Eclipse Open Healthcare Framework

Eishay Smith\textsuperscript{1}, James Kaufman\textsuperscript{1}, Kelvin Jiang\textsuperscript{2}, Matthew Davis\textsuperscript{3}, Melih Onvural\textsuperscript{4}, Ivan Oprencak\textsuperscript{5}

\textsuperscript{1} IBM Almaden Research Center, \textsuperscript{2} Columbia University, \textsuperscript{3} University of Oklahoma, \textsuperscript{4} North Carolina State University, \textsuperscript{5} University of Pennsylvania
1. Executive Summary

This is a Position Paper for the Server-Side Eclipse Symposium at the 2007 European Eclipse Summit in Esslingen, Germany.

The problem with designing a generic component to be used by an arbitrary application is that the developer does not know where the component would eventually be deployed. The application might be running on a server or desktop. With service-oriented architecture (SOA) gaining momentum, we witness server-side applications taking a significant presence on the desktop. Using web services, desktop applications are using more logic running on the server. Using web services, the client and server sides can interoperate seamlessly even when they are running in different environments. Furthermore, we see desktop applications being replaced by rich AJAX-driven web applications, providing users good experience and flexibility.

The above applies strongly to the healthcare IT market. Healthcare systems come in all of the above flavors, and runs on different platforms, yet they want to interoperate amongst each other which is why the Open Healthcare Framework [1](OHF) provides a server-side solution. The paper describes the Eclipse OHF project and the reasons why it chose to use the Eclipse server-side technology. Then, the architecture briefly describes how the OHF Plug-ins expose functionality using web services.

2. Introduction

The OHF addresses part of a need to improve the levels of interoperability between applications and systems within and across healthcare organizations – corporate and regions. The project implements extensible frameworks, exemplary tools, and key health informatics standards wrapped as plugins. The project supports objectives of many government health departments to encourage the use of interoperable open source infrastructure to lower integration barriers. Independent Software Vendors (ISVs) may use the framework, components and tools created by this project to build desktop, gateways and server applications running on healthcare IT infrastructure.
3. OHF on the client side

Many of the OHF components are implementations of healthcare related standard protocols, data structures, encryption & security tools etc. The conventional way of using OHF is to create an Eclipse RCP application with user interface and workflow logic that use the OHF components, as illustrated in Figure 1.

Figure 1 OHH components embedded in a standard Eclipse RCP application

The typical healthcare applications one would build with OHF components are Electronic Medical Record (EMR) or Patient Health Record (PHR). These products run at either a clinic or a patient’s home. Naturally, healthcare applications are deployed in many other environments, though in most of these environments client/server is the preferred architecture.

4. Embracing the Server Side

4.1. The problem

Most OHF Plug-ins are designed to be used as components in a workflow typically executed by a healthcare application. We see two main problems with embedding them only in the traditional Eclipse RCP application model.

The first problem is that as soon as the application targets more than one user, the market demands a client/server solution. Many healthcare applications deployed in clinics and hospitals have web interfaces to their back end to facilitate this need.

The second problem is that most existing healthcare applications run in a .NET[7], LAMP[8], or MUMPS[9] environment. Very few of them use the Java environment and hardly any of them use the Eclipse RCP environment. We anticipate that it will take several years before Eclipse RCP applications appear in this domain.
This situation poses a problem to the OHF community. We wish to be relevant to existing healthcare applications today regardless of their runtime environment. If OHF does not meet market demands soon enough then we will see similar projects duplicating the OHF efforts in other environments. The OHF goal is to create a community that provides implementations of the healthcare standards for interoperability built on a standard (Eclipse) framework. A multitude of non-standard projects for interoperability (even if open source) will not provide convergence around a common set of open standards. Moreover, as in most open source projects, if there is no community around a single common project, the individual projects (without any users) will slowly wither away.

4.2. Our solution

Our solution is to use Server-Side Eclipse and Axis[10] to expose the functionalities of OHF components via web services. This is the OHF Bridge [6] subproject of OHF. The OHF Bridge is an “OSGi on Server” runtime that embeds OHF Plug-ins and exposes a subset of their functionalities as web services. Using the OHF Bridge, applications residing in a SOAP-enabled environment (namely PHP and .NET solutions) may take advantage of OHF and its list of healthcare applications.

4.3. Benefits

Most existing runtime environments are capable of making SOAP transactions. By moving to the server-side and exposing the OHF components as web services, the OHF project is now relevant to existing healthcare applications.
5. OHF on the Server Side overview

The OHF Bridge uses OSGi on Server and Axis to run the OHF Plug-ins within the Tomcat web container. The web container may be any container (e.g. Jetty), and it may be embedded in an application server (e.g. Geronimo). Inside the web container, the bridge runs the ‘OSGi on Server’ servlet. The OSGi environment contains the runtime plug-ins along with the OHF Plug-ins and the Axis engine. The Axis engine has a web service that accepts SOAP calls and translates them to the relevant API that the OHF Plug-ins exposes.

Using an adoption layer, the OHF Bridge simplifies transactions that may require several calls and translate workflows into one SOAP call. Another role of the adoption layer is to translate the Eclipse Modeling Framework (EMF) models we use into simpler SOAP messages. The XML representation of the EMF modules that describes the healthcare standard document schemas are complex and not all environments can parse them easily. The intent is to enable even the simplest SOAP libraries (e.g. PHP SOAP) to work with the OHF data models.

Figure 3  The OHF Bridge as a white box
6. Results

Using the OHF Bridge, OHF has had tremendous success. In a very short time, several ISVs have integrated OHF functionality into their applications. The ISVs using OHF are distributing a range of solutions from open source to proprietary, using .NET or LAMP. Six ISVs are working on OHF integration in their applications and using it in National Healthcare Interoperability Networks (NHIN), as well as in the 2007 IHE Connectathon event and in the 2007 HIMSS Showcase. This server-side option for using the OHF hides the complexity of implementing the standards so effectively that some of these ISVs were able to implement the components and successfully demonstrate basic functions with only a few hours of work.

7. Conclusions

Traditional Eclipse Plug-ins that may be useful as web services should seriously consider the server-side option demonstrated in the OHF Bridge architecture. Waiting for the market to adopt the Eclipse RCP environment would take too long, and targeting only client side applications is very limiting (although this is still a practical alternative for Java applications today).

The OHF project is a healthcare project in its core. The development around exposing the APIs of the OHF Plug-ins as SOAP calls was groundbreaking since there are no alternative open source solutions available today. We hope to see Eclipse projects that will facilitate this task in a convenient way, or even automate it entirely.

Giving Eclipse Plug-ins the ability to participate as first class citizens in the new SOA era is crucial to Eclipse future.

References