ON COMPOSING WEB SERVICES

ICS 243F Project Report

Sushil K Bajracharya   Orion Auld

{sbajrach,oauld}@ics.uci.edu

University of California, Irvine

School of Information and Computer Science

Abstract

This report describes a project we undertook for the course **ICS 243F:** *Middleware Networks and Distributed Systems* during Spring 2004. Our goal behind the project was to investigate the current state of developing applications based on web services and to look into the solutions that exist for composing disparate web services to build such applications. As a result we developed an application that uses web services to meet most of its functional requirements.

Instead of describing the rationale behind web services and other many tools that exist in the market, we discuss our experience in building an application based on web services and present some of our findings and observations regarding building the next generation of applications based on web services We describe the basic functionality and some technical details involved in our application, describe the web services we used and that we developed for the application and then present our analysis.

1 Elements of a first-generation web service application

First generation web services use synchronous RPC via SOAP [17] for communication and expose simple, straightforward services via WSDL [20]. Composition of services occurs manually, security occurs at the protocol level, and reliability measures are ad-hoc. This is the reality of web services at present.

Forays into the next generation are already well underway. .NET and JAXM [10] explore asynchronous, message-based, document-oriented communication. On the composition side, BPEL [3] and WSCI [18] describe how disparate services can be combined, and how they can coordinate to achieve a goal.

The dust from these efforts will take some time to settle. In the meantime, the world is refining its knowledge of the first generation of web services. We have learned a lot since the debut of core technologies such as SOAP. By now, creating a small web application that integrates a couple of established web services and deploys one of its own should be a breeze. And such a small application should not run into the known limitations of first-generation services, should it? This is what we set out to discover.

2 Overview of the application

The web application we built uses web services to locate wineries in California. With this application one can give a point of interest in California as a combination of Street Address, City and Zip code. The address need not be exactly accurate. Once this information is given the user is either presented with a list of matching locations to his/her criteria or is forwarded to another page if the given address uniquely maps to a valid location in California.

Once the application gets a valid starting point, the user then can select preferences for the wineries. Based on the preferences and the starting point the application generates a route consisting of all the wineries that match the criteria. The result is a set of stops in the route and a navigable map. From the result the user can also get driving directions along with relevant maps for the route generated.

2.1 Functional Decomposition

With the functionality described above we needed the following types of services -

Finding Accurate Locations List A user might not be aware of the exact location (s)he wants to start from. We needed a service that can take an incomplete description of a location and return exact/accurate locations that match the description.

Getting List of Wineries We needed a service that can return a list of all wineries that is around the vicinity of the starting point user is interested in. The user must be able to filter (her)his selection according to the different criteria (s)he wanted regarding the wineries to be visited.

Getting Wineries Tour Once we got an accurate starting point we needed to get a set of wineries around that starting location. This might further breakdown into -

- Getting all the wineries stops and wineries information that form a tour
- Getting a map for the tour that constitutes the wineries
- Navigating the map that highlights the tour with basic operations like Panning and Zooming.

Driving Directions Given a route made up of locations, we needed a set of driving directions to visit all the destinations in the tour.

2.2 Web Services mapped to Functionalities

Looking for the services we needed for our application we found few service providers offering such services like MapPoint WebServices [9] by Microsoft and ArcWeb [4] Services by ESRI. Both of the services were sufficient and complete for our purpose. But the area we wanted to looked into was composition of services so we divided the services we needed for our application into two, (i) External Services - those came from outside providers and (ii) Internal Services - that we decided to implement ourselves. For no particular reason, we chose MapPoint as our external service provider. Following sections describe the services we used in some detail.

3 Our Winery Route Planner web service

Since our application, *Winery Route Planner* is a vacation planning tool for those who are interested in wine tasting in California, given the fact that there are nearly a thousand wineries in California alone, it is not surprising that going wine tasting can be a hit-or-miss game. This is particularly if you're new to wine or wine tasting, or if you're interested in finding something new rather than visiting the same wineries you always visit.

Given a set of parameters that specify the characteristics of wineries that a user would like to visit and a starting address, the Winery Route Planner will return a set of wineries nearby which conforms to those characteristics and a route between them. The maps and detailed directions that it provides will be very useful for the designated driver or limousine driver and are clearly superior to a handwritten list of destinations.

3.1 The Winery Database

The `WineryFinder` database consists of two tables: `Destinations` and `WineryAttrs`. Destinations contains all of the properties of a tourist destination: its name, the various components of its address, a code representing the type of destination (of which only one, which represents a winery, is used), a short description, and destination's latitude and longitude.

The second table, `WineryAttrs`, comprises six different properties of a winery, rated on a scale from 1 to 10. Those properties are: `scenery`, `tour` (how good the tour at the winery is), `red_qual` (how good the red wines are), `white_qual` (how good the white wines are), `variety` (in the wines), and an overall `rating`.

The raw data which fills the Destinations table was aggregated from several publicly available sources. These sources were then processed, merged and inserted into the database. An external web service provided by `esri.com` was then used to geocode the addresses.

3.2 The WineryFinder Service

The WineryFinder service is a SOAP based web service. WineryFinder takes as its input a latitude-longitude pair signifying a starting location, a maximum travel distance, and five parameters specifying the relative importance of the (first) five winery attributes, namely, `scenery`, `tour`, `red_qual`, `white_qual` , and `variety`. It returns a list of Destination objects which represent Destination rows in the database. These objects are the best matches for the parameters as specified by the user within the maximum travel distance from the given starting point.

Underneath, the WineryFinder, after normalizing the parameters, makes a call to the database. This call takes the dot product of the vector of parameters and the scores of each winery in the database and produces a compatibility score. It returns the database entries within the supplied maximum travel distance that have the highest compatibility scores. Back in application space, these entries are marshalled into Destination objects along with their scores. These are returned to the caller.

4 The MapPoint WebServices

MapPoint WebServices is basically a SOAP wrapper for the giant MapPoint objet model (API) which has most of the features a typical mapping application requires. The functionalities MapPoint provides is grouped into three basic services -

FindService used for locating an exact address given a partial description of an address.

RouteService used to create a route from a given set of locations, also gives driving directions.

RenderService used to create Maps for routes and supports operations on maps like panning and zooming.

MapPoint also supports creating custom point of interests in their servers that is particularly useful when we want to create points of interests like locations mapped to wineries as in our case. In summary MapPoint itself was sufficient to meet our requirements. To experiment with composition we decided to introduce our own service that will be replacing some of MapPoint's features. We came up with following distribution of tasks to services as shown in Table 1.

The consequence of plugging in our own service to fit into what is already envisioned in MapPoint is the overhead of mapping our object structure with that of MapPoint. This is what got inside the *glue* code we wrote for composing the services.

Task	Services	Providers	Method Signatures [1]
Finding set of exact locations	FindService	MapPoint	`FindResults findAddress (FindAddressSpecification)`
Getting wineries matching criteria	WineryFinder	local service we developed	`Destination[] getLocationsByScore(double, double, double, double, double, double, double, double)`
Generating route from the tour given set of destinations	RouteService	MapPoint	`Route calculateSimpleRoute (ArrayOfLatLong, dataSourceName, preference)`
Getting a map representing a route/tour. Also, navigating the map	RenderService	MapPoint	`ArrayOfMapImage getMap(MapSpecification)`
Getting driving directions	RouteService	MapPoint	can be obtained from the `Route` object

[1] showing only the most relevant methods in format - `return_type web_service_function_name (input_parameter type)`. The types shown in the list represent the classes in Java that maps to the types defined in the MapPoint object model. These classes were auto-generated by WSDL2Java.

Table 1: Mapping tasks to services

4.1 The web application - where services meet

Our approach so far have been statically composing the services together and most of the work revealed out to be what we used to do in any object oriented development. As a web services toolkit we used Apache AXIS [6], a SOAP engine with support for WSDL and tools for some automation. With AXIS, All the intricacies of marshalling/unmarshalling and remoting is gone, handled partly by the proxy code automatically generated by tools available with AXIS and rest by the AXIS core engine itself. What remains to be worked on is mundane object mapping task, converting data from one object type to another, that is from one service to other. This is what lies behind the static composition.

As an integration layer we developed a JavaBean that can live on any servlet complaint web server. The bean also is a controller of the web application, handling requests from webpages (implemented using Java Server Pages), managing the application's workflow and talking back and forth with SOAP proxy classes generated by `WSDL2Java` [6]. These proxy classes talk with web services as required. The bean also handles authentication required to use MapPoint web services. The bean is a stateful entity that manages the life cycles of various objects representing MapPoint's and our WineryFinder services.

5 Future extensions to the application

The vision of *Winery Route Planner* is somewhat grander than its humble origins. Naturally, we imagine an application which is an information resource on wine, wineries, and wine touring in California. Beyond this we see opportunities to increase route planning interactivity by an order of magnitude:

- Once users receive a route, they should be able to click through to detailed descriptions of the wineries. If for some reason one does not appeal, WineryRoutePlanner should pick another one for them and adjust the route.

- In a similar fashion, users should be able to mark wineries that they have already visited or for some other reason have no interest in visiting. The route planner should automatically find other wineries to take their place, giving them a new route. These preferences should be persistent; the next time a user wants to create a route, those wineries will not be considered.

- Other types of destinations should be considered, eg. museums and art galleries. Possible lunch or dinner destinations should be presented to the user; they can select from among them and WineryRoutePlanner should automatically integrate these destinations into their route.

  These possibilities for extensions add further complexity in developing and managing applications based on web services. Solutions do exist that allows to manage [2] and monitor services [7] and configure them, but the question is how easy it would be to switch from one provider to another without having ripple effects, what solutions can exist so that we can pick and choose only those services we want in our application from a big set of services, how do we compare among available service providers, how do we ensure reliability when our applications depend on services provided by others and so on.

The community is still experimenting with solutions to these problems, directories for service providers like UDDI [14], approaches to standardization [11] with technologies like XLANG [13] and BPML [1], and, research projects trying to incorporate techniques like Aspect Oriented Programming [15] and redundant components [8] - all these are trying to answer questions we had put earlier.

6 Curing the ills of first-generation web services

6.1 The trials of composition

In building our application, we started out with a sample application that has a similar structure to our application and is developed by *SpatialPoint* [12] as a demo application for MapPoint. Despite the similarities, there was some effort from our side to reverse engineer and port the sample to fit our requirements.

In principle we almost had everything we needed, the services, bare-bones sample code to start with and a toolkit to access those services. But what we realized is, the service we were dealing with were interfaces to huge monstrous code that needs to be put together. Commitment to a service like MapPoint is a commitment towards the design invariants it has setup and trying to invade those invariants by introducing our new services to replace some of its facilities is not an easy development effort.

Other problems we faced during the development was of adopting a new tool from the open source community. It is not that we have not worked with open source tools but one of the areas open source projects lag behind is in documentation, especially related to configuration changes with every new update and release. So usually the starting curve with open source tools is comparatively higher. In our project, configurations regarding JNDI and database pooling had some problems due to which we had to implement our local and simple database connection mechanism. But the AXIS core worked seamlessly with Tomcat, the server we chose for hosting our application.

The amount of glue code required to compose services together is not significantly high compared to the amount of code required to arrange the sequencing of messages between different objects and maintaining states for different user parameters in those objects, for instance, users preferences for wineries or users current panning/zooming choice for map. These overheads will still be there even if we stick to one particular provider for services we want.

The major industry players are tackling these many of these issues by creating high level languages and environments to compose, coordinate, and manage web services. Languages like BPEL have been created to manage web service applications as a combination of workflow activities, enterprise integrations, and executable business models. Using BPEL, we could easily define a process which could access a group of identical services in a round-robin fashion, or shuffle the output of one service to another.

Still, our experience raises some doubts about this vision. In the course of developing this simple first-generation model, we had many different needs that we could not find tools to meet. For instance, we wanted to define a simple WSDL that was a subset of MapPoint's WSDL, defining only the operations that we needed, and that was a superset of another API, and then develop proxy objects that would allow us to make calls to either one, allowing for some redundancy. The amount of work involved for such a conceptually simple task was actually quite formidable, even if the tasks themselves were not difficult. If, this late in the game of first-generation web services, such a task involves so much manual labor, what hope is there for workable second-generation environments anytime soon?

6.2 The travails of data management

In a way, the WineryFinder service is really just a way encapsulating a set of queries on a database of some sort. It is clear that a better way should exist than exposing an RPC and returning a set of objects defined by the remote host, which is how a first-generation web service would look at the problem. Each kind of query requires a new kind of RPC, which requires new integration effort on the client's part, and this implies a long request-change cycle if a client requires a type of query that the server has not implemented yet. A client's only alternative is to, if possible, perform an overbroad query and perform processing on the client side, an option which is potentially very wasteful.

It is in this situation that the advantages of a document-centric, REST-type [5] architecture are quite clear. Documents can be (selectively) cached, and existing technologies (such as gateways, proxy servers, etc., can be placed between the client and the server to meet various performance/security goals. Furthermore, in a case like this, document-oriented techniques are just simpler. As far as supporting arbitrary queries, some have explored mapping declarative query languages such as SQL onto REST architectures. Perhaps a more natural solution would be provided by XQuery [21]. Finding a way to make it mesh in a REST context without abusing POST is another matter entirely, as is providing fine-grained security of those data sources.

6.3 The agony of paradigm shift

In conclusion we can say that working with a first generation web services application has not been as exciting as we had conceived. After all the efforts we put in, we just felt like we were doing the same kind of development that would have been in the case of primitive object oriented RPC style application. The difference was our application used SOAP and XML (which was entirely taken care of by AXIS itself) instead of some RPC protocol, serialized objects or IDL.

We close our discussion by analyzing the following excerpt on the goals of web services and its role in building the next generation of distributed applications -

> The power of Web services, in addition to their great interoperability and extensibility thanks to the use of XML, is that they can then be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.
>
> In order to make this extensible, loosely coupled, interoperable distributed computing environment a reality, the architecture of Web services needs to be better understood, and several new technologies need to be developed.

So far much of the interoperability has been achieved due to the use of XML, but with just this achievement its not yet time for any celebrations. A pessimistic view on this might be to see it as just an extra overhead by building yet another layer of messaging and protocols on top of what already exists to connect differing products from different vendors. But we definitely are not pessimistic about the vision web services and service oriented architecture holds for the future of distributed computing and the trend of the software development itself.

However, what is necessary at this point is to ponder upon the way we design our application and services. If the services are just interface to a cohesive block of software components dictated by their design invariants, users must commit to a particular vendor and the modularity does not bring much value due to the choices available and the notion of *simple services loosely coupled for sophisticated services* will be still far from possible. A point of further interest might be to look into the solutions that technologies like *ontologies* [16], *choreography description languages* [19] and *the semantic web* [16] are trying to offer. Our experience with the cognitive effort we had to apply even to build a small application like we built still makes us skeptic about the immediate success of such solutions. What is interesting is the window of opportunity that the problems bring in order to rethink about the design and implementation of future generation of distributed applications.

References

[1] BPMI.org. (BPML) - business process modeling language. *http://www.bpmi.org/bpml.esp*.

[2] Hewlett-Packard Development Company. Web services management framework. *http://devresource.hp.com/drc/ specifications/wsmf/index.jsp*.

[3] IBM Corporation. Business process execution language for web services. *http://www-106.ibm.com/developerworks/library/ws-bpel/*.

[4] ESRI. Arcweb services. *http://www.esri.com/software/arcwebservices/index.html*.

[5] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22nd international conference on Software engineering*, pages 407–416. ACM Press, 2000.

[6] The Apache Foundation. Apache AXIS. *http://ws.apache.org/axis/*.

[7] Mindreef Inc. Mindreef soapscope. *http://www.mindreef.com/*.

[8] Chang Liu and Debra J. Richardson. Research directions in raic. *SIGSOFT Softw. Eng. Notes*, 27(3):43–46, 2002.

[9] MapPoint/Microsoft. Mappoint webservices. *http://www.microsoft.com/ mappoint/webservice/default.mspx*.

[10] Sun Microsystems. Java api for xml messaging (JAXM). *http://java.sun.com/xml/jaxm/index.jsp*.

[11] David O'Riordan. Business process standards for web services. *http://www.webservicesarchitect.com/content/articles/BPSFWSBDO.pdf*.

[12] SpatialPoint. Spatialpoint web site. *http://www.spatialpoint.com/*.

[13] Satish Thatte. (XLANG) - web services for business process design. *http://www.gotdotnet.com/team/ xml_wsspecs/xlangc/default.htm*.

[14] UDDI.org. UDDI universal description discovery and integration of web services. *http://www.uddi.org/*.

[15] Bart Verheecke, María A. Cibrán, and Viviane Jonckers. AOP for Dynamic Configuration and Management of Web Services. pages 137–151.

[16] W3C. Semantic web activity statement. *http://www.w3.org/2001/sw/Activity*.

[17] W3C. Simple object access protocol (SOAP). *http://www.w3.org/TR/soap/*.

[18] W3C. Web service choreography interface (WSCI), journal = http://www.w3.org/TR/wsci/.

[19] W3C. Web services activity statement. *The World Wide Web Consortium Web Site*.

[20] W3C. Web services description language (WSDL). *http://www.w3.org/TR/wsdl*.

[21] W3C. Xquery: An xml query language. *http://www.w3.org/TR/xquery/*.

7 Appendix: Project roles/responsibilities

Gvien below is the summary of our individual task distribution in our project -

- Sushil K Bajracharya
  - MapPoint and Web Services integration
  - Web interface and application development (Glue code for composition)
- Orion Auld
  - WineryFinder web service development
  - Data acquisition and processing
  - Database development, JNDI, JDBC configuration