to remote nodes is restricted, for security and consistency reasons. When properly installed and configured, a cluster of machines having installed CASs may be seen as a single machine, behind the interface of the CAS Server interface. While several nodes exists behind the interface, the cluster is seen from outside as a single processing element.

Allow a potential client to learn which CASs are installed and available through the CAS Server interface and which methods/symbols may be used in a submitted call - For security reasons not all functions/symbols implemented by a certain CAS are available to be used remotely. The administrator of the server is able to restrict the list to the ones that are safe/that are needed to be accessed remotely.

Implement a notification mechanism suitable for functionality discovery - In order to integrate CAS Server in a broader architecture that solves complex problems, an orchestration component must have access to meta-information regarding the

processing elements and the functionality they expose. The notification mechanism may be used in order to make sure that, as soon as a relevant characteristic of the CAS Server changes, all the orchestration nodes that require an update are notified. The notification message contains information about the CASs installed on the CAS Server, the methods/symbols implemented by the installed CASs that are available for remote invocation (in the case of OpenMath symbols, the list of symbols that may be used to formulate a call), the capabilities of the machines on which the CASs are installed.

Call resolving mechanism - It is not uncommon for OpenMath documents to use references to other OpenMath documents. When submitting a SCSCP call containing OpenMath references (OMRs), the target documents may be present on the machine to which the call is submitted but most often, they reside on other machines. CAS Server implements a resolving mechanism that makes sure that any references used in a SCSCP call are resolved, i.e. the OpenMath objects referenced by the OMRs are made available on the execution CAS Server. This is achieved by communicating with partner CAS Servers from which target nodes are extracted and retrieved.

Execution management - The capabilities that fall into this section may be more relevant when the CAS Server is integrated within an orchestrated architecture. Controlling the execution of a certain task requires several capabilities: pausing a task, resuming a task, cancelling a task, “a priori” result setting capability – the ability to specify the probable result of the executed task.

Installation Steps

Prerequisites

In order to successfully install CAS Server, several software packages several configurations must be done and several software packages must be already installed. The prerequisite environment consists of:

Operating System: any Linux distribution supported by a full installation of Globus 4.2 Grid middleware packages may be used. Alternatively, a Windows distribution may be used if the security features offered by the GSI security package of Globus Toolkit are not required. For the rest of the document we assume that a Globus Toolkit compliant Linux distribution is used.

Java SDK version 1.5 or later (<http://java.sun.com/javase/downloads>) – the Java Software Development Kit required for compiling and executing the CAS Server and other additional software packages such as Globus Toolkit.

Apache Ant version 1.6 or later (<http://ant.apache.org/>) - is required for building and installing the package.

Apache ActiveMQ version 5 (<http://activemq.apache.org/>) – these packages must be registered to the Java CLASSPATH environment variable and an ActiveMQ broker should be configured and started before starting any component of the CAS Server (see Section “*Installing the packages*”). It is used for interconnection of several CAS Server internal components.

Apache Tomcat version 5.5 (<http://tomcat.apache.org/>) – this is optional and is necessary only if you want to run the ActiveMQ broker embedded in the Tomcat Container

PostgreSQL database server version 8 or later (<http://www.postgresql.org/>) with support for PL/pgSQL – required for persistency by the CAS Server. Once installed, an SQL configuration script available within the CAS Server installation package will be executed in order to create the necessary database structure.

Globus Toolkit version 4.2 – the CAS Server interface consists of several Grid services exposed as WSRF services in the Globus container.

One or more Computer Algebra Systems for which the functionality should be exposed. If these systems shall be used as SCSCP servers then these features should also be installed and configured properly.

Package Content

The software package for the CAS Server contains the current release of the CAS Server and a number of required libraries included in the package for the convenience of the user. The main folder and files in the distribution are:

- `conf`: folder containing various configuration files
- `dist`: folder containing the JAR archives for the CAS Server's internal components
- `external`: folder containing packages external packages required by the CAS Server, included here for the convenience of the user
- `lib`: folder containing external JAR archives required by the CAS Server
- `projects`: folder containing the source-code for the internal components of the CAS Server and the build scripts
- `sql`: folder containing the PostgreSQL scripts for the system
- `test-files`: folder containing various files used during system testing
- `INSTALL`: readme file with installation instruction
- `install-activemq.sh` – bash script for installing Apache ActiveMQ in a Apache Tomcat Container
- `install-activemq.bat` – bat script for installing Apache ActiveMQ in a Apache Tomcat Container
- `install-cs.sh` – bash script for installing the CAS Server
- `install-cs.bat` – bat script for installing the CAS Server
- `log4j.properties` – logging properties file
- `run-cs-scheduler.sh` – bash script for running the Scheduler component of the CAS Server
- `run-cs-scheduler.bat` – bat script for running the Scheduler component of the CAS Server
- `run-cs-tests.sh` – bash script for running a simple test on the CAS Server
- `run-lregadmin.sh` – bash script for running the registry administration component of the CAS Server
- `run-lregadmin.bat` – bat script for running the registry administration component of the CAS Server
- `system.properties` – system properties file with various system configurations (see Section “CAS Server Configurations”)

Installing the Packages

The first step of the installation process of the CAS server component requires you to edit the file *system.properties*. Here you must set the hostname for your PostgreSQL database server, the user and the password necessary for connecting to this server and, if you intend to run the ActiveMQ broker from an Apache Tomcat Container, then you must also set the hostname and the port of your Tomcat Server.

Next you must install and configure ActiveMQ. We detail next only the installation in a local Apache Tomcat Container which can be done using the *install-activemq.bat* or *install-activemq.sh* scripts. (If you do not want to use Tomcat then just follow the installation

instruction received with Apache ActiveMQ.) The ActiveMQ message queues are seen as JNDI resources by Tomcat and so you have to specify the factories for these resources. In order to do this you must edit to Apache Tomcat configuration files as follows:

in *conf/server.xml* in the `<GlobalNamingResources>` element you must add the following child elements:

```
<Resource name="jms/ConnectionFactory" auth="Container"
  type="org.apache.activemq.ActiveMQConnectionFactory" description="JMS
  Connection Factory"
  factory="org.apache.activemq.jndi.JNDIReferenceFactory"
  brokerURL="broker:(tcp://localhost:61616)?persistent=false"
  brokerName="LocalActiveMQBroker" />
<Resource name="jms/LocalTaskScheduledQueue" auth="Container"
  type="org.apache.activemq.command.ActiveMQQueue"
  description="LocalTaskScheduledQueue"
  factory="org.apache.activemq.jndi.JNDIReferenceFactory"
  physicalName="LocalTaskScheduledQueue" />
<Resource name="jms/LocalTaskReadyQueue" auth="Container"
  type="org.apache.activemq.command.ActiveMQQueue"
  description="LocalTaskReadyQueue"
  factory="org.apache.activemq.jndi.JNDIReferenceFactory"
  physicalName="LocalTaskReadyQueue" />
```

in *conf/context.xml* in the `<Context>` element add the following child elements:

```
<ResourceLink global="jms/ConnectionFactory"
  name="jms/ConnectionFactory"
  type="org.apache.activemq.ActiveMQConnectionFactory" />
<ResourceLink global="jms/LocalTaskScheduledQueue"
  name="jms/LocalTaskScheduledQueue" type="javax.jms.Queue" />
<ResourceLink global="jms/LocalTaskReadyQueue"
  name="jms/LocalTaskReadyQueue" type="javax.jms.Queue" />
```

You install the CAS Server component by running the installation batch file *install-cs.bat* or *install-cs.sh*. Prior to running this file, the PostgreSQL server must be up and running. Start the GLOBUS Container.

Running the CAS Server

For starting the CAS Server several steps must be followed:

1. Start the Apache Tomcat server (only if the ActiveMQ message broker runs within the Tomcat Container):


```
$CATALINA_HOME/bin/startup.sh (Unix)
%CATALINA_HOME%\bin\startup.bat (Windows)
```

 or start the ActiveMQ broker as indicated in the Apache ActiveMQ documentation.
2. Start the SCIENCE scheduler using the *run-cs-scheduler.sh* or *run-cs-scheduler.bat* script.
3. If you want to use the SCSCP protocol start the SCSCP server (see CAS specific documentation).
4. Start the GLOBUS container (see Globus Toolkit documentation).
5. The CAS Server is now available at URL
http://localhost:<globus_port>/wsrf/services/science/GridCASFactoryService
6. If you want to send a task to the CAS Server you can use the *run-cs-tests.sh* script provided with this distribution. The script takes two arguments:
 - the URL of the CAS Server (as shown at point 5. above)
 - the path to a file containing the OpenMath encoding of the request. You can find examples of such files in the test-files directory (the *gcd-scscp.xml* contains a

SCSCP call for a service computing the GCD of two numbers and the *gcd-method.xml* contains a simple OpenMath-encoded call of the GAP Gcd function).

- e.g. `run-cs-tests.sh \`
`http://localhost:< port>/wsrf/services/science/GridCASFactoryService \`
`test-files/gcd-scscp.xml`
7. If you want to pause/resume/cancel one or more previously submitted tasks you can use the *run-cs-manage-tests.sh* script. The script takes three parameters:
 - the URL of the CAS Server (as shown at point 5. above)
 - the path to a file containing task IDs as returned by the *run-cs-tests.sh* script
 - one of the three characters: *p* = pause task, *r* = resume task or *c* = cancel task.
- e.g. `run-cs-manage-tests.sh \`
`http://localhost:< port>/wsrf/services/science/GridCASFactoryService \`
`task_file c`

CAS Server Configurations

Configuration File

Apart from the configuration files that are specific to Globus Toolkit 4.2 installation process, all the configuration files that will setup/fine tune the CAS Server component have the Java property format, i.e. a set of property value keys. The *system.properties* property file contains configuration settings for the CAS Server component configuration and for the visual application that allows administrating the CAS Server database after it is installed and running. Depending on the environment where the CAS Server is installed, for some of the keys present in the configuration file the supplied values must not be changed while for others, it is mandatory to supply correct values that correctly reflect the local installation parameters (bolded in the following description). For instance, it is possible to leave the value of the port used for communication between the CAS Server and its scheduling internal component unchanged, while it is mandatory to supply the correct password for connection to the PostgreSQL database. The CAS configuration file must contain the following keys and correspondent values:

database_drivername - the name of the driver that should be loaded to communicate with the database. Should be left unchanged unless other version than the one supplied with the CAS Server of the JDBC connector is used.

database_username – the name of the authorised user that must be used to connect to the PostgreSQL database.

database_password – the valid password associated with the authorised user that should be used to connect to the database

database_hostname – the URL designating the host where the PostgreSQL database was installed

database_name – the name of the database. The default name should be used unless a conflict exists with an already existing database

database_schemaname – the name of the PostgreSQL schema

database_max_connections_number – the maximum size of the connection pool of the database. A value of “0” permits any number of connections

database_max_connection_acquire_time - the maximum number of milliseconds to wait until a connection from the connection pool is allocated to current execution thread.

message_broker_url - the connection URL for the ActiveMQ message broker

message_connection_factory - the name of the ActiveMQ connection factory

ready_tasks_queue_name - the name of the ActiveMQ queue holding ready tasks

scheduled_tasks_queue_name - the name of the ActiveMQ queue holding scheduled tasks

max_wrapper_instances – this setting controls the maximum number of processes that should be used to start a CAS instance. This applies for non SCSCP calls when the target CAS is not able to act as a server so an instance of the CAS must be started in a separate process

resolver_root_files_directory - the root directory that should contain all the files that may be targeted by the CAS Server's OpenMath reference resolver. Any reference encountered while parsing a submitted call or any of the references that recurrently may be discovered following the encountered OMRs must reference an object described in the file within the target directory. The same rule applies to the resolve reference requests that are submitted by other CAS Servers. All targeted files must be in the scope of this directory.

resolver_service_URL – the URL of the OpenMath references resolver service. At this URL, clients may submit resolve requests for target nodes that are the scope of the current CAS Server.

resolver_file_transfer_root_URL – the template URL that must be used to determine the absolute URL of a file that is in the scope of the CAS Server. The URL of a file must comply to the transfer protocol used for transferring files from this CAS Server. By default, the pattern is set to comply with the URL format used by Grid FTP transfer protocol.

binary_transfer – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

block_size – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here: <http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

tcp_buffer_size – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

third_party_transfer – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

no_parallel_streams – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

data_channel_auth – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

concurrency – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

source_grid_subject – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

dest_grid_subject – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

transfer_all – setting used Reliable File Transfer service in Globus Toolkit (for

detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

no_retries – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here: <http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

security_mechanism – setting used Reliable File Transfer service in Globus Toolkit (for detailed descriptions look here:

<http://www.globus.org/toolkit/docs/4.2/4.2.1/data/rft/user/#rft-cmd>)

gridftp_link – the format of the GridFTP server URL. By default this is `gsiftp://host:2811`. If your GridFTP is listening to another port then you may have to adjust the URL correspondingly.

cas_server_service_url – the URL to the CAS Server main service. This URL is used to formulate update messages sent to interested service discovery registries.

A Simple Client

Local Registry Administration Application

The Administrative application allows the administrator of the node where the CAS Server was installed to set properties that have a direct impact on the functionality of the CAS Server. Successful exposure and advertising of the functionality implemented by the server must follow a sequence of steps.

Adding a New Execution Node to the CAS Server

The execution infrastructure that the CAS Server is able to expose, at hardware level, may be compound of a single machine or even multiple machines. The CAS Server's information system must be aware of the existence of every machine that is part of the infrastructure. The machines cannot be contacted directly by a remote client and the tasks that must be run by a certain machine and all the communication logic is managed by the CAS Server. The information describing the capabilities of every machine has a direct impact on the internal algorithms that elect the most suitable machine to execute a certain task. This information may have a similar impact at the client side because a certain client may be aware of existing capabilities of several CAS Servers and may choose to submit a task to be executed by a certain CAS Server based on its capabilities.

The details that must be supplied when registering a machine to the CAS Server are:

Name – this may be any string meaningful for the administrator. It should be unique in the scope of the same CAS Server.

CPU Power – the computational power of the processors installed on the machine. This value must be specified in MHz. If multiple processors exist, this value represents the value corresponding of a single processor.

Maximum CPU Power – the maximum CPU power that may be allocated to solve tasks received from the CAS Server. This value should be expressed in MHz.

Processors Nr – total number of processors available on the machine.

Total RAM – the total RAM installed on the machine. This value must be expressed in Mbytes.

Available RAM – the total RAM available for execution. This value must be expressed in Mbytes.

Available HDD Space – the total storage space available to the CAS Server. This

value must be expressed in Mbytes.

Comments – additional comments relevant for the described machine.

The form where all these details must be inserted is presented in Figure 1.

Figure 1 Insert New Machine form

Once registered, the details regarding a certain machine are relevant only if on this machine are installed CASs implementing functionality that must be made available to remote clients. These details will never be advertised “as is”, only relevant information accompanying a CAS description will be available at the client side. For instance, the name of the machine and comments are not relevant for a client and will never be exposed.

Adding a New CAS to the CAS Server

The CAS instances are actual interpreters/execution software that handle the tasks submitted by the client. Any task submitted by a client is atomic in the sense that it will be executed by a CAS instance that is able to understand all the method/symbols used to describe the task or none at all.

The CAS Server is able to submit a task only if it has detailed information about the CAS that shall run the task: the type of the CAS, the machine where it is installed locally, the list of methods and symbols that it supports. After registering a certain machine to the CAS server, the administrator must register the CASs installed on that machine to the CAS Server. Registering a certain CAS to the CAS Server requires that several pieces of information are provided:

CAS Name – the name of the CAS. This value must be unique in the scope of the machine. A good naming scheme should include the type of the CAS (e.g. GAP, KANT) the version of the CAS and a discriminator string that would differentiate this CAS from others having the same type and version. This unique name enables the client to specify exactly what instance of CAS should be used to solve a certain task. For example: If “GAP 4.0 - 1” and “GAP 4.0 - 2” are registered to the CAS Server, a remote user that wants to target the second one will have to specify as the target CAS the full name of the CAS. If it only requires an instance of GAP 4.0, the name of the targeted CAS to execute the task will be specified at the client side as “GAP 4.0” and the CAS server will have the liberty to elect whichever it considers more appropriate.

Installed On – specifies the machine where the CAS is installed. The correct machine

must be selected from the list of already registered machines.

Description – a small comment regarding the CAS. This value will be advertised and is visible to remote clients.

Example – an example of CAS usage, if appropriate.

CAS location – the path to the executable script that starts the CAS as a new process. This value is used when the CAS does not support SCSCP and the CAS Server must start an instance of the CAS in order to pass it the task to solve. It is thus used when the task submitted by the client is not formatted as a SCSCP call, but rather as a method call.

Library Dir – the location of the additional libraries that the CAS may need to load in order to solve the task.

Machine Address – the hostname of the machine where the CAS running as a SCSCP is installed.

TCP/IP Port – the port where the CAS is listening as a SCSCP server. Usually this is 26133.

Can Pause – valid values are “TRUE” or “FALSE” and specifies if the CAS instance is able to pause the execution of a task and resume it later. This feature is not available to any CAS but it may be implemented by CASs designed to handle long running tasks.

The form through which the administrator is able to specify all the above details is presented in Figure 2

The screenshot shows a window titled 'Local Registry Administrator' with a menu bar containing 'Insert New', 'Lists', 'Export', 'Reports', and 'Notify'. The main area is titled 'Insert CASs' and contains the following form elements:

- CAS Name:** A text input field.
- Installed On:** A list box with three items: 'wn02000', 'wn01', and 'wn03'.
- Description:** A large text area.
- Example:** A large text area.
- CAS Location:** A text input field.
- Library Dir:** A text input field.
- Machine Address:** A text input field.
- TCP/IP port:** A text input field.
- Can Pause:** A text input field.

At the bottom of the form are three buttons: 'Submit', 'Close', and 'Reset'.

Figure 2 The Insert New CAS form

From the details specified above, only several details are published to a potential client: the CAS name, the description and the example. The rest of the details remain hidden and are only visible to the administrator.

Once a CAS is registered to the CAS Server, the administrator must specify which methods and/or OpenMath symbols are available for remote invocation. This operation must be accomplished in two steps: register a certain function/OpenMath symbol to the CAS server and, in the second step, specify which CAS implements which function/symbol.

Example. We assume that we have a GAP instance installed on the local machine at the location `/opt/GAP/`. The executable script that starts the GAP instance is located at `/opt/GAP/bin/gap.sh`. Therefore, to use that GAP instance with the CAS Server, we first need to complete the following information:

CAS Location: `/opt/GAP/bin/gap.sh`

Library dir : `/opt/GAP/` (the GAP system looks by default in the *lib* directory relative to the specified *Library dir* path)

Machine address: `127.0.0.1`

TCP/IP port: - not required for this setting, only if the GAP instance supports SCSCP

Adding a Function to the CAS Server Registry

The simplest approach to exposing functions implemented by the installed CASs is based on the fact that a certain function that is part of a certain function package may be implemented by different versions of the same CAS, having the same list of input parameters and the same provided functionality. Thus, the functionality exposed by a certain CAS may be expressed based on the functions it implements. Unfortunately, the CAS Server is not able to discriminate between two functions that have the same name, are part of the same package but have different signature (i.e. different list of input parameters). This is not a shortcoming because it is always possible to implement a bridge function, with a different name, in order to avoid the function name collision.

Adding a new function to the list of functions available through the CAS Server interface requires as a first step to complete several details regarding the function:

Function Name - the name of the function.

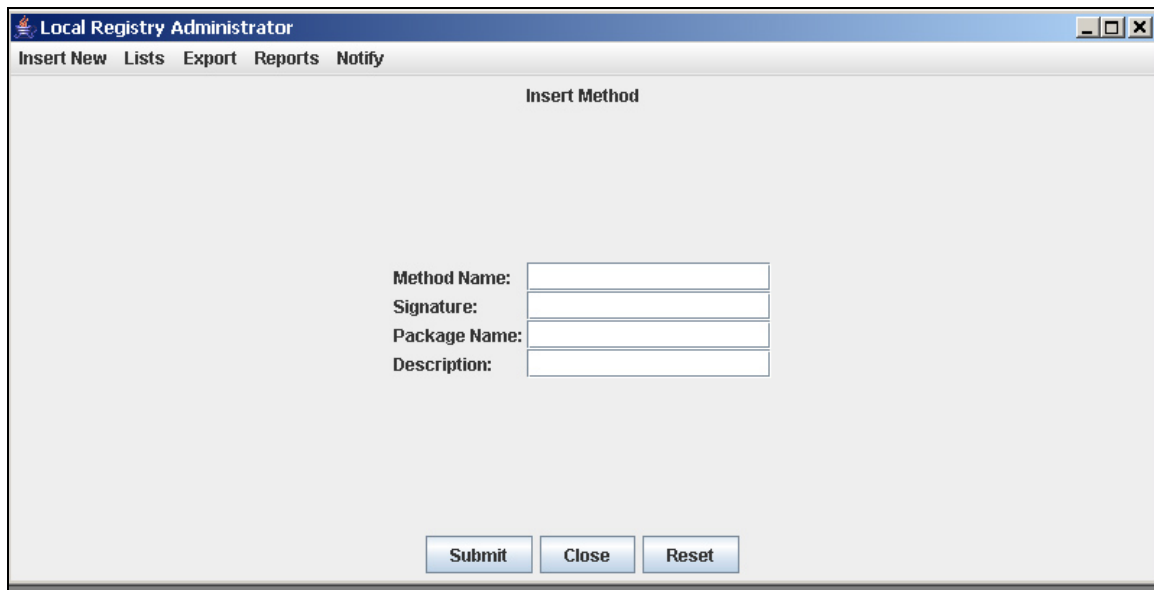
Signature – the returned type of the function, the name of the function and the list of parameters. This information is important in the discovery process if the client wants to see the actual list of parameters that the function expects.

Package Name – the name of the package that the function is part of. We must note that it is not allowed for two functions with the same name to be part of the same package.

Description – a short description of the function. This information is published to the client and may be useful in the discovery process.

The form that permits the administrator to complete these details is presented in

Figure 3.



The screenshot shows a web application window titled "Local Registry Administrator". It has a menu bar with "Insert New", "Lists", "Export", "Reports", and "Notify". The main content area is titled "Insert Method". It contains four labeled text input fields: "Method Name:", "Signature:", "Package Name:", and "Description:". At the bottom of the form are three buttons: "Submit", "Close", and "Reset".

Figure 3 Insert New Function form

A similar approach must be followed in order to register a new OpenMath Symbol.

Adding an OpenMath Symbol to the CAS Server Registry

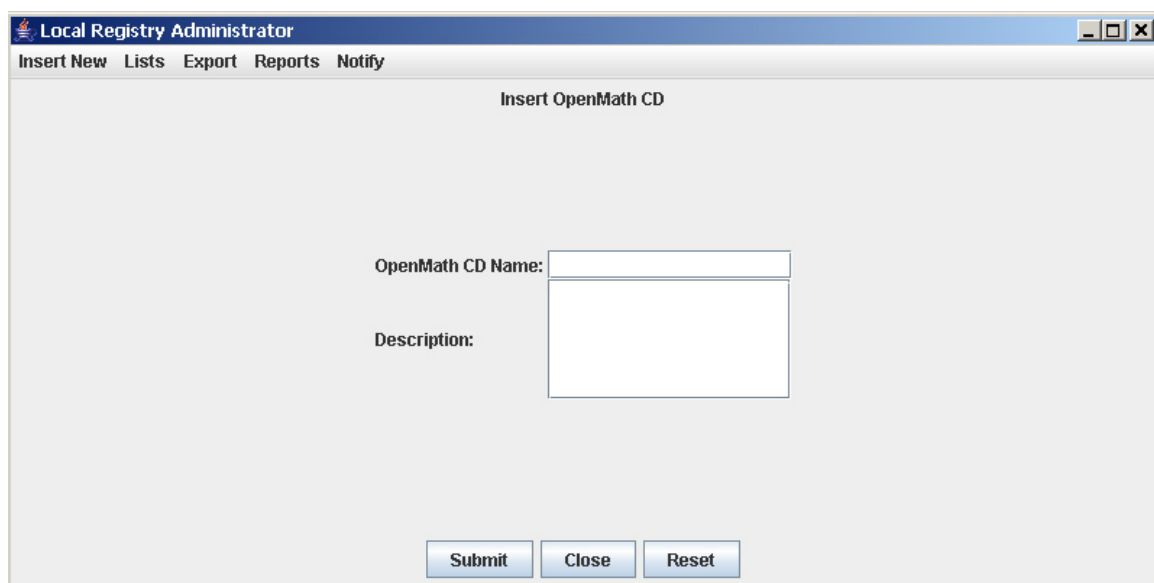
Adding a new OpenMath symbol to the registry may be achieved in two steps. Every OpenMath symbol must be part of a package of symbols, called OpenMath content dictionary (OpenMath CDs). Before adding the symbol, the OpenMath CD must be already registered to the CAS Server.

In order to register a new OpenMath CD, several details must be supplied:

OpenMath CD Name – the name of the OpenMath CD. This value must be unique in the scope of the CAS Server registry.

Description – a short description of the OpenMath CD.

The form that allows the administrator to add a new CD is presented in Figure 4.



The screenshot shows a web application window titled "Local Registry Administrator". It has a menu bar with "Insert New", "Lists", "Export", "Reports", and "Notify". The main content area is titled "Insert OpenMath CD". It contains two labeled text input fields: "OpenMath CD Name:" and "Description:". At the bottom of the form are three buttons: "Submit", "Close", and "Reset".

Figure 4 Insert New OpenMath CD form

The second step to register the new symbol to the registry requires that the administrator provides the following information:

OpenMath Symbol Name – the name of the OpenMath symbol. This name must be unique in the scope of a certain OpenMath CD.

OpenMath CD – the OpenMath CD to which the symbol belongs.

Description – a short description of the OpenMath symbol. This description is visible by the clients of the CAS Server.

The form that allows the administrator to add a new symbol is presented in Figure 5.

The screenshot shows a web application window titled "Local Registry Administrator". It has a menu bar with "Insert New", "Lists", "Export", "Reports", and "Notify". The main content area is titled "Insert OpenMath Symbol" and contains three input fields: "OpenMath Symbol Name:" with the value "base", "OpenMath CD:", and "Description:". At the bottom are three buttons: "Submit", "Close", and "Reset".

Figure 5 Insert New OpenMath Symbol form

Adding Remote Registries

Before describing the process of assigning a function/OM Symbol to a certain CAS, thus marking that the specified CAS implement the designated function/symbol we must explain the two discovery processes available when working with a CAS Server. One way to learn which CASs are installed and are made available through the CAS Server and the functions/symbols exposed for a certain CAS is to call the informative operations available on the CAS Server Grid Service interface. Another way, more useful for automated CAS Server orchestration is a notification mechanism that sends complete information regarding the functionality exposed by the CAS Server. Any change that is made in the functionality exposed by the CAS Server must be notified to the interested third party discovery registries. These registries must provide a Web service interface that a CAS Server is able to call. This interface will supply to other registries the list of changes that were made since during the last update. We call these other registries *Main Registries*.

The list of main registries where the CAS Server must send notification messages is controlled through the administrator list interface. To add a new main registry to the list of recognised registries, the administrator must supply:

Registry URL – the URL of the service where the update message must be sent

Adding a new main registry may be done by completing the form presented in Figure 6.

The image shows a web application window titled "Local Registry Administrator". It has a menu bar with "Insert New", "Lists", "Export", "Reports", and "Notify". The main content area is titled "Insert Main Registry" and contains a text input field labeled "Registry URL:". At the bottom of the form are three buttons: "Submit", "Close", and "Reset".

Figure 6 Insert New Main Registry form

Linking a Function to a CAS

As said above, the functionality exposed by the CAS Server may be characterised in terms of the functions that are available for remote invocation or through the list of OpenMath symbols that may be used to describe a SCSCP call. The process of publishing a certain function (or method) to be available for remote calls consists of two main steps: marking that a certain function is implemented by a certain CAS and make sure that the correct main registry is aware about the availableness status of the function. Once a function is linked to a certain CAS and is published to a main registry, the function may no longer be removed from the list of functions implemented by the CAS. In turn, it may be deactivated, obtaining the similar effect. The form that lets the administrator link a function to a certain CAS and to choose the main registries where this information must be published is presented in Figure 7.

Local Registry Administrator

Insert New Lists Export Reports Notify

Publish Method

Methods:

Method Name	Method Package	Method Signature	Description
GCD	pkg	signature	descr

CASs:

CAS Name	Installed On	Implements Sel. Met...	Description
GAP	wn01	true	descr
testCas1	wn03	true	asdf
testCAS	wn01	false	asdf

Add to CAS Delete from CAS

Registries:

Registry URL	Is Visible To
http://science.ieat.ro/secondregistry	true
http://science/fistregistry	false

Add to registry Delete from Registry

Figure 7 Link Function to CAS and Advertise Management form

By selecting a method (or function) from the list of “Methods” table, the CASs table is populated with the list of CASs registered to the database. The information in the CASs table is sorted by the link status between the CAS and the function. If the function is implemented by the CAS, the name of the CAS is coloured with green. Otherwise, it is coloured in blue. Pressing “Add to CAS” and “Delete from CAS” buttons registers/unregisters the method to the selected CAS.

Advertising that a certain function is implemented by a certain CAS to a certain Main Registry is achieved by pressing the “Add to registry” button. For a published function, pressing the “Delete from Registry” button marks the function as unavailable in the designated main registry.

Note. The notification of a change in the details regarding a certain function, its available state for a certain CAS or for a certain main registry is only changed when the administrator especially requires the administrative application to send the notification. The administrator requests this operation only when all modifications of the current session are operated for the CAS Server. Sending a notification update occurs when the “Notify” menu item is selected.

For efficiency reasons, a main registry receives notifications with only necessary and sufficient information reflecting the changes operated in the notifying CAS Server since the last successful update.

Linking an OpenMath Symbol to a CAS and Main Registry Publication

The CAS may be able to parse calls described as OpenMath objects. Usually, a certain CAS is able to understand several symbols and, if more than one CAS is installed on the CAS Server, it is possible that the same symbol is supported by multiple CASs. The administrator may control which symbols are supported by a certain CAS and to which main registries this information must be published. The publishing process is similar to the one used to link and advertise methods of a CAS. The form that allows registering an OpenMath symbol as supported by a certain CAS and main registry publishing is presented in Figure 8.

Local Registry Administrator

Insert New Lists Export Reports Notify

Publish Symbol

OM Symbols:

Symbol Name	Symbol CD	Description
gcd	base	descr1
gcd1	base	asefddas

CASs:

CAS Name	Installed On	Implements Sel. Sy...	Description
----------	--------------	-----------------------	-------------

Add to CAS Delete from CAS

Registries:

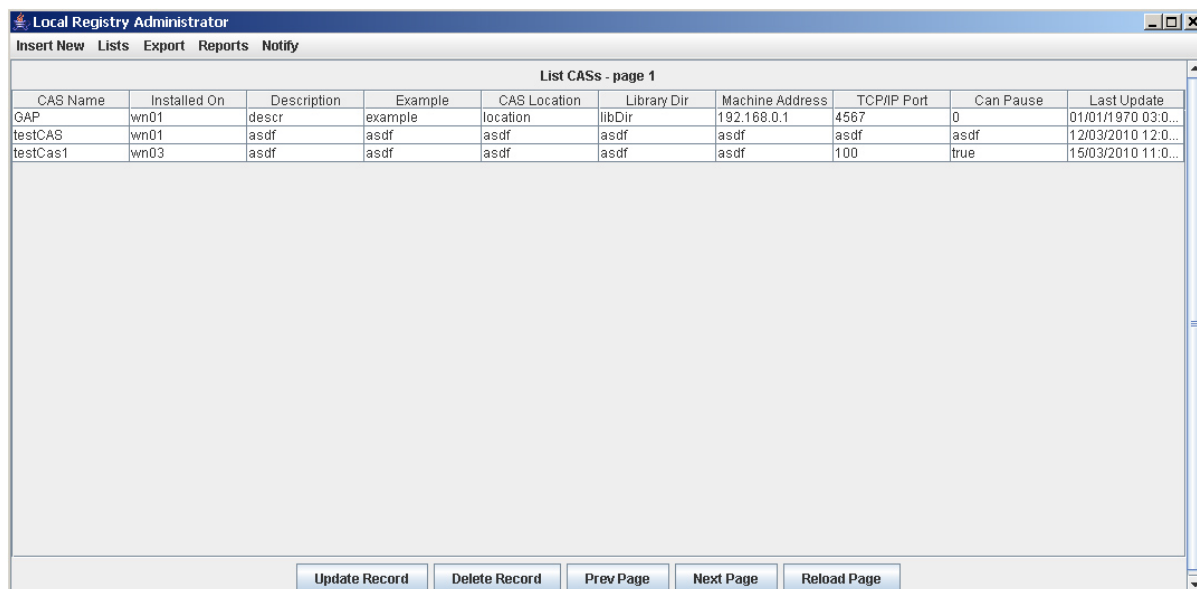
Registry URL	Is Visible To
--------------	---------------

Add to registry Delete from Registry

Figure 8 Link OpenMath Symbol to CAS and Advertise Management form

Consulting the CAS Server Registry

The administrative application implements browsing capabilities of the basic information stored in the registry and reports the tasks and related details. The basic information is available by accessing the desired listing from the “Lists” menu. An example of such listing is presented in Figure 9.



The screenshot shows the 'Local Registry Administrator' window. It has a menu bar with 'Insert New', 'Lists', 'Export', 'Reports', and 'Notify'. The main area displays a table titled 'List CASs - page 1'. The table has 10 columns: CAS Name, Installed On, Description, Example, CAS Location, Library Dir, Machine Address, TCP/IP Port, Can Pause, and Last Update. There are three rows of data. Below the table, there are five buttons: 'Update Record', 'Delete Record', 'Prev Page', 'Next Page', and 'Reload Page'.

CAS Name	Installed On	Description	Example	CAS Location	Library Dir	Machine Address	TCP/IP Port	Can Pause	Last Update
GAP	wn01	descr	example	location	libDir	192.168.0.1	4567	0	01/01/1970 03:0...
testCAS	wn01	asdf	asdf	asdf	asdf	asdf	asdf	asdf	12/03/2010 12:0...
testCas1	wn03	asdf	asdf	asdf	asdf	asdf	100	true	15/03/2010 11:0...

Figure 9 The list of CASs registered to the CAS server

As the above listing shows, selecting the CASs listing will show the list of CASs previously registered to the CAS Server and their related information. Selecting the “Update Record” button may open in this case two different forms. If a row is selected and the selected column is other than “Installed On” column, the form that allows editing the CAS record will be opened. If the “Installed On” cell is selected, the application will open the form that allows the administrator to edit the details regarding the machine where the CAS is installed. The rest of the buttons functionality is straightforward:

“Delete Record” – deletes the selected record, regardless the column that is currently selected.

“Prev Page” – loads the previous listing page, if the current page is not the first page.

“Next Page” – loads the next listing page, if the current page is not the last page.

“Reload Page” – reloads the current page by connecting to the underlying database.

Advertising messages to remote registries

The CAS Server may be integrated into a bigger architecture to which the CAS Server represents an execution node. Automated election of execution nodes must be based on real time information that describes the capabilities and status of the execution nodes. The interface of the CAS Server exposes operations that allow a remote client to find all these details. In order to save communication effort, a better approach is to allow CAS Servers to advertise only those changes that are relevant for a certain discovery registry, usually hosted at the same location with the automated workflow engine.

When using *Local Registry Administrator* application, changes made to CAS Server status are stored into a local database but they are not automatically advertised to the intended registries. **After the administrator operates all the changes in the CAS Server status, he**

must activate the synchronization to remote registries explicitly. This operation is achieved by selecting the “Notify” menu.

When the “Notify” menu item is selected, specific messages are constructed and sent to the registries via Web services calls. **The remote registry – the recipient of the update message - must expose an operation that is ready to receive update messages from CAS Servers.**

Based on the current status of the CAS Server and the actual information successfully submitted in a previous synchronization call, the custom message tailored for every registry may contain:

1. If the registry was newly added to the CAS Server and no synchronization messages were exchanged between the CAS Server and the remote registry, the message contains:
 - the list of CASs exposed by CAS Server
 - for every CAS, the capabilities of the machine where a particular CAS is installed
 - for every CAS, complete details about the functions and OpenMath symbols that are supported/implemented by every CAS
2. If the main registry was already successfully informed by existing characteristics of the CAS Server but in the meanwhile several characteristics were updated by the CAS Server’s administrator, it contains:
 - full details of the entity that was changed and identification details regarding the related CAS. For instance, if the details regarding a certain function were changed, the update message contains the details regarding the method and the list of the CAS that supports it.

Note. Each entity described in an update message, i.e. CAS, method, OpenMath symbol, etc..., is identified by a unique key that does not change over time. If a CAS named GAP1 is installed on a machine and it is registered through the *Local Registry Administrator* application, its key is advertised to all related discovery registries. If at a later time the administrator of the CAS Server decides to uninstall the GAP1 instance and add a new CAS called GAP2, it should register the adding the new CAS and not by changing the name of the CAS from GAP1 to GAP2 in the *Local Registry Administrator*. With the latter approach taken, the information in the database will be inconsistent and for all tasks that were run on the GAP1 instance the system will indicate that they were run on the GAP2 instance.

Changes update message structure

The update message sent by the CAS Server to a certain registry contains only those pieces of information that are relevant for the registry to which the update is sent. It contains details regarding the CASs, symbols and functions that are related to the registry and that were updated since the last successful update.

The interface of the Web service through which the discovery registry accepts update from CAS Servers must have the following signature:

```
updateRegistryInfo(String CASServerURL, String updateMessage)
```

where the *CASServerURL* represents the URL address of the CAS Server sending the update message. The *updateMessage* parameter must be an XML document having the following structure:


```

<!ELEMENT server ((machine*, cas+, method*, symbol,
                    casmethod,cassymbol, symbolcd) >

<!ELEMENT machine(machineid, cpupower, availcpupower,
                    processornr, totalram, availram,
                    availdisk )>
<!ELEMENT machineid (#CDATA)>
<!ELEMENT cpupower (#CDATA)>
<!ELEMENT availcpupower (#CDATA)>
<!ELEMENT processornr (#CDATA)>
<!ELEMENT totalram (#CDATA)>
<!ELEMENT availram (#CDATA)>
<!ELEMENT availdisk (#CDATA)>

<!ELEMENT cas(casid, machineref ?, casname, casactive,
               casdescription, casexample,updated)>
<!ELEMENT casid (#CDATA)>
<!ELEMENT machineref (#CDATA)>
<!ELEMENT casname (#CDATA)>
<!ELEMENT casactive (#CDATA)>
<!ELEMENT casdescription (#CDATA)>
<!ELEMENT casexample (#CDATA)>
<!ELEMENT updated (#CDATA)>

<!ELEMENT method(methodid, methodname, methodpackage,
                  methodsignature, methoddescription )>
<!ELEMENT methodid (#CDATA)>
<!ELEMENT methodname (#CDATA)>
<!ELEMENT methodpackage (#CDATA)>
<!ELEMENT methodsignature (#CDATA)>
<!ELEMENT methoddescription (#CDATA)>

<!ELEMENT symbol(symbolid, symbolname, symbolcdid,
                  symboldescription, updated)>
<!ELEMENT symbolid (#CDATA)>
<!ELEMENT symbolname (#CDATA)>
<!ELEMENT symbolcdid (#CDATA)>
<!ELEMENT symboldescription (#CDATA)>
<!ELEMENT updated (#CDATA)>

<!ELEMENT casmethod(casmethodcasid, casmethodmethodid,
                    casmethodactive, updated)>
<!ELEMENT casmethodcasid (#CDATA)>
<!ELEMENT casmethodmethodid (#CDATA)>
<!ELEMENT casmethodactive (#CDATA)>
<!ELEMENT updated (#CDATA)>

<!ELEMENT cassymbol(cassymbolcasid, cassymbolsymbolid,
                    cassymbolactive, updated )>
<!ELEMENT cassymbolcasid (#CDATA)>
<!ELEMENT cassymbolsymbolid (#CDATA)>
<!ELEMENT cassymbolactive (#CDATA)>
<!ELEMENT updated (#CDATA)>

<!ELEMENT symbolcd (symbolcdcdid, symbolcdname)>
<!ELEMENT symbolcdcdid (#CDATA)>
<!ELEMENT symbolcdname (#CDATA)>

```

NOTE. The *updated* node had as a text node child one of the values „true“ or “false” that indicate if the information described by the corresponding node of the updated node is present in the document because the information was update or only because some information regarding a referenced node was updated. For instance, an *updated* node of a *symbol* node having the value of „false“ indicate that the details described by the *symbol* node were not modified but the corresponding *symbolcd* node contains updated data regarding the OpenMath symbol’s CD.

Interfaces of CAS Server Service

Task Management Interface

The core functionality of the CAS Server relates to enabling a client to submit symbolic computation calls that must be solved by the computational infrastructure offered by the CAS Server. To achieve this goal, there are two main sets of functionalities. Before submitting any task to the CAS Server, the client must be sure that the CAS Server is really able to respond to the request. The details regarding functionality offered by the CAS Server may be obtained by calling informative operation from the interface. The second set of operations implements means that enable the client to submit calls and to control the state of the execution.

Due to its advantages, the CAS Server’s interface is implemented using the Grid services technology supplied by the Globus Toolkit 4.2. A Grid service is a special type of Web service that offers, among other features, built in support for state-full information access. Grid services are built around the concept of a resource, as described the WS-Resource OASIS specification [4].

CAS Server is implemented using the resource factory pattern. This means that a special factory service must be invoked by the client when a new call must be submitted. As a result of the call, the client receives a special resource identifier, Endpoint Reference (EPR), which may be later used to control the execution status.

Grid Factory Service Interface

The core functionality of the CAS Server relates to enabling a client to submit symbolic computation calls that must be solved by the computational infrastructure offered by the CAS Server. To achieve this goal, there are two main sets of functionalities. Before submitting any task to the CAS Server, the client must be sure that the CAS Server is really able to respond to the request. The details regarding functionality offered by the CAS Server may be obtained by calling informative operation from the interface. The second set of operations implements means that enable the client to submit calls and to control the state of the execution.

Due to its advantages, the CAS Server’s interface is implemented using the Grid services technology supplied by the Globus Toolkit 4.2. A Grid service is a special type of Web service that offers, among other features, built in support for state-full information access. Grid services are built around the concept of a resource, as described the WS-Resource OASIS specification [4].

CAS Server is implemented using the resource factory pattern. This means that a special factory service must be invoked by the client when a new call must be submitted. As a result of the call, the client receives a special resource identifier, Endpoint Reference (EPR), which may be later used to control the execution status.

The Resource property associated with the Grid Factory Service has the following structure:

```
<serverDescription>
  <exposedCass>
  </exposedCass>
  <staticProperties>
  </staticProperties>
  <dynamicProperties>
  </dynamicProperties>
</serverDescription>
```

The service interface consist of default operations implemented by Gobus Toolkit that offer the client a certain type of functionality and operations implemented especially for the purposes of the CAS Server.

The list of operations that allow the remote user to learn details regarding the capabilities of the CAS Server are:

- **String getInstalledCASs(String CASServerName)** – returns the list of the CASs installed on the CAS Server having the name
- **String getAllSupportedSymbols()** – returns the list of supported OM Symbols by the CAS Server, regardless the CAS that implements it
- **String getAllSupportedFunctions()** – returns the list of functions supported by any CAS installed on the CAS Server
- **String getCASsSupportingFunction(Sting functionName, String package)** – returns the list of CAS that implement the function given as a parameters
- **String getCASsSupportingSymbol(Sting omSymbol, String omcd)** – returns the list of CASs that support the OM Symbols that have the name *omSymbol* and are part of the OM Cd *omcd*
- **String getFunctionsMatch(Sting match)** – get the list of functions that are implemented by a CAS installed on the CAS Server and that have as a substring of their name or of their package name the string given as parameter.
- **String getSymbolsMatch(Sting match)** – get the list of OM symbols that are supported by a CAS installed on the CAS Server and that have as a substring of their name or of their Cd name the string given as parameter.
- **String getSupportedFunctions(Sting casName)** – the list of functions that are implemented by the CAS specified through the given parameter
- **String getSupportedSymbols(String casName)** – returns the list of supported OM Symbols by the CAS having the name *casName*

All operations specified above return results formatted as strings. The results are well formed XML documents that contain the actual relevant information, depending on the operation being invoked.

The structure of the *getInstalledCASs* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT cass (cas+) >
```

```
<!ELEMENT cas (casname, casdescription, casexample, cascanpause,
               cpupower, availablecpupower, nrofprocessors,
               totalram, availableram, availablediskspace)>
<!ELEMENT casname (#CDATA)>
<!ELEMENT casdescription (#CDATA)>
<!ELEMENT casexample (#CDATA)>
<!ELEMENT cascanpause (#CDATA)>
<!ELEMENT availablecpupower (#CDATA)>
<!ELEMENT nrofprocessors (#CDATA)>
<!ELEMENT totalram (#CDATA)>
<!ELEMENT availableram (#CDATA)>
<!ELEMENT availablediskspace (#CDATA)>
```

The structure of the *getAllSupportedSymbols* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT symbols (symbol+) >
<!ELEMENT symbol (symbolname, symbolcd, symboldescription)>
<!ELEMENT symbolname (#CDATA)>
<!ELEMENT symboldescription (#CDATA)>
```

The structure of the *getAllSupportedFunctions* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT methods (method+) >
<!ELEMENT method (methodname, methodsignature, methodpackage,
                  methoddescription)>
<!ELEMENT methodname (#CDATA)>
<!ELEMENT methodsignature (#CDATA)>
<!ELEMENT methodpackage (#CDATA)>
<!ELEMENT methoddescription (#CDATA)>
```

The structure of the *getCASsSupportingFunction* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT cass (cas+) >
<!ELEMENT cas (casname, casdescription, casexample, cascanpause,
               cpupower, availablecpupower, nrofprocessors,
               totalram, availableram, availablediskspace)>
<!ELEMENT casname (#CDATA)>
<!ELEMENT casdescription (#CDATA)>
<!ELEMENT casexample (#CDATA)>
<!ELEMENT cascanpause (#CDATA)>
<!ELEMENT availablecpupower (#CDATA)>
<!ELEMENT nrofprocessors (#CDATA)>
<!ELEMENT totalram (#CDATA)>
```

```
<!ELEMENT availableram (#CDATA)>
<!ELEMENT availablediskspace (#CDATA)>
```

The structure of the *getCASsSupportingSymbol* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT cass (cas+) >
<!ELEMENT cas (casname, casdescription, casexample, cascanpause,
               cpupower, availablecpupower, nrofprocessors,
               totalram, availableram, availablediskspace)>
<!ELEMENT casname (#CDATA)>
<!ELEMENT casdescription (#CDATA)>
<!ELEMENT casexample (#CDATA)>
<!ELEMENT cascanpause (#CDATA)>
<!ELEMENT availablecpupower (#CDATA)>
<!ELEMENT nrofprocessors (#CDATA)>
<!ELEMENT totalram (#CDATA)>
<!ELEMENT availableram (#CDATA)>
<!ELEMENT availablediskspace (#CDATA)>
```

The structure of the *getFunctionsMatch* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT methods (method+) >
<!ELEMENT method (methodname, methodsignature, methodpackage,
                  methoddescription)>
<!ELEMENT methodname (#CDATA)>
<!ELEMENT methodsignature (#CDATA)>
<!ELEMENT methodpackage (#CDATA)>
<!ELEMENT methoddescription (#CDATA)>
```

The structure of the *getSymbolsMatch* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```
<!ELEMENT symbols (symbol+) >
<!ELEMENT symbol (symbolname, symbolcd, symboldescription)>
<!ELEMENT symbolname (#CDATA)>
<!ELEMENT symboldescription (#CDATA)>
```

The structure of the *getSupportedFunctions* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```

<!ELEMENT casmethods (casname, method+) >
<!ELEMENT casname (#CDATA)>
<!ELEMENT method (methodname, methodsignature, methodpackage,
                  methoddescription)>
<!ELEMENT methodname (#CDATA)>
<!ELEMENT methodsignature (#CDATA)>
<!ELEMENT methodpackage (#CDATA)>
<!ELEMENT methoddescription (#CDATA)>

```

The structure of the *getSupportedSymbols* operation's response

The string returned by invoking the operation of the service is an XML document with the following structure:

```

<!ELEMENT cassymbols (casname, symbol+) >
<!ELEMENT casname (#CDATA)>
<!ELEMENT symbol (symbolname, symbolcd, symboldescription)>
<!ELEMENT symbolname (#CDATA)>
<!ELEMENT symboldescription (#CDATA)>

```

Submitting a New Task and Task Management

SCSCP Call

One of the goals of the SCIENCE project is to develop a protocol, namely SCSCP that would enable CASs to interact using a standard communication model. According to SCSCP specification, any message exchanged between CASs should be a valid OpenMath object describing the call and potentially meta-data regarding the call.

Formulating a call request to a CAS implementing SCSCP starts from the basic idea that algorithms regular functions implemented a CAS may be associated to a custom defined OpenMath symbol. Therefore, any OpenMath object that would use the associated symbol would in fact request that the function is executed.

```

1. <OMOBJ>
2.   <OMATTR>
3.     <OMATP>
4.       <OMS cd="scscpl" name="call_ID"/>
5.       <OMSTR>anid</OMSTR>
6.     </OMATP>
7.     <OMA>
8.       <OMS cd="scscpl" name="procedure_call"/>
9.       <OMA>
10.        <OMS cd="SCSCP_transient_1" name="Factorial"/>
11.        <OMI> 10</OMI>
12.      </OMA>
13.    </OMA>
14.  </OMATTR>
15. </OMOBJ>

```

The above call represents a simple example. The OpenMath symbol used at *line 8*. notifies the CAS parsing the call that this is a remote call that targets a function implemented by the

CAS. At *line 10*, the message clearly specifies the OpenMath symbol that identifies the exact function that should be called, and further, it states that the simple OpenMath object `<OMI>10</OMI>` should be passed as a parameter. Since the CAS is able to identify the correct function to call based on the above information, the call is executed and the result of the computation is returned.

Using OpenMath for data encoding ensures interoperability between different types of CASs or even different versions of the same CAS. It must be stated though that the CAS can understand completely a call if it understands all the OpenMath symbols that are used to describe the call.

Remote Function Call

Remote function call is an alternative to standard SCSCP call supplied for commodity reasons. This type of invocation may be useful if the target CAS does not implement the SCSCP protocol and does not have built in support for OpenMath. Usually, such CAS instances may be started as a separate process and the required interaction with the process may be carried out through the standard input/output streams. In this situation using the CAS functionality is similar to direct interaction through the command line interface of the CAS. The CAS Server acts as a communication bridge between the remote client and the CASs installed on the local machine.

The correct formulated call must comply with the following DTD structure:

```
<!ELEMENT OMOBJ (OMA) >
<!ELEMENT OMA (OMS, OMSTR,OMSTR,OMSTR*)>
<!ELEMENT OMS (#CDATA)>
<!ATTLIST OMS cd CDATA #FIXED "casall1">
<!ATTLIST OMS name CDATA #FIXED "procedure_call">
<!ELEMENT OMSTR (#CDATA)>
```

An example of such call might be:

```
<OMOBJ>
  <OMA>
    <OMS cd="casall1" name="procedure_call"/>
    <OMSTR>gcd</OMSTR>
    <OMSTR>default</OMSTR>
    <OMSTR >5</OMSTR >
    <OMSTR >20</OMSTR >
  </OMA>
</OMOBJ>
```

The *OMS* tag and its fixed valued parameters mark that the call is a remote function invocation so the CAS Server is able to treat the call accordingly. The first *OMSTR* tag contains the name of the function that must be called on the target CAS, the second *OMSTR* designates the package where the function is implemented, while the rest of the *OMSTR* tags represent the parameters that must be sent to the function. Based on the given description the function call string is constructed, the correct CAS is started in a new process and the call is sent to the process. For the above described call, the function call is:

`gcd(5,20)`

Once the result was computed by the CAS, it is read by the CAS Server and the result is sent back to the original client. Since the CAS Server is not able to semantically understand the result, it provides to the client the result “as is”, in whichever format it was returned by the executing CAS. It is the responsibility of the client to parse the result and extract any useful information it may contain. The result provided to the client has the structure:

```

<!ELEMENT OMOBJ (OMSTR|OME) >
<!ELEMENT OMSTR (#CDATA)>
<!ELEMENT OME (#CDATA)>

```

where the *OMSTR* tag contains the result, or in case of an error, the *OME* contains the error message.

Execution Management Interface

In scientific computational field, task execution management may be proven useful due to general characteristics of such tasks: long running time, amount of computational resources required by the tasks, real-time management of computational results. Execution management may be achieved simple through the following operations:

PAUSE – enables the client to pause an already submitted task. The client has submitted a task using an asynchronous call and therefore, it waits for a call-back message from the executing CAS Server with the response. If a pause request is issued, as a first consequence, the call-back containing the response is blocked, regardless of the real execution status of the task at CAS Server level. The CAS Server is not directly executing the task but it submits it further to a CAS. If the execution CAS implements pause/resume mechanisms the actual execution is paused. If not, the execution continues and, when it finishes, the result is stored so it may be provided to the client when a resume operation is called.

RESUME – enables the client to resume the execution of a task that was previously paused. If the underlying CAS was actually able to pause it, the execution is resumed. If the result was obtained meanwhile, the result is sent to the client.

CANCEL – enables the client to cancel a certain task. The CAS Server acknowledges the received cancel call and tries to stop the execution of the task. Since the task was cancelled at client's will, the CAS Server does not execute a call-back call as it would normally do when a task is finished. It is the client's responsibility to manage this situation at client side.

EXPRESS RESULT SETTING – a task execution may require a big amount of computational resources and a long completion time. Especially when the CAS Server is used as an execution unit integrated in a bigger architecture that is able to run computational workflows it may be useful to enable the client to assume a certain result for a task without really computing it. If the result of the task is set by the client, any result obtained by computing the task is discarded. The express result set operation triggers also the call-back through which the result is provided to the waiting client. This behavior is useful especially when the CAS Server's client is a workflow management engine.

The operations on the interface that allow the above mentioned functionality are:

- void pauseTask (String clientIdentifier, String invokeIdentifier) throws CASServerFaultType
- void cancelTask (String clientIdentifier, String invokeIdentifier) throws CASServerFaultType
- void resumeTask (String clientIdentifier, String invokeIdentifier) throws CASServerFaultType
- void setTaskResult (String clientIdentifier, String invokeIdentifier, String result) throws CASServerFaultType

OpenMath Resolver Interface

For simplicity, clarity and efficiency reasons, an OpenMath document may contain references to sections of the same document or even sections that are part of another document. In fact, the OpenMath object may be defined by reusing objects defined in the same or external document. In order to understand correctly an OpenMath object, a CAS using the described object must be able to parse any referenced object encountered. The resolve operation must implement mechanisms that collect all referenced objects used in the description of the OpenMath object being parsed even if the resolve process must use calls to remote third party storage entities that are able to provide the referenced objects.

One of the functionalities implemented by the CAS Server is ability to act as OpenMath storage repository. The CAS Server implements two main functionalities. Given a set of OpenMath references targeting OpenMath objects that are stored as XML documents in the CAS Servers file system, the CAS Server is able to extract:

- the list of OpenMath symbols used in the scope of the target OpenMath objects
- the targeted objects stored in a separate file for later retrieval by a client

Retrieving the list of OpenMath symbols is important when a client has to decide if a certain CAS Server implements all the functionality that a certain object requires. In order to execute a certain task, the CAS installed on the CAS Server that will actually execute the call must be able to understand all OpenMath symbols that describe the task. Therefore, the first criterion in electing a CAS Server is the ability of a installed CAS to understand the task.

General use-case scenario for resolving symbols. The client wants to submit a task to the most suitable CAS Server that is able to solve the call. The first requirement is to make sure that the CAS Server hosts a CAS that is able to execute the call. The client is able to interrogate known CAS Servers and is able to learn the list of symbols that every CAS installed on every CAS Server understands.

The call that the client wants to execute may contain OpenMath references to objects hosted on CAS Servers of the system. Thus, the client parses the initial call and retrieves the list of references that it uses. It groups references by the target host, and sends requests to CAS Servers requesting the list of symbols used to describe the targeted objects. A CAS Server receiving such a request and opens targeted objects.

If a targeted object contains in turn, references to other OpenMath objects, it must solve them. It contacts other CAS Servers and asks them to solve the references that are in the scope of the contacted CAS Server. Following this scenario, a resolve chain may be created. It must be noted that every CAS Server receiving such a request is aware of CAS Servers that precede it in the resolving chain and does not resolve references discovered during local resolving process that are in the scope of a precedent CAS Server. This way, the graph formed with active CAS Servers participating to a resolve process of a task, at a given moment, does not contain cycles.

General use-case scenario for obtaining targeted nodes. After the client elects the most suitable CAS Server to execute a certain task, it contacts the CAS Server and submits the task for execution. At this point, the CAS Server must make sure that the complete OpenMath objects used in the task request are available on the host where the execution is scheduled. The CAS Server parses the initial call and retrieves the list of references used by this call. All objects that can be found on the local CAS Server are extracted from their original files and stored in a common file. References that point to partner CAS Servers are grouped by their hosting CAS Server, and requests are sent to partner servers.

A server receiving a node resolve request will extract OpenMath objects that are available locally to a temporary file and a download URL that can be used to download the file is sent back tot the requesting CAS Server. Similarly with the symbol resolve process, node resolve process may result in a resolve chain.

Temporary files are created at every involved CAS Server that contains only the OpenMath objects that are local to that resolver. The download URL for the result files are passed from one resolving CAS Server to the server that requested the resolve, and additionally, the list of files that it received from descendant servers.

When the node resolve process ends, the CAS Server designated to execute the task has a list of URLs to files that contain all the objects required for the task to be executed. It downloads them and appends the content of those files to the temporary file where local objects were extracted. At the end of this process, all objects required for execution are therefore located in a single local file from which the executing CAS can retrieve them. The resulted file is OpenMath compliant, as it describes a valid OpenMath object acting as a container for other objects.

Note. While the node resolve process may lead to a resolve chain, in the final step, the one in which the files are downloaded, the execution CAS Server contacts directly the servers hosting the files.

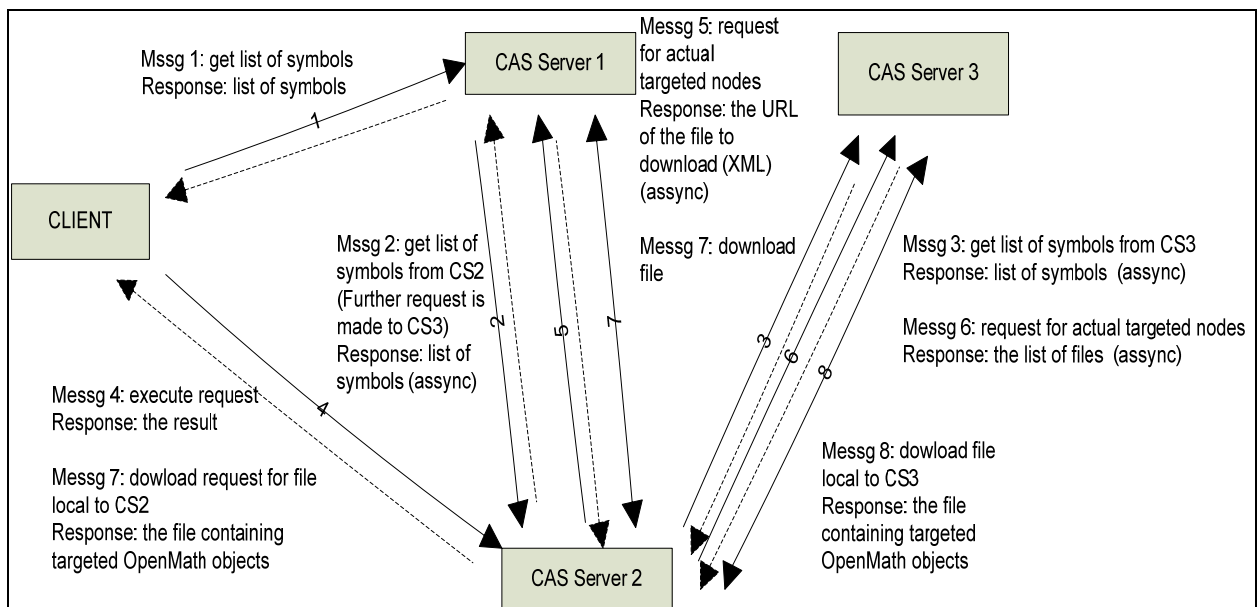


Fig. 1 Execution Scenario

Example. A simple scenario is depicted in Fig. 1. The client wants to solve a task containing references to OpenMath objects hosted by the CAS Server CS1. The object hosted on CS1 contains the references *CS1_OMR1* that targets an OpenMath object hosted in a local document and *CS1_OMR2* that targets a OpenMath object hosted by CAS Server CS2. We also assume that the object targeted by *CS1_OMR2* contains the reference *CS2_OMR1* that targets an object hosted by CAS Server CS3. The symbol resolving chain is thus:

$$CS1 \rightarrow CS2 \rightarrow CS3$$

CS1 will formulate the request to CS2 and will suspend the resolving process until CS2 contacts back the CS1 server with the response to the request. The response contains the list of symbols discovered by CS2 or any other resolver further contacted by CS2, in our case, CS3. When the symbol process is ended, CS1 responds to the resolve request of the client by sending the list of discovered symbols. Based on the list of symbols that the task contains and

the capabilities of the CASs installed on the CAS Servers, the task is assigned to the most suitable CAS Server. For our scenario we assume that the task is sent to CS2.

When the task is received by the CS2, it is parsed and CS2 discovers that the call contains a reference hosted by CS1. It sends a node resolve request to CS1 where the object targeted by *CS1_OMR1* is copied to a temporary file. The *CS1_OMR2* points to a CS2 object so at CS1 level there is nothing to do in this respect. Thus, CS1 responds to CS2 by sending the URL targeting the temporary file and a notice regarding the *CS1_OMR2*. At CS2 level, the object targeted by *CS1_OMR2* is parsed and the system discovers that another object, hosted by CS3 is required. Similar with the previous case, CS2 contacts CS3 and obtains a link to a temporary file.

Since there are no other references to be solved, CS2 contacts CS1 and CS3 and downloads from them the temporary files containing targeted objects. As a result of this resolve process, all objects required for execution are now stored locally to CS2.

Reference resolving internals

While the symbol resolving process only tries to discover the list of symbols that are used to describe a certain task, the role of node resolving process is to make sure that all objects required, that are part of the task, are downloaded and accessible to the CAS that will carry out the execution. A task submitted for execution at CAS Server may contain references to OpenMath objects located in the scope of the CAS Server or located on other CAS Servers. In OpenMath, the general format of a OMR is:

```
<OMR href="URI" />
```

For consistency reasons, we have restricted the format of the URI that can be used within OpenMath documents to the following:

- absolute URI: `http://host:port/path/to/file/filename#identifier`
 - o the protocol „http“ is used here only for consistency reasons. The http protocol will not be actually used to transfer the files. A specific protocol will be used instead.
 - o the host and port identify a valid CAS Server implementing the resolving server interface
 - o the path to the file section (“path/to/file/filename”) must identify a file stored by the specified CAS Server at the relative location specified in the path. By default, all the files that must be accessed locally by a CAS Server must be stored in a common root directory. The system is able to map this directory on the URL specified by a client and identify the file. The “path/to/file/filename” section of the URL is translated in terms of the default directory location used locally.
 - o the identifier represents the XML id of the node targeted by the URL
- relative URI: “path/to/document#identifier”. The actual XML node identified by this relative URI is determined relatively to the document’s location where it was found. If the URI has the format “#identifier”, the current file is assumed.
- local file URI: “file:///path/to/file#identifier”. This format must be used only for local documents that are going to be referenced locally by the executing CAS. As it can be observed, this format does not contain referencing information that can be interpreted in a global environment and therefore, if it is expected that the file must be available from remote locations, this URI must be changed accordingly to the absolute URI specification. On the other hand, this format is expected to be supported by CASs installed.

In order to illustrate in which way a real resolve process is carried out we shall use the setup depicted in Fig. 1.

Handling references in the main SCSCP call document.

The SCSCP call document describes the call and meta-information related to the call, as specified by the SCSCP specification. The document has several sections: the header section where detail regarding the call are specified; the main part where an OMA OpenMath objects attaches to a symbol identifying the remote operation to be executed, a list of parameters. Parameters must be valid OpenMath objects specified that the SCSCP call document is compliant with the OpenMath XML representation rules.

CAS Servers was built to accept OpenMath references that replace parameters of sub-objects of these parameters. OpenMath references are therefore not accepted in the header of the call or in other section of the document.

Since the execution CAS must be able to locate all objects referenced within the call, this document must not contain relative or file URIs. The only type of accepted references at this point are absolute references that comply with the format stated above so that objects referenced may be later retrieved from any executing CAS Server.

The original call submitted at CAS Server is parsed and the original document is modified such that all OpenMath references in the call are transformed to file URIs. When the resolve process ends, all objects referenced directly or indirectly by the call are stored in a temporary local file. The new URIs must identify correctly the objects stored in the temporary file. The system makes sure that the new XML IDs used in the result document have unique values.

Handling references local to the execution CAS Server.

It is possible that objects discovered through the resolving process contain references to documents hosted on the local host or even to objects described in the same document. As stated before, these relative URIs cannot appear directly in the main SCSCP call document, but may be discovered during the resolving process. Any such reference is modified to represent a relative URI to the same document (i.e. it only contains the anchor part of the URI – “#identifier”). The XML Id of the targeted object is also modified accordingly and the objects is copied to the local temporary file. A similar approach is used during the resolve process for references that are resolved on third party CAS Server resolvers.

Handling references pointing to remote locations.

Absolute references, discovered in the resolving process, that do not point to local documents must be resolved by the CAS Servers to which they point to. When the resolver on the execution CAS Server finds such a reference, it formulates to the appropriate CAS Server a request that contains: the list of references that the third party must resolve, new identifiers that will replace the identifier of the targeted objects and a list of absolute root references that the third party resolver must ignore. The identifiers are required because all discovered objects are copied to a single file at the execution CAS Server, and therefore, every node must have a unique Id. The list of root references are required because CAS Servers have to know their position in the resolving chain so it does not try to solve references that are in the scope of ancestor solvers.

The structure of the document containing all required objects

The temporary document containing all objects referenced directly or indirectly by the task call must be a well formed XML document that complies with all OpenMath rules. Because

this file contains objects appended during the resolve process, we use a standard OpenMath list specification structure. The basic structure of the file is:

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath"
      version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="list1" name="list"/>
    <OMOBJ > ... </OMOBJ >
    <OMOBJ > ... </OMOBJ >
  </OMA>
</OMOBJ>
```

where the inner `<OMOBJ> ... </OMOBJ>` must represent valid OpenMath objects from which the starting and trailing `OMOBJ` tags are omitted. The resulted document will thus contain a multi-dimensional list for which the depth and dimension depend on the structure of the objects/references that must be resolved.

An example

We assume the following call:

```
<OMOBJ xmlns = \"http://www.openmath.org/OpenMath\" >
  <OMATTR>
    <OMATP>
      <OMS cd=\"scscp1\" name=\"call_id\" />
      <OMSTR>194.102.63.120:26133:6766:dgsyte</OMSTR>
      <OMS cd=\"scscp1\" name=\"option_return_object\" />
      <OMSTR></OMSTR>
    </OMATP>
    <OMA>
      <OMS cd=\"scscp1\" name=\"procedure_call\" />
      <OMA>
        <OMS cd=\"scscp_transient_1\" name=\"WS_factorial\" />
        <OMI>..

```

We assume that this call (partially incomplete) is sent to the CAS Server CS2. At the CS2 level the call is parsed and the `"http://cas1.ieat.ro/file1.txt#id1"` is discovered. This absolute URI must be changed in order to be correctly handled by the executing CAS. Because it is part of the main SCSCP call, the reference is changed from an absolute URI to a file URI pointing to a new local temporary file that will contain all the objects required to execute the task. The reference is changed from the main call from

“`http://cas1.ieat.ro/file1.txt#id1`“ to
“`file:///local_repository/result_file#newid1`”.

Since the original URI is an absolute one and it identifies an object from CS1 server, the following call is formulated to CS2:

unsolved reference: `http://cas1.ieat.ro/file1.txt#id1`
new identifier: `newid1`
skipped reference: `http://cas2.ieat.ro/`

The above call instructs the CS1 server to extract the object targeted in the reference and to store it in a temporary file with the new XML Id “newid1“. It also notifies that all references that are discovered at the CS1 level and starts with “http://cas2.ieat.ro/“ must be skipped from resolving. Now we assume that, in the resolving process at CS1, two references “http://cas2.ieat.ro/file2.txt#id2“ and “/file11.txt#id11” are discovered. The targeted objects are copied to the a local temporary file and the two references are changed: “http://cas2.ieat.ro/file2.txt#id1“ is changed to “#generatedId1CS1 “ and “/file2.txt#id2” is changed to “#generatedId2CS1“. The ID of the XML targeted nodes are changed, in the temporary file, in our case from “id11” to “generatedId2CS1“. The response of the resolve process at CS1 level contains the following information:

unsolved reference: http://cas2.ieat.ro/file2.txt#id1

file to download: http://cas1.ieat.ro/tempfile1.txt

At CS1 level, the message is received and the unsolved reference is resolved by identifying the targeted object and by copying the object, with updated identifier, to the local result files. In a similar way, the rest of references are resolved. When this phase of resolving is completed, as the final step, files are downloaded and their content is copied to the result file. Thus, the file from CS1 “http://cas1.ieat.ro/tempfile1.txt“ and the file from CS3 “http://cas3.ieat.ro/tempfile1.txt“. All their content is copied to the file designated by the reference “file:///local_repository/result_file”

Service Interfaces Participating in OpenMath Symbol List Retrieval

Retrieving the list of OpenMath symbols that are used to describe a certain OpenMath object is important if the client needs to determine if a certain CAS is able to understand a call to solve. An OpenMath call may be solved by a CAS Server only if it manages a CAS instance that is able to understand every symbol describing the call.

Obtaining the list of OpenMath symbols describing a call should be achieved by trying to obtain the list of symbols used in the call and every targeted object used within the call. A correct approach is to group by target host the references encountered in the call and send a symbol resolve request for every target host. At CAS Server level, if parsing the referenced objects results in additional discovered references, it is the responsibility of the CAS Server to contact other CAS Servers using the same interface described below.

The interface is compound of two operations:

- *getSymbolList* – used to submit to a CAS Server a request to resolve a list of references that are in the scope of the contacted CAS Server. The references are in the scope of the CAS Server if the host used in the reference URLs matches the host of the CAS Server.

Parameters:

- *requestID* – of type string, represents an identifier of the request, unique both in the scope of the client and the CAS Server.
- *request* – of type string, represents a well formed XML describing the request. The structure of the parameter is detailed in the following sub-section.
- *callbackURL* – of type string, represents the URL of the service where the resolve response shall be sent.

Returns: nothing

Fault: OMRResolveFaultType(requested, errorMessage)

- *putSymbolList* – used to accept resolve responses from third party CAS Servers to which previously submitted resolve requests to obtain the list of symbols have been sent. This operation must also be implemented by any client regardless it is a CAS Server or a client that needs to resolve a list of references for the contained list of symbols.

Parameters:

- *requestID* – of type string, represents an identifier of the request that was originally sent by the client, unique both in the scope of the client and the CAS Server. It may be used to pair the original request.
- *result* – of type string, represents a well formed XML describing the response. The structure of the parameter is detailed in the following sub-section.

Returns: nothing

Fault: OMRResolveFaultType(requested, errorMessage)

Messages for Retrieving the List of OpenMath Symbols

In order to obtain the list of symbols, the client must call the correct CAS Server and supply the list of references that must be solved. The contacted CAS Server will contact the client when the submitted references and subsequent references are solved or it may respond with an error message.

The OM Symbol Resolve Request

The request message must be a plain string representing a well formed XML document having the structure described by the following DTD:

```
<!ELEMENT symbolrequest (reference+, skipreference*)>
<!ELEMENT reference (#CDATA)>
<!ELEMENT skipreference (#CDATA)>
```

An example of a valid message is:

```
<symbolrequest>
  <reference>http://host1.com/path/file#idl</reference>
  <skipreference>http://host.com/</skipreference>
</symbolrequest>
```

The above request message specifies that the server should resolve the reference specified by the “<reference>” element. All subsequent references that are in the scope of the host specified through the “<skipreference>” (i.e. the references that begin with the string `http://host.com`) should not be solved by the resolver.

The OM Symbol Resolve Response

The response message must be a plain string representing a well formed XML document having the structure described by the following DTD:

```
<!ELEMENT symbolmessage ((symbol+, reference*)|error) >
<!ELEMENT symbol (name, cd)>
<!ELEMENT name (#CDATA)>
<!ELEMENT cd (#CDATA)>
<!ELEMENT skipreference (#CDATA)>
<!ELEMENT error (#CDATA)>
```

An example of a valid response message that is sent when the resolve process ended successfully is:

```
<symbolmessage>
  <symbol>
    <name>symbol1</name>
    <cd>cd_of_symbol1</cd>
  </symbol>
  <symbol>
    <name>symbol2</name>
    <cd>cd_of_symbol2</cd>
```

```

    </symbol>
    <reference>http://host.com/skipped_ref/document#id</reference>
</symbolmessage>

```

The above response message specifies that the server found the OpenMath symbols (symbol1, cd_of_symbol1) and (symbol2, cd_of_symbol2) and encountered a reference that was in the scope of a parent resolver “http://host.com/skipped_ref/document#id” which the resolver did not attempt to solve.

An example of a valid response message that is sent when the resolve process ended with an error is:

```

<symbolmessage>
    <error>Error message</error>
</symbolmessage>

```

The above response message specifies that the resolve process ended with the error “Error message”.

Service Interfaces Participating in Target Node Retrieval

Before starting to execute a call described based on OpenMath objects, all the required objects must be available for the executing CAS. Since CASs are not in general able to contact remote hosts to obtain objects that are not hosted on the execution machine, the CAS Server implements such mechanisms.

The interface consists of two operations. One operation allows third party clients to submit object requests while the other one is able to store responses for requests that the current CAS Server previously submitted to other CAS Servers. The signature of the operations and their usage scenario is presented below:

- *fullSolveGetList* – used to submit to a CAS Server a request to resolve a list of references that are in the scope of the contacted CAS Server. The references are in the scope of the CAS Server if the host used in the reference’s URLs matches the host of the CAS Server. This operation examines the list of references, finds the targeted objects and stores them in a temporary file that is available for download.

Parameters:

- *requestID* – of type string, represents an identifier of the request, unique both in the scope of the client and the CAS Server.
- *request* – of type string, represents a well formed XML describing the request. The structure of the parameter is detailed in the following sub-section.
- *callbackURL* – of type String, represents the URL of the service implemented at client level waiting for the resolve response.

Returns: nothing

Fault: OMRResolveFaultType(requested, errorMessage)

- *fullSolvePutList* – used to accept resolve responses from third party CAS Servers to which it has previously submitted resolve requests. The request message contains a list of file URLs that may be used to retrieve the actual targeted OpenMath objects.

Parameters:

- *requestID* – of type string, represents an identifier of the request that was originally sent by the client, unique both in the scope of the client and the CAS Server. It may be used to pair the original request.
- *result* – of type string, represents a well formed XML describing the response. The structure of the parameter is detailed in the following sub-section.

Returns: nothing

Fault: OMRResolveFaultType(requested, errorMessage)

Messages to Retrieve the Targeted Nodes

At the moment of the execution, all the objects referenced by an OpenMath reference and used in the initial call document or in a subsequent referenced object must be available locally. The CAS executing the call should not and is not expected to contact remote machines to transfer the requested objects.

The OM Object Resolve Request

The request message must be a plain string representing a well formed XML document having the structure described by the following DTD:

```
<!ELEMENT noderequest (reference +, skipreference*) >
<!ELEMENT reference (value, newid)>
<!ELEMENT value (#CDATA)>
<!ELEMENT newid (#CDATA)>
<!ELEMENT skipreference (#CDATA)>
```

The string value supplied as a text child for the “<value>” element should contain the reference URL that must be solved. The *newid* value supplied as a text child of the “<newid>” element should be the identifier value that replaces the identifier part of initial reference. In the reference “http://host.com/skipped_ref/document#id” the target identifier “id” is replaced with the new identifier supplied in the request message. This replacement is required in order to avoid identifier collision of identifiers since all objects retrieved by executing the resolver process will be at the end stored in the same OpenMath document. As a consequence, the client request should be formulated to make sure that identifier name collision is highly improbable.

The “<skipreference>” elements should be used to designate the root URLs that identify references that should be skipped by the server. An example of a valid request message is:

```
<noderequest>
<reference>
  <value>http://host1.com/path/file#idl</value>
  <newid>newidl</newid>
</reference>
<skipreference>http://host.com/</skipreference</skipreference >
</noderequest>
```

The above request message requests that “http://host1.com/path/file#idl” should be resolved and the identifier of the target object should be replaced with the “newidl” value. It also instructs the resolver to skip all references that begin with “http://host.com/”.

The OM Object Resolve Response

The request message must be a plain string representing a well formed XML document having the structure described by the following DTD:

```
<!ELEMENT noderresponse ((reference*, filename+)|error) >
<!ELEMENT reference (#CDATA)>
<!ELEMENT error (#CDATA)>
```

An example of a success response for a object solver request is presented below:

```
<noderresponse>
  <reference>http://host.com/skipped_ref/document#id</reference>
```

```
<filename>gsiftp://temp_dir/file1</filename>  
</noderesponse>
```

The message informs the client that during the resolve process a reference that was in the scope of a higher rank resolver was discovered and was not solved. It also informs the client that targeted objects were saved and may be retrieved by transferring the file at the designated URL. As the DTD shows, the list of skipped references may contain more than one element. Similarly, the list of files containing the target nodes may contain more than one file but only one file is hosted on the current CAS Server. The rest of files were created and are stored on partner CAS Servers that the current CAS Server had to contact in order to fully resolve the references encountered that were in its resolving scope.

If an error occurs during the resolve process, an error message similar with the following one may be sent to the client:

```
<noderesponse>  
  <error>An error message.</error>  
</noderesponse>
```

Download of Result Files

There are often situations when the input data for a task and output data generated by the processing of the task are large. In this situation combining multiple CASs in an execution workflow requires moving this data from/to processing servers that are involved in the computation. A rule of thumb in distributed computing is to minimize the load on the network as much as possible.

Describing tasks using OpenMath offers the advantage to use references to external documents. Therefore, an execution call or the response to such call may be fragmented in one or multiple OpenMath objects that can be stored, for convenience, in separate files. A simple scenario that emphasizes the advantages of this approach may be imagined based on the architecture depicted in Fig.2 :

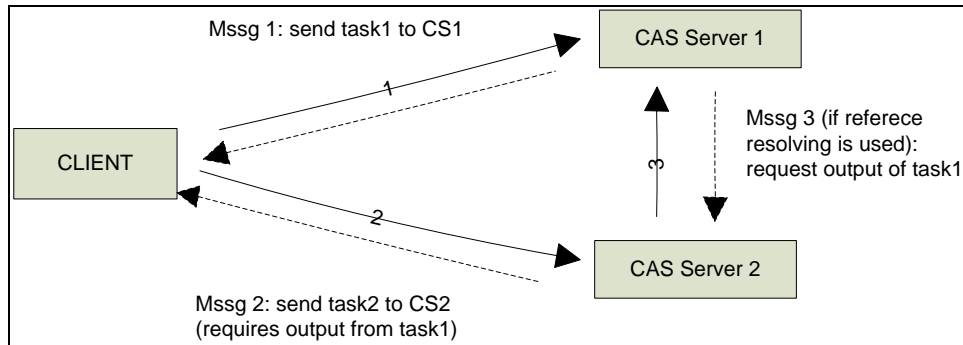


Fig. 2 Advantages of using OMR references

Scenario 1. In this scenario we assume that OpenMath references are not used. The client wants to execute the task *task1* on CAS Server CS1. Further, the result of the computation of *task1* is required as input data for another task *task2* that the client submits to the CAS Server CS2. Since no references are used in this scenario, the result from *task1*, which may be just an intermediary result must be transferred from CS1 to the client. When formulating the *task2*, the client must submit to CS2, the complete task, including the complete output of *task1*.

Scenario 2. Similarly with Scenario 1, the client wants to execute *task1* and *task2* on CS1 and CS2 respectively. Unlike the first scenario, CS1 computes *task1* and due to the large size of the output it decides that the result should be broken in two parts. The first part is a simple OpenMath objects that contains a OMR reference to a temporary file where the rest of the result is stored. The client receives the response from CS1 and forwards it together with other details describing *task2* to CS2. Using the reference resolving mechanism implemented by CAS Servers, CS2 is able to resolve references and is able to obtain the complete result of *task1* by downloading the referenced object.

Advantages of using OpenMath reference are easily identified. The result of *task1* which may not be even needed directly at client side does not have to travel back and forth on the network. Since CS1 and CS2 are server nodes it is highly probable that the bandwidth capacity of the link between CS1 and CS2 exceeds by far the capacity of the links between the client and CS1 and client and CS2. A collateral advantage of this setup is that the client requires little computation resources and can be even implemented on hand held devices.

As said above, in general CAS lack network communication capabilities and therefore they should not be expected to be able to coordinate transfer of files. CAS Server implements the reference resolver as a separate sub-component of the system and the file transfer is implemented over the functionality provided by Globus RFT (Reliable File Transfer Protocol). Security is also ensured by implemented mechanisms offered by the Globus GSI.

References

- [1] S.Freundt, P.Horn, A.Konovalov, S.Linton, D.Roozmond, Symbolic Computation Software Composability Protocol (SCSCP) specification, Version 1.3, 2009 (<http://www.symbolic-computation.org/scscp>)
- [2] The OpenMath standard, <http://www.openmath.org>
- [3] GAP Computer Algebra System, <http://www.gap-system.org/>
- [4] Oasis Web Services Resource Specification, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf