Project HealthDesign Common Platform Components

Technical Specifications

Proposed Framework for Web-Service Interfaces

December 10, 2007

Sujansky & Associates, LLC

1. Introduction

This document is a memorandum to describe the planned technical approach for the Project HealthDesign common platform components (CPCs). Specifically, the document describes the technical mechanism by which personal health applications (PHAs) will request and receive services from one or more CPCs. The goal of the document is to solicit feedback on the feasibility of the planned interface approach from each of the individual PHAs involved in the project.

2. General Technical Framework

The technical framework for the planned approach is based on *web services*. Web services are software services provided by remotely hosted server applications over communication channels based on internet protocols (i.e., HTTP, FTP, SMTP, etc.). Web services are typically specified and provided in a platform-independent and language-independent manner, such that, for example, a PHA written in Visual Basic and running on a Windows desktop computer in Tennessee could request and receive a drug-drug interaction check from an application written in Java and running on a Linux server in California. The advantage of this approach is that it creates an open and interoperable environment for providing or accessing software services anywhere the internet reaches.

For web services to work correctly, there must be a specific mutual understanding between the application requesting services and the application providing services as to (1) what services are available, (2) what data those services operate on, and (3) how requests for those services and responses to those requests are formulated and communicated. The manner in which this understanding is expressed must be independent of any specific programming languages or computing platform, yet be *map-able* to the large variety of programming languages and computing platforms in which software applications are actually implemented.

The technical approach that we propose is based on a set of technologies commonly used for defining web services in a platform independent way: Web Services Definition Language (WSDL), eXtensible Markup Language (XML), and Simple Object Access Protocol (SOAP).

1. *Web Services Definition Language*: WSDL is a language to specify the *interface* to a web service, i.e., the set of operations that the web service provides, the types of data that the web service understands, and the ways that applications can communicate with the web service. For example, the following WSDL excerpt describes a web service (or "port") that determines whether a medication is a controlled substance.

```
<portType name=”medClassifier”>
   <operation name=”isControlled”>
      <input message = isControlledRequest”/>
      <output message = isControlledResponse”/>
   </operation>
</portType>

<message name=”isControlledRequest”>
   <part name=”inputParameter” element=”medName”/>
</message>

<message name=”isControlledResponse”>
   <part name=”outputValue” element=”booleanResponse”/>
</message>

<types>
   <schema>
      <element name=”medName” type=”string”>
      <element name=”booleanResponse” type=”boolean”>
   </schema>
</types>

<binding name=”medClassifierBinding”>
   <soap:binding transport=”http://schemas.xmlsoap.org/soap/http”/>
   <operation name=”isControlled”>
      <soap:operation
soapAction=”https://medServices.org/classifier”/>
   </operation>
</binding>
```

   In this example, the `<portType>` element describes the operations that this web service provides (in this case, just a single operation – "`isControlled`"). The `<message>` elements describe the contents of the messages used to request the operation and to provide a response. The `<types>` element defines the data types of the message contents (which may be primitive data elements, as in the example, or large and complex objects, such as a prescription record). Finally, the `<binding>` element specifies how the request and response messages will be formatted (per the SOAP protocol, in this case), how the messages will be transported over the internet (via HTTP, in this case), and where on the internet the specified operation will be available (at `https://medServices.org/classifier`, in this case).

2. *eXtensible Markup Language*: XML is a flexible and very widely used syntax for encoding structured data using ASCII characters. Data types from tables to hierarchical trees to images may all be encoded using XML in an entirely language- and platform-independent way. For example, the entire WSDL specification above is encoded in XML. Specific types of XML structures may be defined using the *XML Schema* meta-language. The simple schema appearing in the `<types>` section of the WSDL specification above is defined this way.

3. *Simple Object Access Protocol*: SOAP is a specific protocol for formatting web-services requests and responses when they are sent over the internet. In particular, SOAP defines a specific way of encoding as XML documents the operations, messages, parameters, and data types that make up a WSDL specification. SOAP typically uses the HTTP or HTTPS as the transport protocol (as is the case above). The following XML fragment shows how a request for the operation described above would be encoded as an XML message per the SOAP protocol:

   ```
   <soap:envelope>
      <soap:body>
         <isControlled>
            <medName>Vicodin<medName/>
         <isControlled>
      <soap:body>
   <soap:envelope>
   ```

   Interface specifications defined using WSDL may include any number of binding types, including raw XML over HTTP, SOAP over FTP, or SOAP over HTTP (as in the example above). When both an application and a server agree to use the same binding, however, they can receive each others messages and correctly interpret them with respect to the WSDL specification. In short, they're interoperable.

3. Accessing WSDL-defined Services from PHA Applications

As described above, WSDL, XML, and SOAP are sufficient to define in a precise and platform-independent manner the way in which two applications can interoperate over the internet. However, these technologies provide only *specifications*, not functioning software modules. As such, they do not provide any operational mechanism for actually building, sending, and processing the messages that make up web-services communications. Fortunately, the existence of these standards has enabled others to develop software tools that greatly simplify the task of building functional web-service interfaces in a variety of programming languages and on a variety of platforms. These tools essentially provide the "glue" between conventional software applications developed in Java, VB, C#, PHP, etc., and the web-services infrastructure defined by WSDL, XML, and SOAP.

The tools provide this glue by performing two important functions:

1. The tools generate source code in a variety of languages that implements "stubs" to call the operations defined in the WSDL file. These software stubs allow an application to request the web services using its native programming language. Specifically, the tools convert a WSDL specification into Classes with methods that may be called from the target programming language to request the web-service operations. For example, these tools would convert the "isControlled" operation defined in the WSDL above into the following Java method definition:

   ```
   Boolean isControlled(medName String)  {
        …some automatically generated source
        code that prepares a SOAP-encoded message…
      }
   ```

   Using this code, applications may invoke the defined web service simply by calling this generated method, such as in the following Java code:

   ```
   if (isControlled(myMedication)) {
           System.out.println("you cannot order " + myMedication + " online"); }
   else {
           submitOrder(myMedication); }
   ```

2. The tools also provide the runtime services necessary to send the web-services request over the appropriate transport protocol, and to receive and process the response messages. These runtime services are typically provided by software libraries that are part of integrated development environments (such as Visual Studio .NET or Eclipse Europa) or that can be downloaded and integrated into software projects (such as Java Community's JAX-RPC SDK or the NuSoap classes for PHP). Importantly, applications developed in different environments and using different code-generation tools and runtime libraries may all access the same web-services implementations, as long as they and the web service have all implemented the same WSDL specifications. This independence from specific platforms, languages, and tools is the strength of the WSDL + SOAP web-services model.

We propose the model above as the technical architecture for the Project HealthDesign common platform components. Specifically, we plan to define the functionality of and interfaces to the common platform components primarily as WSDL specifications, and to implement the components based on these specifications. We believe this model will allow the PHA prototypes, although developed in a variety of environments using a variety of languages, to all access the common platform components. We also believe that the available software tools and libraries for WSDL and SOAP will simplify the task of remotely interfacing to the CPCs from the native programming environments of the PHAs.