Implementation of a Web Services Based Recruitment Platform for Student Jobs

Rami Hansenne[1], Veerle Van der Sluys[1], and Bartel Van de Walle[2]

[1]Actonomy NV, Stapelplein 70,
9000 Gent, Belgium

[2] Tilburg University, Department of Information Systems and Management,
5000 LE Tilburg, the Netherlands

Abstract. This paper discusses the implementation of a web services enabled application framework allowing distributed university websites to register and exchange information with a centralized recruitment server regarding small or odd jobs that can be performed by their students.

1 Introduction

Web Services are the next logical step in the evolution of distributed computing. Based on open industry standards, web services enable integration and communication of applications in a loosely coupled, simple and platform independent manner.This article discusses the use of web services in the context of an HR recruitment application, which we have named OddJobs. The OddJobs Central Information System manages a multitude of job openings targeted specifically at university students. These jobs are typically small jobs – or odd jobs – which can be done in a few hours or days at most. This excludes in principle any full time or part-time jobs, for which dedicated job sites are plentiful these days. The main concept behind OddJobs is the centralized management of these jobs, such that a job opening posted on one university website is available to all other registered universities as well. To achieve this objective, certain considerations need to be made. The number of client websites is not static as new member sites can join at any time. Therefore, an architecture needs to be present allowing websites to connect to the centrally hosted OddJobs, independent of the website's hosting location. Furthermore, the implementation details of these websites are unknown and may be set up by third parties. In other words, the server connection mechanism needs to be transparent and independent of the client platform (jsp/asp/…). Since these client websites are hosted in a distributed manner, a means of communication was required, capable of passing through firewalls without problems. Finally, it should be relatively straightforward for client websites to integrate the OddJobs functionality into their portal. In order to meet these requirements we chose to use a SOAP web services architecture, as it allows clients to connect to the services, independent of the client platform or location. SOAP uses HTTP as its transport mechanism, which avoids potential problems with firewalls and thus makes it an ideal choice in comparison to other RPC architectures. The general principle is that OddJobs offers a number of well defined services, such as registration, user login, job catalogs or categories, and posting resumes to the system. These resumes are presented in the format of questionnaires, and can be filled in by the students to allow the backend to identify the most suitable jobs for the applicant. Participating websites need to integrate a SOAP client capable of handling the requests/ responses to and from these services. To ease client integration, a SOAP client module in the form of a JSP taglib was implemented, such that the OddJobs functionality can be incorporated with a minimal required knowledge of the web services protocols.

2 The OddJobs platform

The OddJobs platform consists of several main modules, each consisting of a number of specific submodules, as illustrated in Figure 1.

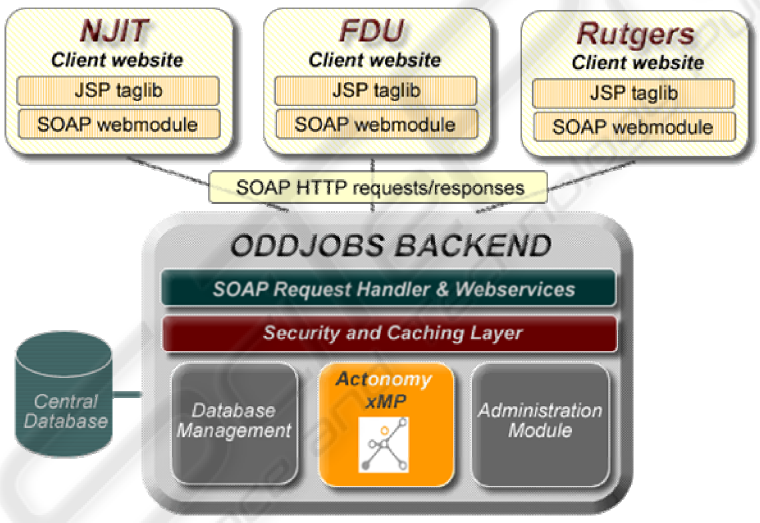

Fig. 1. Overview of the OddJobs Platform

2.1 Data handling and business logic

All jobs submitted to OddJobs via a client website are stored and maintained on a central server. Client web sites need not concern themselves with the data handling or business logic involved in managing job submissions or registrations. They can instead focus entirely on presenting this functionality in a userfriendly manner to their respective audiences. It is the task of the centralized back-end to process the client requests. The J2EE compatible (and therefore platform-independent) back-end contains several submodules for doing just that. The data module is composed of a central database with an object abstraction layer, shielding the object oriented logic business layer from the relational database access and allowing easy switching of database dialects, thus making the back-end portable over a multitude of database vendor implementations. A separate module is in charge of modeling each of the system's use cases. These use cases may for instance be 'creating a job catalog', 'retrieving job information for a certain client website' or 'logging out a user'. One specific business logic submodule, the xMP matching engine, handles the matching of a job applicants cv and/or preferences and with the job specifications. It does so in a fuzzy manner in order to determine the best possible matches in case no exact matches exist (Hansenne *et al*., 2004). A centralized administration module takes care of the server configuration and management of functionalities that are not available to the client websites, such as management of website permissions.

2.2 The web service providers

The SOAP Web Services are a set of Java libraries which connect to the backend to perform certain services, based on the input they get from the client. These services mirror the use cases provided by the business logic layer. The data transfer between client and server is abstracted away from the underlying SOAP XML transmission and processing, by encapsulated it into POJOs (Plain Old Java Objects). These are (de)serialized to and from XML by the Soap request handler (described later on) and streamed to the client websites. The mapping of data objects to XML has been made as generic as possible to allow an easy modification of the data objects structure (for instance adding extra fields), without having to redesign the services themselves.

2.3 The web service clients

The websites that join the OddJobs platform need to implement a SOAP client that is capable of connecting to the SOAP services. Such a client has been implemented for JSP based websites. The clients actually consist of proxy objects (client stubs) which mirror the functionality of the web services. This means that the websites can use the web service classes as if they were locally available to them. All of the XML mapping and streaming is handled behind the screens and is thus transparent to the client developer. The XML received by the proxy objects is deserialized into data objects which can then be read and modified by the client.

2.4 The JSP taglib

Using the proxy objects directly in the web pages would be quite messy and would make the source code of the JSP pages hard to read. It would also require the client side programmers to have a thorough knowledge of the inner workings of the proxy objects. To avoid this, a JSP tag library has been constructed which handles all of the communication with the proxy objects. The JSP developers working on implementing the OddJobs functionality into the site need only know how to use and configure these JSP tags. The JSP tags make the data bean objects available to the jsp page for getting/setting properties and submitting data to the back-end.

3 OddJobs Webmodule Architecture

In this section, we describe the architecture of the different components that make up the web module. The Java Beans data objects encapsulate the data that need to be passed around by the SOAP services and SOAP clients. They avoid having to use method calls that require dozens of arguments. For instance, all details concerning a user (login, password, address, email, company or university, gender, …) can be encapsulated into a single object. Even if the exact amount of fields contained in the beans changes, the SOAP/Taglib implementation remains independent of the exact contents of a data object, as mapping/serialization is handled in a generic way through introspection.

All web services are handled by dedicated service managers. These managers can be used transparently by the client, as if they were available locally. For instance, the *User Manager* handles all requests that are related to the OddJobs users: user login/logout; user registration; user profile updates, etc. The catalog manager handles job catalog or job category related requests: retrieval of a job catalog; retrieval of the questionnaire attached to a certain catalog; posting the answers to a questionnaire back to the system, etc. In the future, additional managers can be added, without worrying about the SOAP remoting code. By implementing a manager interface, new managers can automatically be exposed to clients without having to write a new invocation handler.

The Web service managers do nothing more than calling the corresponding methods of the backend and encapsulating the sent and received data into Java Beans data objects. All non primitive data needs to be type mapped however before it can be serialized into an XML SOAP call. On the server side these mappings are defined as deployment descriptors and can easily be modified and redeployed (this is only required if the object types sent and returned by services change, not if the structure of the beans change). On the client side however a mapping must also be defined before the SOAP call can be invoked. Most web services deployment tools generate hard-coded client stub and server skeleton classes containing the logic of type mapping and serializing the transferrable data. This method has a serious disadvantage: whenever the deployment parameters change or the structure of the passed data needs to be modified, the client stub code needs to be regenerated. This of course is not a valid option when dozens of clients connect to a back-end solution as each of these clients would need to be updated.

To alleviate this problem, all SOAP requests/reponses pass through a central Call handler. In stead of defining the serializers for every argument or returned object, every time a soap call is made, the SOAP Handler dynamically introspects the Java beans it receives and create the required type mappings at runtime. Furthermore, the Handler is capable of proxying the serverside service manager objects as local objects through introspection of the manager interfaces. The handler will create a dynamic proxy object for every service request and handle all the invocations on the proxy by relaying the method call through SOAP. This means that the client application (taglib) will only need to use the proxy objects without knowledge of SOAP or remote method invocation. As for the deployment details, these are also configured at runtime by the Call Handler, by parsing the web services description (WSDL) file made available by the back-end. Any change to the deployment specifications is thus automatically noticed by all clients and configuration changes are performed as required.

For an even more flexible setup the WSDL file(s) describing the web services can be published to a UDDI (Universal Description Discovery and Integration) server, thus allowing the clients to query the UDDI server in case a call to the back-end fails. Based on the server response the clients can then dynamically update their configuration and automatically reconnect to the back-end using the new setup.

4 Caching

The OddJobs platform features both front-end (website) and back-end caching. The back-end caches database-queries, while the front-end limits the number of soap-requests that get sent to the back-end. Both caching mechanisms are absolutely essential. After all, the back-end is constantly and concurrently handling requests from numerous client websites, so it is important to limit the number of database queries while maintaining a synchronized state for all clients. Soap calls are also costly operations as they require XML parsing on both client and server side. They are also heavily dependant on the connection speed between the back-end and its clients. As few as possible of these remote calls should be made. All objects transferred to and from the web services can be cached: individual and lists of catalogs, offers, content items, user profiles, questionnaires, preferences, as well as form data (countries, nationalities,…).

A typical client request is handled as shown in Figure 5. The web client calls the correct service proxy to handle the request. The proxy checks it’s local cache for an existing response to the specific request parameters it is passed. If it finds a hit, the result is returned without performing a soap call. In the other case, the corresponding back-end web service is called. The back-end will check its cache and based on the results, either directly return a cached response or perform the business logic operation including database access. To make sure all client websites are synchronized, the back-end may send a synchronization signal to all the client websites, instructing them to update a specific part of their cache. This action is performed in case the client request modified the platform state (for example, when a job gets added/removed) and consists of an http connection with an event listener servlet on the client side. The original client web site receives the server response and caches it locally. he back-end cache keeps a record of all the items requested by all the websites, while the website cache only keeps track of items it requested itself. That’s why a miss on the local client cache might still produce a hit on the back-end cache, even though they all get cleared on the same event notifications. The caches are based on soft references, to make sure items automatically get de-allocated when server memory usage peeks. A time-out can be configured to ensure the cache gets cleared from time to time, in case something goes wrong with the notification system; for example, when the back-end temporarily can't notify the clients due to network problems.

5 Conclusion

The OddJobs platform serves as the backbone of what is intended to become a dynamic student job marketplace (Van de Walle 2003). Indeed, while the platform has been developed in close collaboration with early adapters (in particular the New Jersey Institute of Technology in Newark), the OddJobs roll-out throughout Europe in 2004 is a major objective of Actonomy. Based on the technological framework described in this paper, and the underlying fuzzy matching technology described elsewhere (Van de Walle et al., 2002; Hansenne *et al*., 2004), the authors aim to establish a thriving virtual community of European students using OddJobs to find small jobs they can take up during their studies (Bieber et al, 2002).

References

1. Actonomy 2002. OddJobs API - Actonomy Technical Report 10/2002.
2. Bieber, Michael, S.R. Hiltz, E.A. Stohr, D. Englebart, J. Noll, M. Turoff, R. Furuta, J. Preece and B. Van de Walle 2002. Towards Virtual Community Knowledge Evolution, *J. of Management Information Systems* 18 (4) Spring 2002, 11 – 35.
3. Hansenne, R., V. Van der Sluys and B. Van de Walle, 2003. Smart Web Services in Action: Student Odd Jobs on University Websites. In *Proceedings of the International Conference on Information Technology: Research and Education* ITRE2003 (Newark, New Jersey USA), 255 – 256.
4. Hansenne, R., J. Van Poucke, V. Van der Sluys and B. Van de Walle 2004. Design and implementation of a scalable fuzzy case-based matching engine. *Accepted for ICEIS2004* (Porto, Portugal, April 2004).
5. Monson-Haefel Richard, 2003. J2EE Web Services. *Addison-Wesley*.
6. Nagappan Ramesh, Skoczylas Robert, Sriganesh Rima Patel, 2003. Developing Java Web Services: Architecting and Developing Secure Web Services Using Java. *John Wiley & Sons*.
7. Snell James, Tidwell Doug, Kulchenko Pavel, 2001. Programming Web Services with SOAP. *O'Reilly & Associates*.
8. Van de Walle, B. and V. Van der Sluys, 2002. Non-symmetric Matching Information for Negotiation Support in Electronic Markets. In *Proceeding of the International Workshop on Information Systems* EuroFuse2002 (Trento, Italy), 271 – 276.
9. Van de Walle, B., 2003. A relational analysis of decision makers' preferences. In *International Journal of Intelligent Systems* 18, 775 – 791.