SERVICE ORIENTED STATUS MONITORING FOR DIP MIDDLEWARE

B. Copy, CERN, Geneva, Switzerland#

Abstract

DIP [1] is a middleware infrastructure developed at CERN to allow lightweight communications between the various distributed components of a control system (such as detector control systems or gas control systems). DIP publications are currently subject to a lack of visibility from the CERN general purpose network and a lack of formal service level agreements between information publishers and consumers. The DIP contract management system addresses these limitations by providing a publication monitoring tool that can make available both publication data and publication status on the web through a javascript API for inclusion in web pages and integration with advanced AJAX libraries (such as the Google Web Toolkit Visualization API). It also performs status information logging, and advertises such information in the form of DIP publications (to ease integration with SCADA systems such as PVSS). We will demonstrate how complex structured information can be easily made available to a large array of consumers through the usage of the Spring framework and the multiple configuration based adapters it offers to a vast choice of communication protocols.

STATUS MONITORING FOR DIP

DIP is a CERN developed middleware platform [1] that provides services allowing the exchange of information between control systems or between the distributed components of a control system :

- A "naming" service that exposes a tree of named publications (i.e. a publication is a set of name-value pairs)
- A C++ / Java API to support the creation of publications or subscriptions to existing publications.
- Integration layer with SCADA or measurement systems, such as PVSS or NI LabVIEW.

The DIP naming service (DNS) is a useful indirection or "publisher lookup" mechanism but not a fully fledged directory service, in the sense that publication entries are non-persistent. It is therefore impossible to determine whether such non-existent publication is missing because it is temporarily unavailable or because it has been retired or renamed. And because the DNS does not incorporate any notion of timing, it is also impossible to apply time-based constraints, such as minimum publication refresh times or publication status time outs.

Numerous systems at CERN rely on large numbers of publications being active at the same time and originating from a large variety of infrastructure or accelerator services. Such lists of publications are essentially publication contracts, binding the publisher to a consumer, but such associations were not registered anywhere.

The DIP monitoring service aims at solving these limitations by making publications contract status widely available on the web and let expert users identify which publications are at fault from any web browser client on the CERN network.

SERVICE ORIENTED ARCHITECTURE

Service Oriented Architecture (SOA) is a software design approach that compasses information systems as a set of loosely coupled and interoperable services, thereby improving their testability and scalability [2]. Services in the SOA sense are typically defined in terms of atomic and stateless actions, thereby making them easier to combine and encapsulate in order to build up higher orders of functionality.

Loose coupling in SOA is achieved :

- On one hand by using standard messaging protocols that guarantee platform independence;
- On the other, by the usage of meta-data that describe available service calls, such that their discovery and invocation can be easily automated.

As a mean to deal with the complexity of building large information systems, SOA lets developers focus on business relevant code and make abstraction of communication protocols, programming languages or messaging format requirements. SOA implementations rely on off-the-shelf information protocols, such as SOAP [3], CORBA or Microsoft DCOM. To this day, the HTTP protocol, due to its simplicity and ubiquity, is a particularly suitable foundation for most SOA implementations.

SOA IMPLEMENTATION WITH THE SPRING FRAMEWORK

The Spring framework (Spring) [4] is an open-source framework that also focuses on re-usability and testability of complex information systems, but at the programming language level. While SOA focuses on high-level business services, Spring acts at the object instance level and encourages interface-first design and unit testability by implementing the "dependency injection" design pattern [5].

As demonstrated in Figure 1, Spring assists developers by letting them write service implementations that are free of any traditionally complex concerns such as transaction support, security, resource pooling or thread safety. Such cross-cutting aspects can be weaved into and amongst object instances as and when such needs emerge.

# brice.copy@cern.ch

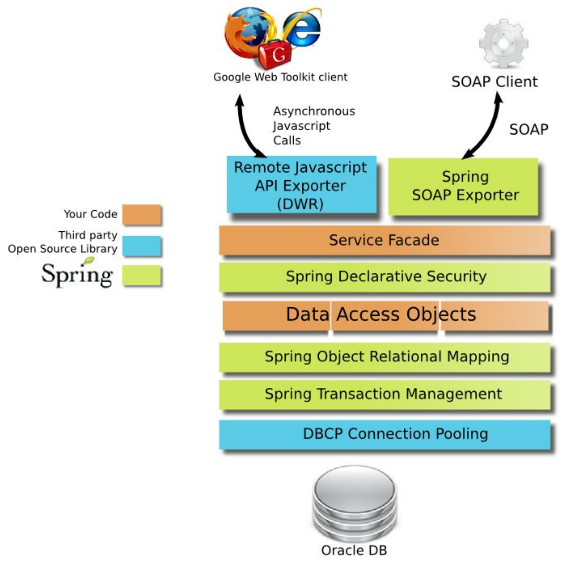
Google Web Toolkit client
SOAP Client
Asynchronous Javascript Calls
SOAP
Remote Javascript API Exporter (DWR)
Spring SOAP Exporter
Your Code
Third party Open Source Library
Spring
Service Facade
Spring Declarative Security
Data Access Objects
Spring Object Relational Mapping
Spring Transaction Management
DBCP Connection Pooling
Oracle DB

Figure 1 : Spring service implementation stack.

The Spring framework incidentally provides support for a large range of SOA compliant remoting protocols, such as SOAP, through the usage of service exporters. Service exporters transform an interface definition into a fully fledged service, generating on-the-fly metadata (for instance WSDL definitions [6]) and providing messaging support functions such as data marshalling and broking.

LIGHTWEIGHT SOA

One great criticism of SOAP is its heavy reliance on XML, which carries both an important performance penalty and added complexity for platforms without native XML processing support.

There are many lighter alternatives to SOAP as a communication protocol, such as DWR, JSON-RPC [7] or REST [8] which yield the same advantages in numerous situations and are a more natural fit to their service client platform.

Direct Web Remoting (DWR) [9] in particular is a Javascript centric remoting protocol that exhibits SOA characteristics (namely support for service call metadata and a standard underlying messaging protocol) in a form that any Javascript enabled web browser can use natively [10].

DWR is used by the DIP monitoring tool to expose Spring configured object method calls to any web browser. It is therefore a simple and efficient way to expose any service implementation to a large population of users.

Such lightweight service implementations are conveniently reused to implement AJAX web applications [11].

AJAX web applications clearly separate the data from its presentation. Presentation information is served in most cases as a mix of HTML DOM documents and CSS styling directives. Data is served asynchronously (*i.e.* independently from the surrounding HTML DOM document), either in XML or in a more natural Javascript object notation (also referred to as JSON [7]). This results in lower server resource consumption : the static presentation layer is served once per client and cached, the rest of the traffic being dedicated only to data exchanges.

RICH WEB CLIENTS WITH GOOGLE WEB TOOLKIT

AJAX applications rely on two important features of modern web browsers :

- Their capability to emit HTTP requests asynchronously, without affecting the HTML page currently displayed.
- Their capability to manipulate DOM structures programmatically with Javascript.

This second point has made Javascript the most obvious choice for writing AJAX applications. As a scripting language, Javascript currently suffers from differences between browser implementations (forcing developers to tune their code for specific browsers or specific browser versions), poor programming tooling (*e.g.* no debuggers can offer cross-browser support) and limitations inherent to scripting languages (*e.g.* lack of strong typing support, poor support for code refactoring). Such shortcomings make it inappropriate for writing large and complex applications.

The Google Web Toolkit (GWT) [12] is an open-source development platform used to write Java event-based applications (much similar to desktop applications) which can be compiled to Javascript. The resulting applications will run in any Javascript enabled web browser without any need for plugins or a Java virtual machine runtime; Such applications can be deployed wherever a static HTML page could be published. GWT applications are insulated from cross-browser issues, can be unit tested and functionally tested using familiar testing frameworks (such as JUnit) and benefit from mature Java tooling (debugging, execution profiling, code refactoring, build automation).

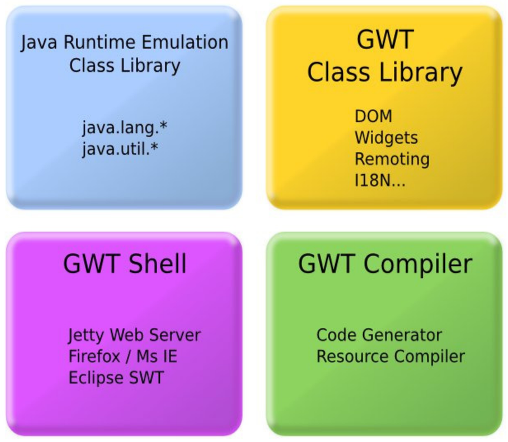

Figure 2 : Components of the design-time GWT platform.

Figure 2 provides an overview of the technology stack provided by GWT at design time.

The Java Runtime Emulation class library is a near complete standard Java 5 core classes library, with the notable exception of features that have no equivalent role in a web application (for instance, object serialization support).

The GWT class library provides essentially toolkit widgets that are used to implement event-based user interfaces (such as found in desktop applications).

The GWT shell is a web browser / web server combination used at design time to simulate the behaviour of a deployed application. At design-time, GWT applications are still executed on top of a Java virtual machine, and can therefore be tested, debugged and profiling with standard Java tools (such as the Eclipse development environment).

The GWT compiler comes into action at deployment time, when the GWT application is ready to be deployed. The GWT compiler performs Java source to Javascript translation, code optimizations, resource packaging and browser specific alternative implementations. The output result is a set of web resource files (HTML, Javascript, images, stylesheets...) that can be readily published.

Code reuse is also strongly promoted by GWT, as it naturally supports Java distribution formats (*i.e.* JAR files) and package based naming. Numerous GWT extensions (such as the Google Visualization API) and advanced widget libraries (such as EXT-GWT [13]) have been published in the few years since the platform's inception, offering a richness of presentation on a par with most sophisticated desktop applications.

GWT also offers native Javascript integration through the usage of a native interface mechanism comparable to Java Native Interface (JNI). This offers the possibility to integrate existing Javascript code base, at the cost of having to deal with low-level Javascript concerns (such as memory leaks, lack of cross-browser debugging).

The DIP monitoring application uses GWT to offer rich interactions and complex user interfaces (such as a recursive tree grid) which would require a large amount of effort and yield a poorer user experience, had they been implemented in a page based web application framework.

CONCLUSION

By employing modern dependency injection frameworks such as Spring, it is now possible for a Java or Microsoft.NET service development team to focus solely on core business functionality and add at the cost of an affordable configuration effort all the essential features offered by the most advanced service oriented implementations. These services can be deployed on a variety of platforms, thereby avoiding vendor lock-in.

Lightweight services can in turn be easily integrated in rich web based applications, providing a desktop-like experience, with support for mature software development tooling and without having to deal with the usual hindrances imposed by Javascript code authoring.

REFERENCES

[1] W. Salter et al., “DIP Description” LDIWG (2004); https://edms.cern.ch/file/457113/2/DIPDescription.doc.

[2] K. Channabasavaiah, K. Holley and E. Tuggle, “Migrating to a service-oriented architecture”, IBM *DeveloperWorks*, 16 Dec (2003); http://www.ibm.com/developerworks/library/ws-migratesoa/

[3] W3 consortium, “SOAP Specifications”, April (2007); http://www.w3.org/TR/soap12.

[4] Springsource, “Spring 2.5 reference manual”, November (2008); http://static.springsource.org/spring/docs/2.5.x/reference/

[5] M. Fowler, “Inversion of Control Containers and the Dependency Injection pattern”, January (2004); http://martinfowler.com/articles/injection.html.

[6] W3 consortium, “Web service description language”, March 2001, http://www.w3.org/TR/wsdl.

[7] R. Koebler, “JSON-RPC 2.0 specification”, JSON-RPC working group, May (2009); http://groups.google.com/group/json-rpc/web/json-rpc-1-2-proposal?pli=1.

[8] C. Pautasso, O. Zimmermann, F. Leymann, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision", *17th International World Wide Web Conference (WWW2008)* Beijing, China, April (2008).

[9] J. Walker, “Direct Web Remoting”, July (2009); http://directwebremoting.org/dwr/index.html.

[10] P. McCarthy, “Ajax for Java Developers : Ajax with DWR“, IBM Developerworks, November (2005); http://www.ibm.com/developerworks/java/library/j-ajax3/

[11] J. Garrett, “Ajax: A New Approach to Web Applications”, February (2005); http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[12] B. Johnson et al., “Google Web Toolkit”, November 2006; http://code.google.com/webtoolkit/

[13] J. Slocum, “EXT GWT: Rich internet applications for GWT”, February (2008); http://www.extjs.com/products/gxt/