Java Portals and Java Portlet Specification and API

Gennaro Costagliola
Università di Salerno, Italy

Filomena Ferrucci
Università di Salerno, Italy

Vittorio Fuccella
Università di Salerno, Italy

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 (2004), 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

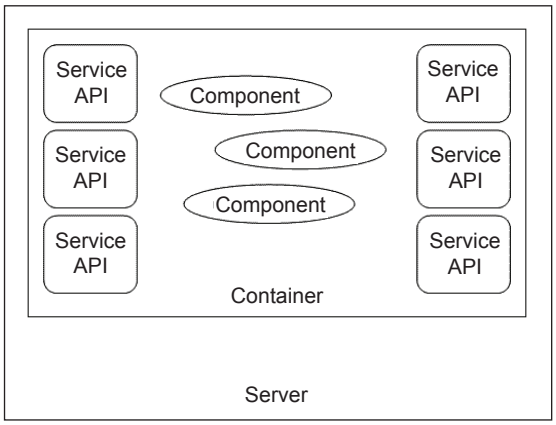
establish a standard. An early solution has been found defining the *Web services for remote portlets (WSRP)* standard: a Web service interface (defined in *Web Service Definition Language*) through which portals can interact with the remote producer's portlets. The *WSRP 1.0* specification (OASIS, 2003) was approved as an *OASIS* standard in August 2003. 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*. For example, 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 *Applet Container* and *applets*, *Web Container* and *servlets*, *Enterprise Container* and *Enterprise JavaBeans*. The *container-component* architecture is shown in Figure 1.

A specification is issued by the Java community through a process called *Java Community Process (JCP)*, which is

Figure 1. The J2EE container/component architecture



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 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 nonprofit organizations and individual members. A list of things a member can do includes: 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.

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,