Portlet

Contributors

Martin Skou Andersen		skou@nbi.dk
Giuseppe Fiameni		g.fiameni@cineca.it
Luigi Zangrando		luigi.zangrando@pd.infn.it

Document Status Sheet

Version	Date	Status	Comment
0.1	10/21/09	Draft	Fiameni
0.2	10/23/09		Contributions Fiameni, Andersen
0.3	10/25/09		INFN staff (Cresti, Candiello, Caltroni, Zangrando)
0.4	10/26/90		High-level plan Fiameni

Summary

1. List of acronyms

DC	*Distributed Computing*
HPC	*High Performance Computing*
HTC	*High Throughput Computing*
JSR	*Java Specification Requests*
OASIS	*Organization for the Advancement of Structured Information Standards*
SAGA	*Simple API for Grid Applications*
WSRP	*Web Services for Remote Portlets*

2. Introduction - portlet specifications

Users and visitors of grid web resources are currently faced with a host of different interfaces, each often incompatible with the others and each requiring different customizations even when using the same set of basic building blocks to offer a particular service.

There is a need to make the interfaces more uniform and usable and to enable the re-use of basic web components, in particular within the context of current and planned science gateways and grid user portals. Thus the development of middleware must include solutions compatible with these usability needs and with the development of these portals.

One solution lies in the use of tools implementing the new standards for web portals like the Java Portlet Specifications (v1.0 JSR 168, v2.0 JSR 286) and WSRP (WS for Remote Portlets).
Portlets are pluggable components that generate dynamic contents in response to web

requests. The content that a portlet produces is a fragment of mark-up code that, in aggregation with other non-overlapping portlet results, contribute to the creation of the portal page.
JSR 286 is a Java SUN specification that defines the concept of a Portlet API as a mean on aggregating several content sources and application sources and web application front ends within a portal frame. A portal framework that complies with the JSR 286 specifications is called portlet container and it represents the environment where portlets are instantiated, operated and finally destroyed. The intent of the portlet specifications is to solve the interoperability issues enabling software developers to develop portlets that can be plugged into any portal which complies with the specifications. The JSR 286 is a specification for Java developers and therefore neglects the use of different languages for the development of portlets. To overcome this limit, OASIS[1] has elaborated a new specification called WSRP. The main aim of WSRP is to keep separate portlets from portals introducing the concepts of producer and consumer. A producer is a service provider, and a consumer is a service client. A consumer is responsible for redirecting requests from portal users to the corresponding producer service. The producer handles the request and sends a response back to the consumer providing well formatted mark-up fragments.
The client-side portal can then retrieve these mark-up fragments from its consumer and render the Web page. Since WSRP is based on Web services technology, it is automatically language and platform independent. Thus the producer can be developed using any programming language that supports web service technology.

Compliant portlets can be deployed to any portal supporting the standard with minimal or no effort. This promotes code re-usability and the use of a component-based model on the developer's side. In perspective, the market value of standard-compliant portlets is also augmented since they can be marketed to the wider audience of everyone using a portal technology supporting the standards. WSRP deals with the aggregation of content produced by portlets running on remote machines that use different programming environments (J2EE, .NET).

1 *www.**oasis**-open.org*

3. State of the art and known issues/requirements

3.1. General consideration on portlet development

Portals have evolved from simple Web applications providing basic functionalities to enterprise-level application that deliveries platform for offering composite applications.
In a world where organizations are moving toward service-oriented architecture (SOA), portlets become strategic to provide the user interface for organization services.
Portlets are attractive because of the promise of flexibility, but they are well suited to aggregate pure content, functionally independent or simply related. Portlets can also enable reuse, because you can configure them into multiple pages/locations in a fairly simply way.
The problems can start when there is the need to decompose complex functions with multiple steps and interactions. In this scenario determining the granularity of the portlets is difficult, and careful consideration needs to be given in designing the interactions between the portlets.
Thus, the adoption of the portlet technology strongly depends on the complexity of the functions that are expected to be implemented and how much reuse of the components it has been anticipated. Deciding to exploit the portlet concept means to embrace the full lifecycle, and avoid any temptation to work around them, otherwise the risk to incur all the overheads and restrictions of portlets, without being able to realise their advantages, becomes high.

3.2. Unicore

3.2.1. eDEISA

As part of the eDEISA project work an alternative version of the existing Life Sciences Portal has been developed. This portal is based on the open source portal framework Liferay and exploits DEISA's operational UNICORE 6 infrastructure. The portal framework interacts with the UNICORE 6 infrastructure through an implementation of the Open Grid Forum's Simple API for Grid Applications, also known as SAGA, and that relies on HILA for the interaction with the machine resources. The portlets in the portal exploit the JSR 286 standard as well as other de-facto Java technology. The current implementation of the portal requires various improvements, at different levels, to be considered a production-ready solution. There are some aspects that need to be investigated prior to reconsider the possibility to take the current release of the portal and move it a step forward the production level. Critical issues are:

- the SAGA specification is not exhaustive and has never received much attention from the community
- the HILA library, for the access to UNICORE resource, is a rather framework than a library and its adoption in a pre-existing framework, as the portlet container is, may require much effort for the resolution of library dependency conflicts that might arise. Besides, the performance of the portal is negatively affected by the presence of several software layers.

3.2.2. VINE Toolkit (http://vinetoolkit.org)

Vine Toolkit is a portal framework for the development of grid applications and is the result of a collective effort lead by the Poznan Supercomputing and Networking Center and funded by several EU projects.
The toolkit consists of a modular, extensible Java library and offers to developers API for implementing Grid-enabling applications. Vine can be deployed for use in desktop, Java Web Start, Java Servlet 2.3 and Java Portlet 1.0 environments. The Java Portlet 1.0 is the previous portlet specification (JSR 168) still supported by several portlet containers. Besides Vine supports a wide array of middleware and third-party services. The middleware currently supported are gLite, UNICORE, Globus Toolkit, Gria.

3.3. ARC

ARC does not contain any portlet implementations. However the libarcclient offers a plug-in based API for grid applications, and it implements all functionality needed for different proxy generation (Pre-RFC, RFC, VOMS, SAML-assertions and MyProxy), information (GLUE2), submission and management of grid jobs and data handling (arc, file, gridftp, http, ldap, lfc, rls and srm). Through language bindings created with SWIG (part of ARC), this API have been used by the Lunarc Application Portal (LAP) to create a Python GUI and a web portal utilizing the features contained in the libarcclient. I do not know the exact details on LAP.

With regard to plug-ins in libarcclient, there currently exist plug-ins for the legacy gridftp based interface to the ARC grid-manager, the BES compliant job execution web service A-REX, the gLite CREAM interface and UNICORE. The gLite and UNICORE plugins do not support yet the full set of features available in ARC plugins

3.4. gLite

Although gLite is based on consolidate standards (e.g. WS), at the present time it does not to take advantage of portlet standards. However it provides a large set of libraries (for Java and C++) implementing APIs covering a wide area on grid environment. In particular gLite provides libraries to access middleware services (WMS, CREAM, CEMon, LB, StoRM etc), handling data, jobs, and certificates and security aspects. All of them can be considered as building block for the support of portlet-based user interfaces.

4. What EMI can do to address the issues/requirements (impact)

In the context of a comprehensive EMI programming interface consolidation, a set of activities is planned to complete the APIs and to develop also portlet-compliant programming interfaces.
The benefits that a portal interface can provide to grid users are many. Access to computational grid resources normally requires the utilization of client-side tools. The configuration and the installation of these tools usually imply a set of system administration tasks that, for the major part of the scientific users, can be of difficult resolution (users might have limited knowledge in the involved technology).
A set of portlet-based (EMI) services would ease the task of designing complex user applications and could have the final effect to widen the EGI use in the scientific communities. Being based on finalized standards, and thanks to the availability of high quality portlet containers (Apache Pluto, Liferay, Apache Jetspeed-2, the eXo platform, uPortal), these EMI Web User Interface (WUI) services could be re-used in several contexts and aggregated in specific community-service portals.
Consequently, the Web is an ideal platform for distributing services and guarantee that an increasing number of users can exploit them with ease. This makes a Web-based portal an

adequate choice for providing the same services as a desktop client.

4.1. Possible solutions, implementation details, relationships with other services

A portlet-based web user interface simplifies the development of applications through standards and the aggregation of re-usable components into a single point of access.
The typical use case would see scientists accessing the computational resources through an application of their specific scientific field and setting up a simulation and/or computation context to be transferred to the target infrastructure for massively parallel or distributed computations.
They would be able to use the EMI portlet-based services to execute the entire life-cycle of data production from a single point (the portal) using uniform and standard applications (the portlets):

- user authentication, where the pertaining virtual organization is matched, the certificates are exposed and the relevant proxies are set up based on the credentials of the scientist;
- data environment setup, allowing the application to transfer the simulation/computation data to dedicated servers in the infrastructure,
- job submission, where a single job or a collection of jobs is submitted to the EGI infrastructure in the specific VO via the specific HPC/HTC/DC services (via workflow management systems or via lower level interfaces) defined by the application; the application could also submit jobs reacting to specific results occurring in the elaboration of data;
- continuous job monitoring, by giving to the user a set of views around his jobs, their status, timings, queues, etc;
- job management, allowing the user to interrupt, cancel and resume the job execution;
- data result transfer/display, returning the computed data to the user through the transmission of relevant information for the retrieval (addresses, interfaces, etc).

Pre-packaged portlet-compliant interfaces will be provided for each relevant component in EMI.

The goal is to:

- provide out-of-the-box WUI interfaces for EMI services ready to be accessed by end users in grid portals,
- pre-package a set of unit interface modules to help application developers to assemble complex interfaces accessing EMI services in an integrated logic. It is supposed that such applications will be useful for the development of thematic “science gateways” (for Life Sciences, Earth Sciences, HEP, etc).

5. High-level plan (duration, milestones, success criteria) (assume three years)

The aim of the EMI project to provide a portal based interface bases on the following working plan. A list of high-level tasks, deliverable and principal milestones is presented below.

Tasks:

1) **Gather, analysis, consolidate requirements from user communities (2/3 Person Months);**
    a) determine which functionalities it makes sense to offer through a portal interface considering the limitations of the web technology in performing specific activities (e.g. some limitations exist in offering data transfer functionality);
    b) deliver a list of case stories as a collection of tests to prove that any developed component provides the expected functionalities inline with the requirements stack;
2) **Portal architecture design (4/6 PM):**
    a) produce a survey on portal technology and architecture based on portlet specifications;
    b) investigate the already available portal implementations and determine if they can be either totally or partially exploited in EMI (e.g. a re-factoring of the HILA library may facilitate the completion of the eDEISA portal);
    c) identify a preliminary architecture vision of the EMI portal system and determine its main components;
    d) consolidate the portal architecture design in accordance to user’s feedback, implementation activities and test results;
3) **Implementation of the EMI portal (24/36 PM):**

   a) implementation of the basic functionalities;
   b) implementation of the advanced functionalities.
4) **Test and validation (6/8 PM):**
   a) evaluate the work results:
   b) validate the user requirements on the base of the user's requirements collected on Task 1;
   c) measure the quality of the work achievements;

An iterative approach that involves short burst to deliver nearly usable components could reduce the risk of being way off in terms of requirements fulfilment. The requirements, especially because they will come from different communities, might evolve over the course of the project with more detail and more unexpected surprises extracted along the way.

If for some reason it is not possible to merge the work on portlet conformance into the work on clients/common API then wrappers for the part of the API which does not conform to portlets standards must be created in order insure conformance.

Deliverables:

- D1: user requirements analysis and evaluation criteria specification (Task 1)
- D2: preliminary portal architecture design (Task 2)
- D3: first release of the portal offering basic functionalities (Task 3)
- D4: user requirements consolidation and new evaluation criteria specification (Task 1)
- D5: final portal architecture design (Task 2)
- D6: second release of the portal offering advanced functionalities (Task 3)
- D7: test results and quality measurements (Task 4)

Milestones:

- M1: deliver of the first portal release (PM 18)
- M2: deliver of the second portal release (PM 36)