Management Issues in Portlet Development

Tony Polgar
Sensis Pty. Ltd, Australia

Jana Polgar
eBlueprint Pty Ltd., Australia

INTRODUCTION

Software development methodology refers to a standardised, documented methodology which has been used before on similar projects or one which is used habitually within an orsganisation (McGovern et al., 2003). The successful software development depends on the flexible choice of software development method, and applying the right method for the job. From this perspective, the portlet development encounters new circumstances which affect the chosen method. A portal development manager must be aware of the technological properties and constraints, because there is a large (and very new) range of issues, risks and hidden costs that must be addressed in both the development and deployment processes. These issues are not well defined yet; there is no proven methodology for driving portal projects. This article provides discussion of practical approaches to the resolution of development issues and risks in portal environment. The discussed topics include implementation of portals in enterprise environment, portlet applications' high availability, portlet disaster recovery, and cost of portlet deployment. An attempt is made to forecast future trends in portlet technology at the end of the article, as well as suggest the directions for the flexible selection of methodologies and managerial experience suited to the portal development.

BACKGROUND

Enterprise portals entered the business scene as a new generation of integration services, in a logical sequence of creating ever easier access paths to enterprise information and services. One can regard portals as a happy marriage between network enabled access through the Web and specialised businesss focused access to grouped information and functions. Development of portlets has been originally regarded as yet another metamorphosis of J2EE or .NET technology. The expectations and promises of portal suppliers included powerful user interfaces, fast development using rich APIs, compatibility of portlets originating from different suppliers (JSR-000168 Portlet Specification), integration of content, and document management with functional portlets, single sign-on, and easy implementation of authorisation/authentication services. A number of questions arose immediately:

1. Is the development as mature as it would appear from the above promises?
2. Can a development manager with experience in other Web technologies easily become a successful portal development manager?
3. Is there anything specific that a portal development manager must know about the technology?
4. Are the best practices in Web development applicable to portlet development?
5. What are the hidden costs and pitfalls of portal development?

In this article we concentrate our discussion on questions 1 and 5 and at the same time, we provide the background to the answers for questions 2, 3, and 4.

PORTLETS IN ENTERPRISE ENVIRONMENT: TECHNOLOGY MATURITY

In order to understand the complexity of the development, we need to explain the container based architecture of Web and portal servers. Referring to Figure 1, all portlets run in one or more portal processes, which create Web pages and also communicate with one or more application servers. The Web server container distributes HTTP requests to application servers. Therefore, portal is a Web application with portlets sharing not only the operational parameters but also Java Virtual Machine. Consequently, if a portal application fails in any way, it brings down all other portlet applications with it. In a typical enterprise environment, some portlet applications are more critical for business than others. The sturdiness and stability of the developed product often depends on the environment and the behaviour of *neighbours*—other portlet applications sharing the portal container.

This brings about the question of how many portals should run in an enterprise environment. While there is no technical reason for running more than one portal, it is a good practice to separate critical and noncritical portlet

applications in such a way that they run in separate portals, and therefore in separate Java virtual machine environments. This way, the running and monitoring of various servers can be controlled more easily. The architect in this case is faced with the task of deciding what method should be used for integration of portlets running in separate environments and also of placing the application servers on various platforms. The obvious choice is the use of Web services for remote portlets (WSRP) but other options are available, such as I-framing or data-oriented Web services. A simple method of integration is the use of navigational means to direct the user to various portals, without making it obvious, which particular portal is being used.

Loose Coupling of GUI and Functional Components in Enterprise Environment

Loose coupling of variety of components is the trademark of service-oriented architectures (SAO). The use of WSRP supports aggregation of fragments produced by portlets running on a different (remote) platform. The service is *presentation oriented* which means that the fully formed mark-up fragment is submitted for aggregation to the local portal. The integration occurs with the exchange of SOAP messages containing HTML fragments.

Since the remote portlet runs in a remote portal container, the stability of the *home* system is vastly increased. On the other hand, there are costs in terms of response time, and maintenance of another system (which may run a different operating system and portal container). The installation of WSRP is an administrator's job, so while the portlet code does not need to change whether the portlet is local or remote, the administration cost is very different (Polgar, Polgar, & Wilkinson, 2006).

However, another architectural concept can be used for achieving the same goal of making the critical applications stable and separated from noncritical ones: use portal only as a container for a thin-veneer UI layer and place the majority of the functionality on another platform, preferably providing Web services to the portlets in portal. It should be noted that the remote application may provide interface complying with WSRP, even though the application itself is not implemented as a portlet. This can be seen as *data oriented* Web services, as only data are provided by the remote service. Such solution could be more complex than creating new interface to this remote Web service.

A further option is to place the application in a separate application server and provide connectivity through some sort of messaging, such as MQ or JMS.

The user experience does not need to suffer from the separation of the portlet applications as the user interface pages can and should integrate portlets which originate from different application servers.

It is also possible to mix portlet and Web applications. As to the decision of which application should be implemented in portlet and which in pure Web technology (such as servlet), a good rule of thumb is the requirement for the appearance of the user interface. If a portlet application makes use of multiple small windows on one Web page, single sign-on, and some portal services (such as deployment of the same portlet on multiple pages, interportlet communication, and various portlet-style customisations), then the use of portlet APIs is justified. Otherwise, building a Web application (such as a servlet and a JSP) is just as effective, provided enough attention has been given to the quality of the user interface and navigation.

It is apparent that there are always several sound solutions to fulfil the stakeholder needs. All strongly adhere to the principles of SOA and separation of concerns advocated in many papers (Grassi & Patella, 2006).

Cost of Loose Coupling and Separation

The development manager and stakeholders might wish to consider the cost of maintaining a relatively high number of platforms if they implement any of the above options, and weigh its value against building a simpler platform but with higher risk of discontinuity of service for the whole installation.

Careful considerations should be also given to the user experience in cases where the remote application is not available. In such cases, the portlet should gracefully announce its unavailability, while the rest of the portal applications continue working.

Adoption of Web Services and Service-Oriented Architecture

The use of SOA, and specifically Web services, provides an opportunity to integrate loosely coupled services originating from various platforms, making it possible to separate business critical and noncritical applications. The main expected advantage of implementing SOA is the reduction of costs, and high level of agility and flexibility. Among the top three reasons for not pursuing an SOA strategy are the ability to reuse services in the future (20.4%); ability to lower integration costs (17.6%); and the ability to enable faster delivery of projects (16.2%) (Putting the SOA infrastructure together: Lessons from SAO leaders).

New Web service specification JAX-WS 2.0—the new version of the Java API for XML-based RPC (JAX-RPC) is mostly concerned with the improvements of typing and support for document oriented services. The ease of invoking Web services from JavaScript is offset by the lack of annotation options (Vinoski, 2006) thus making the implementation of SOAs more complex.