Java Portals and Java Portlet Specification and API

Gennaro Costagliola, Filomena Ferrucci, Vittorio Fuccella
Dipartimento di Matematica e Informatica, Università di Salerno
Via Ponte Don Melillo, I-84084 Fisciano (SA)

{gcostagliola, fferrucci, vfuccella}@unisa.it

Introduction

Second generation Web portals distinguish themselves from first generation ones for their architecture, which is component-oriented. In particular, the basic component constituting them, is often referred to as *portlet*. The portal is responsible for aggregating information coming from different sources, local or remote, available in the form of mark-up fragments. Each of those fragments is produced by a portlet. In the context of web portals, the possibility to deploy a portlet in any portal is particularly significant. To this extent, that is, to achieve interoperability among portals, it has been necessary to define a standard way to develop and deploy portlets. Two main standards have been defined and widely adopted by producers: the *Web Services for Remote Portlets (WSRP)* and the *Java Portlet Specification and API (JSR 168)*. The former is more oriented to the definition of rules about the use of remote portlets, the latter is focused on the definition of interfaces for the development of portlets which can run in Java-based portals. The definition of a standard specification for Java technology follows a specific process, known as the *Java Community Process*, where several contributors, under the supervision of *Sun Microsystems*, write and revise the draft of the specification several times until its final publication as an approved standard. Most of the Java technologies, part of the *Java 2 Enterprise Edition*, the platform for the development and deployment of distributed enterprise applications, follow a consolidated architectural model, called container/component architecture. This model offers the chance to develop components and deploy them on different containers. Both component and containers compliant to specifications can be developed independently and commercialized by different software vendors, thus creating a market economy on Java software. Furthermore, several good-quality Open Source products compete with them. The *JSR 168* follows the container/component model and, as shown by a survey presented in the sequel, its adoption has grown until it has become an important reference-point which cannot be excluded from the projects aimed at the development of Web portals. An overview of *JSR 168* follows: its content is summarized, starting from the definitions of portal, portlet and portal container, and continuing with other important matters, such as how portal technology relates to other *Java* technologies. Furthermore, a parade of the most important existing implementations of the specification is presented.

Background

According to Bellas (2003), we are at the second generation of Web portals. The main characteristic which distinguishes it from the previous, regards the architecture of such Web based applications. A second generation portal has a component-oriented architecture. Its adoption, compared to that of a monolithic architecture, typical (with several exceptions) of the first generation portals, improves development, maintenance and reusability.
One of the basic components of a web portal is the portlet. Such a software entity, is responsible for rendering the mark-up fragment necessary for showing information or

providing a service coming from a source of the World Wide Web. The portal is responsible for aggregating several portlets in the pages of a unique system, homogeneous in its appearance, and tailored to the user preferences. The mark-up fragment is directly generated by a service located on a remote host.
Some years ago, several vendors were already producing portals based on the mechanism described above. A noteworthy example was *Jetspeed*, the portal of the *Apache Group*, developed with *J2EE* technology. Almost each portal producer defined a proprietary *APIs* for building portlets, resulting in a lack of interoperability.
Having to aggregate content from different sources, a fundamental step was to reach an agreement between portals and portlet producers on the way in which portal could obtain the *HTML* fragment for the portlets. The need for interoperability has often been the most important reason to establish a standard. An early solution has been found defining the *Web Services for Remote Portlets (WSRP)* standard: a Web service interface (defined in *Web Service Definition Language*) through which portals can interact with the remote producer's portlets. The *WSRP 1.0* specification (OASIS, 2003) was approved as an *OASIS* standard in August, 2003. Being based on Web services, several interfaces to adopt the standard have been developed for the most used technologies (e.g. *J2EE*, *.NET*, and so on).
Some months later, a new important specification reached its final release: the *Java Specification Request 168* (Abdelnur & Hepper, 2003), also known as *The Java Portlet Specification*. The need for *JSR 168* was motivated by the inadequacy of the *Servlet/JSP* specification to represent the high level concepts of a web portal application: even though it is possible to build any Web-based application using *Servlet/JSP* specification, the development of a portal needs deals with new concepts, such as portal, portlet and portlet container. The scope of the specification was to develop an *API* set layer on the underlying one of servlets. Contents treated in *WSRP* specification often overlap with those treated in *JSR 168*. E.g., both define portlet view modes and window states. The main differences reside in the location of the portlets and in the technologies: *WSRP* is more oriented to the definition of mechanisms for the use of remote portlets, which can be developed using different technologies. To this extent, it defines two standard Web service-based interfaces: one for the description of the services provided by a portlet and another one for the mark-up generation. The main benefit which can be gained by supporting the standard is that a portlet, developed with whatever technology, can be deployed on a location and displayed in several remote Web portals. *JSR 168*, instead, defines an interface suitable for local portlets, developed with *J2EE* technology.
Several *APIs* and Java related technologies have been aligned together in a cohesive development and deployment platform, called *Java 2 Enterprise Edition (J2EE)*. A strong point of *J2EE* is the support for component-oriented development, which simplifies the development and maintenance of software and contributes to improving its quality. The *J2EE* component-oriented development is based on the so called container/component architecture. A container is a software entity that runs within the server and is responsible for managing specific types of components (Ahmed & Umrysh, 2002). It provides several services to the *J2EE* components deployed within it, such as: managing its lifecycle, resource pooling, enforcing security, providing more information and services through Service *APIs*. Examples of containers and components fit to them are A*pplet Container* and *applets*, *Web Container* and *servlets*, *Enterprise Container* and *Enterprise JavaBeans*. The *container-component* architecture is shown in figure 1.

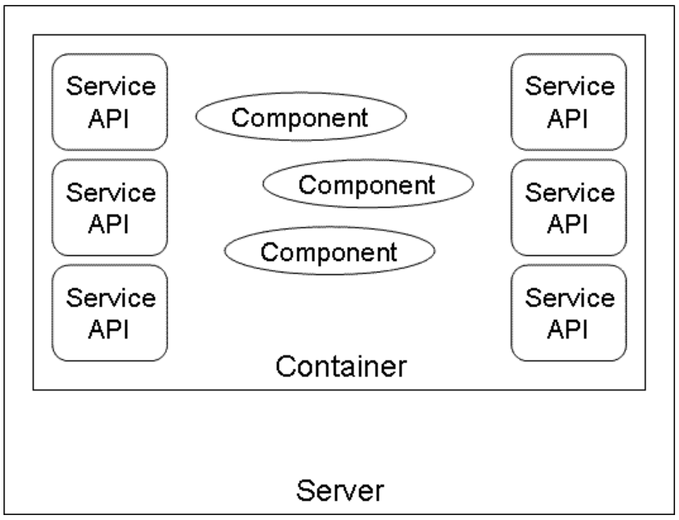

Figure 1 - The J2EE Container/Component Architecture

A specification is issued by the Java community through a process called *Java Community Process (JCP)*, which is the attempt of *Sun Microsystems* to involve the international Java community in developing Java specifications. Its introduction took place in 1998 and, since then, it has involved, on a membership basis, over 700 corporate and individuals in participating in a series of steps designed to produce high-quality, widely accepted Java specifications. The membership is regulated through an agreement, the *Java Specification Agreement (JSPA)*, between the new member and *Sun Microsystems*. A fee is due for the member. It varies according to the nature of the member, decreasing respectively for commercial entities, educational, governmental or non-profit organizations and individual members. A list of things a member can do, include: submit proposals, provide feedback on others' proposals, implement specifications and administrate the process. The *JCP* is overseen by *Sun Microsystems* through *The Process Management Office (PMO)*. Its main duty is to manage the daily running of the program. The *PMO* works in coordination with the *Executive Committee (EC)* to supervise the lifecycle of a proposed specification. *Sun Microsystems* has a permanent seat in the *EC*, while the other 15 seats are elected: 5 of them are replaced every year, the remaining 10 are ratified. The lifecycle of a successful specification, from its first submission to its maintenance, follows the steps resumed in figure 2.

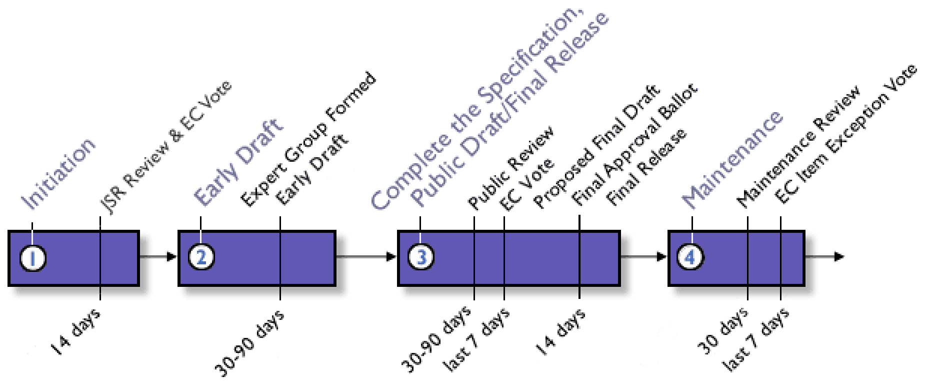

Figure 2 - Lifecycle of a Specification Under the Java Community Process

After the submission by a member, the *EC* checks that the information is in order and it does not conflict with an existing specification or *JSR*. If so, the *EC* posts the *JSR* to the *JCP* Web site for review. The proposal can be accepted, rejected or deferred. If this step is passed, a *Call For Experts*, aimed at forming the *Expert Group* (*EG*) in charge for producing an *Early Draft* in 30-90 days, is open for 15 days. The *EG* is formed by the *EC*,

choosing a subset of experts among the ones nominated by other members. Before a *Final Release* is reached, the draft is revised several times, first by the participants and then by the public. Afterwards, the maintenance phase is composed of the activities of monitoring feedback from the Java community, including clarifications and requests for major enhancements and bug fixes, and proposing changes to the specification. Other than the specification document, the *JCP* produces a reference implementation and a compatibility test suite.
The *JCP* has often been positively judged by the community of java developers as a means for being involved and sharing ideas with other developers. Some criticism has been expressed regarding the membership fees and the democracy in the process. The formers, even though *Sun Microsystems* claims that they are only collected to cover the administrative costs, have been judged too high for smaller companies and academies. The latter, according to some detractors, is scarce or lacking due to the overly strict control exerted by *Sun Microsystems* in the process (Philion, 1999).

The Specification and Some Related Implementations

The *JSR 168* starts giving some basic definitions, such as the ones for *portal*, *portlet* and *portlet container*. A *portal* is defined as “A web based application that – commonly - provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of Information Systems”, while a *portlet* as “a Java technology based web component, managed by a portlet container, that processes requests and generates dynamic content”. “A *portlet container* runs portlets and provides them with the required runtime environment”. To understand how these concepts are related, a typical working example, involving all such entities, is provided. Briefly, the portal receives the HTTP requests from the user-agent (the Web browser). If the request contains an action targeted to a specific portlet, the request is routed to the portlet through the *portlet container*, which is responsible for obtaining content fragments. The mark-up fragments generated by all the portlets included in the portal page are aggregated by the portal to compose the complete page, which is sent back to the client. If a *caching* mechanism is enabled, for portlets whose content is not changed, the previous mark-up available is used, instead of generating it again, thus saving time.
The *portlet container* is responsible for handling the portlet lifecycle. To elaborate, the portlet must implement several methods invoked by the portlet container on the occurrence of several events. In particular, there are methods for initializing and destroying the portlet, for defining the actions to undertake in response of user-interaction and for generating the mark-up fragment. The *JSR 168 API* provides a generic class which can be extended by the portlet developer, whose name is *GenericPortlet*.
As for the Relation with other *J2EE* technologies, it is worth noting that the *container/component* architecture is fully applied in the context of portal and portlets: a *portlet* represents a component and the *portlet container* the container. The specification emphasizes the relation, with analogies and differences, between a portlet and a servlet. The concept of a portlet is very similar to the that of a servlet. In particular, they both are Web components. In spite of this similarity, and in spite of the completeness of the *Servlet/JSP* specification, it has been necessary to define a new specification to deal with concepts related to portlets and portals, due to several differences among portlets and servlets. The most important of them are:

- Portlets only generate mark-up fragments, not complete documents. Nevertheless, portlets can exist many times in a portal page.
- *Servlet/JSP* specifications do not define modes or states.
- Portlets are not directly bound to a *URL*.

In spite of the above differences, the presence of a strong similarity, has allowed the participants to the *JCP* to completely define the portlet *API* as a new layer constructed on the *Servlet/JSP API.*
An important analogy between servlets and portlets is in the independence between their development and their deployment. Actually, this independence is a fundamental matter of *J2EE* philosophy. For all *J2EE* components, the deployment phase foresees the packaging of all of its files in a compressed archive together with a special *XML* file, called *Deployment Descriptor*, which holds some information useful in the deployment process. A set of portlets can be packaged together and deployed on a portlet container. The deployment descriptor holds both general information on the portlet application, including the definitions of all the portlets, custom portlet modes and window states and user attributes, and information about each portlet, such as name, title, portlet preferences, information about security, etc.
The specification defines the portlet modes and the window states. A portlet is developed to display some content to accomplish a task and it may include one or more screens that the user can navigate and interact with, or it may consist of static content that does not require any user interaction. Besides performing the task they have been developed for, the portlets could perform some more standard tasks, such as allowing user customization and providing help information about themselves. An indication on which of the above task the portlet is performing, is given by the portlet mode, whose value can be, respectively, *VIEW*, *EDIT*, and *HELP*. Portal vendors may define custom portlet modes for vendor specific functionality. The window state is a value used to define the amount of space a portlet will be assigned in the page. The output to render is determined by the portlet on the basis of such a value. Window states can be *NORMAL*, indicating that the portlet is sharing the page with other portlets, *MINIMIZED*, indicating that the portlet should only render minimal output, such as only its title, or *MAXIMIZED*, when the portlet has plenty of space available and it may be the only one displayed on the screen. Portal vendors may define custom window states.
As mentioned before, the portlet container must provide the portlet with a suitable runtime environment. This means that a portlet might obtain information and services from the container. Through the provided interface, it is possible to access context initialization parameters, retrieve and store attributes, obtain static resources from the portlet application and obtain a request dispatcher to include servlets and *JSPs*. It is worth noting that the portlet container provides the portlet with personalization support through the availability of several user information, which can include, for example, name, email, phone or address of the user. Support is offered also for storing portlet preferences. Personalization data can be defined in the deployment phase through the coding of the preferences in the *Deployment Descriptor*. This characteristic offers the chance to define different personalization data for different portlet deployments.
Security in portlet applications is largely based on the same mechanisms provided by the underlying servlet container. Authentication and role handling mechanisms leverage on it. As for the servlets, portlets can obtain the user name used by the client for the authentication and its role. Additionally, there is a mechanism to protect the content of portlets, for content integrity (preventing data tampering in the communication process) or for confidentiality (preventing reading while in transit). This mechanism allows the definition of requirements for the transport layer and makes use of *Secure Socket Layer (SSL)*.
Several commercial and Open Source solutions are now available for enterprises or service providers that want to offer their services through a Web portal of the second generation. Several portal implementations, developed with different technologies, have been considered for a survey on the adoption of *WSRP* and *JSR 168*, as shown in table 1.

Product	Website	License	WSRP Support	JSR 168 Support
BEA WebLogic Portal	http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/portal	Commercial (free to try)	YES	YES
eXo Platform	http://www.exoplatform.org	Open Source (Gnu GPL)	YES	YES
IBM WebSphere Portal	http://www-306.ibm.com/software/genservers/portal/	Commercial	YES	YES
PHP-Nuke	http://phpnuke.org	Commercial (free with adverts)	NO	NO
Apache Jetspeed II	http://portals.apache.org/jetspeed-2/	Open Source (Apache License)	YES*	YES
Microsoft SharePoint Portal Server	office.microsoft.com/sharepoint/	Commercial	YES	NO
Oracle Application Server	http://www.oracle.com/lang/it/appserver/portal_home.html	Commercial	YES	YES
* The basic version of Jetspeed II does not support WSRP. The integration of the Apache WSRP4J framework is necessary.				

Table 1 – Support of Standards in Web Portals

As it is manifest from the above table, a great part of the most popular Web portals are Java-based systems conformant to the *JSR 168*. The reason for the success of Java and the wide adoption of the specification is probably due to the *J2EE* component-oriented architecture, particularly fit for the duties of a Web-portal, and the great effort of the developers, including worldwide enterprises, contributing to the specification. In most cases, solutions supporting *JSR 168* also support *WSRP* standard. This often means that portlets developed with *JSR 168 API* and deployed in a portal that supports both standards, can be consumed remotely through *WSRP* with no additional effort from the developer.
Two of the analyzed products, *PHP-Nuke* and *Microsoft SharePoint Portal Server*, do not support *JSR 168*, since they are not *J2EE* based solutions. *Microsoft SharePoint Portal Server* supports *WSRP* standard. It also has a component-based architecture, whose basic components are called *Web Parts*, which work similarly to portlets.
All of the surveyed portals come with a basic set of portlets. Anyway, the presence of the standards has boosted the birth of several projects, aimed at the development of set of portlets and of portlet development kits.

Future Trends

The development of the market of the enterprise portals has definitely benefit from the presence of standards and specifications, such as *WSRP* and *JSR 168*. The success of *JSR 168* and, in general, of all the Java specifications is due to the fortunate choice made by *Sun Microsystems* to institute *JCP*, a successful program which has succeeded in involving commercial enterprises, academic institutions and independent developers in the development of Java specifications. In spite of the presence of contrary voices, the success of *JCP* specifications is witnessed by its widely adoption in the software market.

JSR 168 adopts the *J2EE* component-based architecture. The *API* defined in its ambit leverages on the underlying servlet specification.
The *JSR 168*, since the issue of its final release, has been adopted in almost all of the *J2EE* based Web portals, generating a market for portals, portlets and development kits, and enhancing interoperability among such products. Some proposals to enhance the specification, including aspects not currently addressed, have already been suggested by the community, and will result in the release of a new specification, the *JSR 286: Portlet Specification 2.0*. The new specification will extend the previous in regards, among others, to the support of *WSRP 2.0*, the definition of portlet filters and of an *API* for inter-portlet communication and in regards to the enhancement of caching.

References

Abdelnur A & Hepper S. (2003) JSR 168 - Java Portlet Specification Version 1.0

Ahmed KZ, Umrysh CE. (2002) Developing Enterprise Java Applications with J2EE and UML. Addison-Wesley. ISBN 0-201-73829-5

Bellas, F. (2004). Standards for second-generation portals. IEEE Internet Computing. 8 (2), 54 – 60.

Hepper S. (2006) JSR 286 - Java Portlet Specification Version 2.0

Java Community Process Program. (JCP) http://jcp.org/en/home/index

OASIS Web Services for Remote Portlets. (2003). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

Philion P. (1999). A look inside the Java Community Process. Java World (on-line magazine). www.javaworld.com/javaworld/jw-10-1999/jw-10-jcp.html

Key Terms

Web Services for Remote Portlets (WSRP): a specification standardizing presentation-oriented Web services for use by aggregating intermediaries, such as portals, approved as an OASIS standard in August, 2003

Java Portlet Specification (JSR 168): a specification document developed under the Java Community Process aimed at achieving interoperability between portals and portlets.

Java Specification Request (JSR): a document managed by The Java Community Process, submitted by one or more members to propose the development of a new specification or significant revision to an existing specification

Java 2 Enterprise Edition (J2EE): A Java platform, provided by Sun, for the development and deployment of distributed enterprise applications

The Java Community Process (JCP): A program by Sun Microsystems aimed at developing and revising the Java technology specifications, reference implementations, and test suites, involving the community of Java developers

Portal: A web based application that – commonly - provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of Information Systems

Portlet: A web component, managed by a portlet container, that processes requests and generates dynamic content.

Portlet Container: an element of the architecture of a Java portal, which runs portlets and provides them with the required runtime environment.