CREAM-OWS

INTRODUCTION

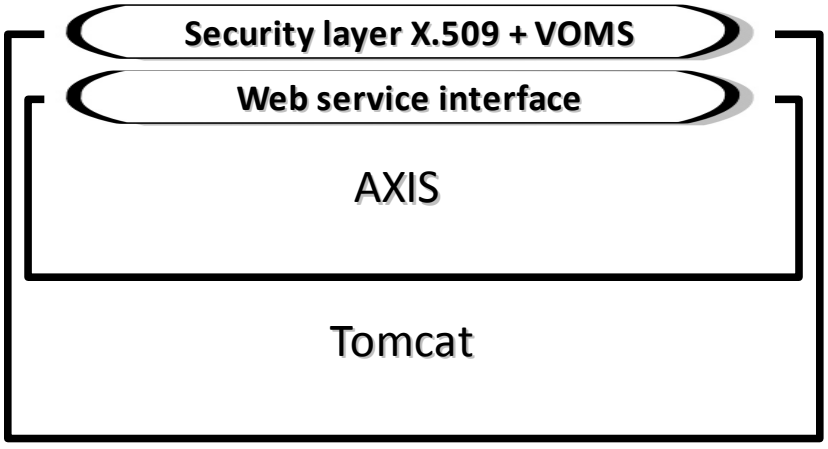
CREAM (Computing Resource Execution and Management) is a system designed to efficiently manage a CE (Computing Element) in a Grid environment and it is part of the gLite middleware distribution, currently in production used within the EGEE Grid infrastructure.
CREAM is a simple, robust and lightweight service for job operations designed to provide efficient processing of a large number of requests for computation on managed resources.
Requests are accepted from distributed clients via a Web Service based interface which enables an high degree of interoperability with clients written in different programming languages: currently Java and C++ clients are provided, but users are free to use their own tools and languages to interface to CREAM.

CREAM FUNCTIONALITY

Although the main functionality of CREAM concerns the job management operations (e.g. job submission, cancellation, monitor) at the Computing Element (CE), CREAM has been designed for providing a functional core and allowing to plug any type of new WS in a simple way. In fact, in order to increase the functionality exposed by CREAM, it is possible to plug different Web Service Interfaces interacting with the internal core of CREAM.

CREAM ARCHITECTURE

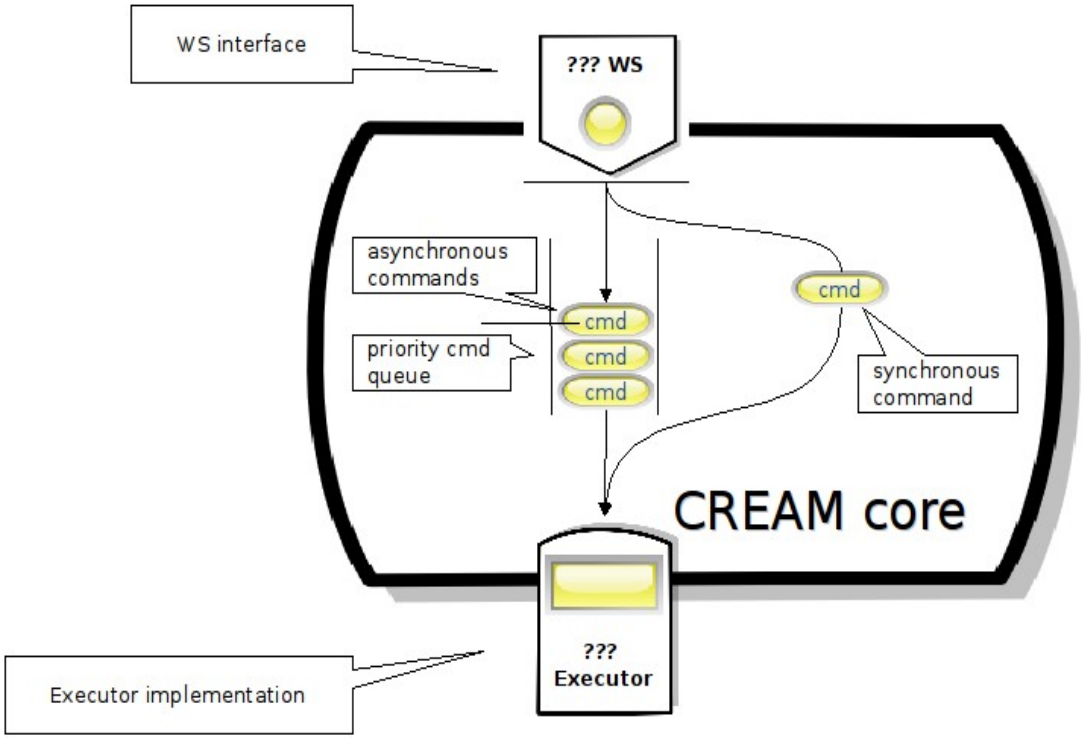
The CREAM has been designed for general purpose framework with pluggable functionalities/operations. So, for example, the functionality for accessing a database can be easily added and plugged to CREAM. CREAM is based on Open Standards especially related to the Web Services and Grid technologies and in particular it is compliant with the SOA architectural model.
From the implementation point of view, CREAM is a Java application running as an extension of a Java-Axis servlet inside the Tomcat application server, as shown in figure below. For what concerns the security, authentication and authorization mechanism are implemented with X509 and VOMS support.



CREAM has been developed for providing a functional core, that is a generic command executor. It is possible to customize the core by defining concrete implementations of the generic command executor.

These implementations are new command executors pluggable on CREAM.

Commands are the operations defined on the web service interface exposed by CREAM and they can be managed by different command executors. Two kinds of commands can be defined: synchronous and asynchronous. Synchronous commands must be executed immediately upon receipt, while asynchronous command execution can be deferred at a later time. Moreover, it is possible to define an execution mode for each command: sequential or parallel.



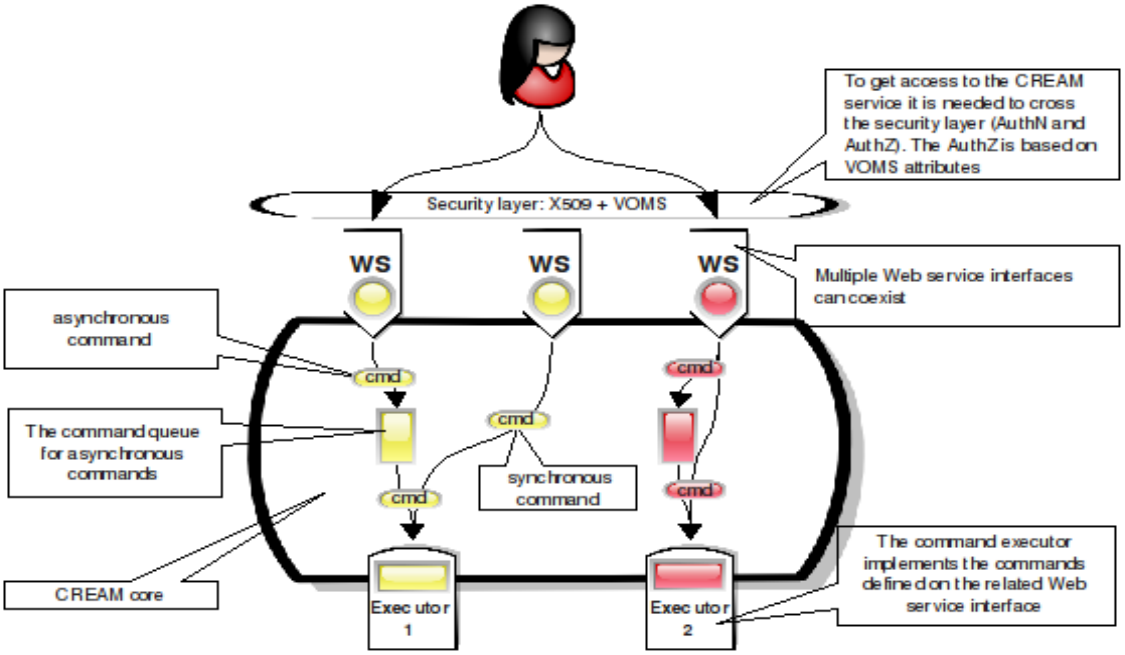
CREAM provides also a “generic” WS interface for executing commands. The main operation is the execute() one which allows a generic client to execute synchronously/asynchronously the specified command implemented by CREAM through the associated command executor.

Execute() is very generic operation:

CommandResult execute(Command)

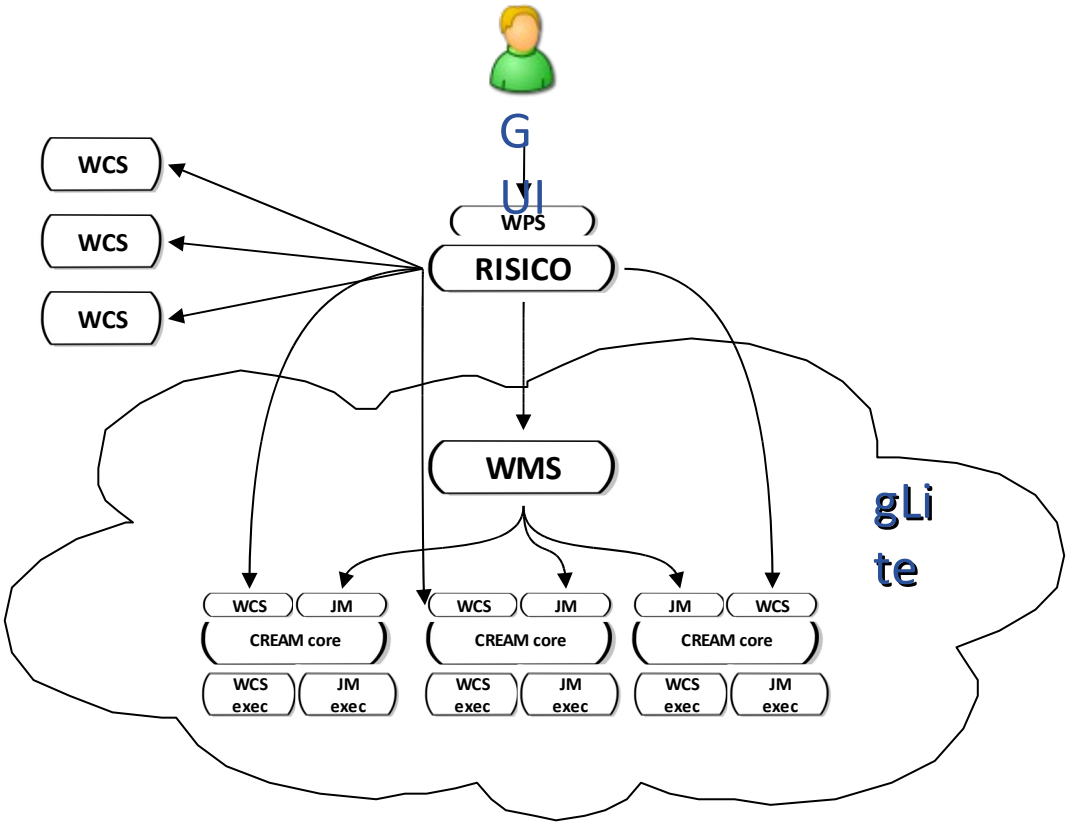
Command: is the argument used by the client to provide input parameters, execution category, specific executor name, etc

CommandResult: is the outputs produced by the execution of the command (if executed synchronously) or a command identifier (if executed asynchronously) that it'll be used to retrieve the outputs.


To get access to the CREAM service it is needed to cross the security layer (AuthN and AuthZ). The AuthZ is based on VOMS attributes
Security layer: X509 + VOMS
WS
WS
WS
Multiple Web service interfaces can coexist
asynchronous command
cmd
cmd
cmd
cmd
cmd
The command queue for asynchronous commands
synchronous command
CREAM core
Executor 1
Executor 2
The command executor implements the commands defined on the related Web service interface

OWS implementation on CREAM

The integration of Grid computing and Geospatial infrastructures is a clear need expressed by several international and European projects/initiatives. The better approach to achieve this purpose encourages the integration of the basis technologies from the Grid and Geospatial fields, the hiding of complexity from end-users and applications developers as well as relying on existing standards.
The main goal is to make Geospatial services grid-enabled by implementing just their business logic without worrying about grid things like security, access to computational and storage resources, resource discovery, monitoring and accounting and proxy delegation.
As described above, CREAM is very flexible and its functionalities can be extended easily. The use of CREAM as building block for new grid-enabled services makes the developer free to concentrate just on the implementation of the business logic of its service. In fact grid things like security, support for data persistence, fault tolerance, high performance are all features inherited by the CREAM core.
So CREAM could be a good candidate for implementing OWS services.
For example, consider the OGC-WCS grid-enabled service which allows clients to access part of a grid coverage offered by a server. The data served by a WCS is grid data usually encoded in a binary image format and the output includes coverage metadata.
A possible implementation of such OWS service could be based on CREAM technology. The figure below describes a lightweight architecture. In particular it is shown how CREAM could become a coverage provider by adding the WCS interface and the related executor to its core. Notice that, in the example inspired by the G-RISICO prototype developed by CYCLOPS project, both WCS and Job Management (JM) interfaces coexist in CREAM and will be accessed respectively by RISICO and WMS services.



Analog considerations can be done with respect to the OGC-WPS service. In this case a possible integration of WPS in CREAM is encouraged because the WPS interface presents strong analogies with the CREAM “generic” one (e.g. execute() operation). Moreover there are pragmatical advantages if (almost) all resources (e.g. geospatial data) are locally distributed and provided by a single site which must provide an adequate local cluster of PCs handled by a LRMS. This aspect avoid the intrinsic grid overhead. In this scenario CREAM will provide three different services accessible directly by the user or through other services.