Integrating Web Services into Map Image Applications

Shengru Tu
Eric Normand
Sriram Kuchimanchi
Vianney Bizot
Shujing Shu
Mahdi Abdelguerfi
Computer Science Department
University of New Orleans
New Orleans, LA 70148
[shengru, enormand, skuchima, bizot, sshu, mahdi]@cs.uno.edu

Jay Ratcliff
U.S. Army Corps of Engineer
New Orleans, LA 70118
Jay.Ratcliff@ mvn02.usace.army.mil

Kevin Shaw[*]
Naval Research Laboratory
Stennis Space Center, Mississippi
shaw@nrlssc.navy.mil

Abstract

Web services have been opening a wide avenue for software integration. In this paper, we have reported our experiments with three applications that are built by utilizing and providing web services for Geographic Information Systems (GIS). The services are designed to handle a large number of concurrent requests. It is clear that performance has to be the central consideration in design of GIS web services. The lessons learned from these experiments include the application of the rich metadata message approach, choosing large size of unstructured data but limiting the structured message's sizes, and minimizing COTS software customization.

1. Introduction

Web services have been gaining a strong momentum as a platform upon which to develop applications that take advantage of the Internet infrastructure. Web services are the self-contained, web-enabled applications capable not only of performing business activities on their own but also engaging other web services in order to complete higher-order business transactions. Web services are based on three specifications, namely SOAP [9], WSDL [2], and UDDI [1], which provide an XML-based basic framework for application interoperability, service description, and service discovery. These standards have helped web services to evolve into a means to integrate processes and applications at an inter-enterprise level.

In this paper, we report our experiments with three applications that are built with Web services and at the same time provide Web services for Geographic Information Systems (GISs). There are at least three characteristics of GIS services that make it difficult to design GIS web services with satisfactory performance. First, services provided by a GIS typically require heavy CPU usage due to the complex computation involved in the underlying computational geometry. Second, GIS services often transmit large resulting data sets such as images. Third, the "clients" of GIS web services are often complex software tools such as the CAD desktop applications. The services are designed for a large number of concurrent requests. It is clear that simply establishing communications between components is not sufficient for scalable GIS web services; performance has to be the central consideration in design of GIS systems. In addition, the design should not only consider the server side but also the client side. As an eminent example of heavy-duty web services, the lessons we learned from the GIS applications (as well as a human resource management system) are valid to general enterprise software integration using web services.

The remaining part of this paper is organized as the following. Sections 2 briefly provides the background about GIS and the web service technology. Section 3 surveys the related works. Section 4 presents three GIS related web applications that utilize web services. The lessons learned from these three projects are summarized in Sections 5. Finally, Section 6 concludes.

[*] The work of Kevin Shaw was partially funded by the National Guard Bureau with program manager Major Mike Thomas.

2. Background

2.1 Web services

The XML-based Web services have become one of the most popular technologies in the computing industry. Its underlying protocol is SOAP, a lightweight XML protocol for structured information exchange across a distributed environment. The World Wide Web Consortium (W3C) has been successfully steering SOAP's evolution from an HTTP-based RPC mechanism in XML to a leading interoperable technology with replaceable bindings [9]. Design of web services systems have received attention from industry [3], academia [4] and government [8, 10].

2.2 Geographic Information Systems (GIS)

GIS refers to computer systems capable of assembling, storing, manipulating, and displaying geographically referenced information, the data identified according to their locations. By storing all kinds of correlation data into a GIS database according to layers and locations, we can easily query, update and insert information from many different sources in many different forms according to the different applications.

2.3 Web map services and Web feature services

Parallel to the evolution of the web service technology, the Open GIS Consortium (OGC) has been pursuing web map services. The current Web mapping standards including, the Web Map Service (WMS) and the Web Feature Service (WFS) Implementation Specifications [5]. These specifications have considered not only access to simple features but also data with temporal information, as well as transactions of feature manipulation. Leading GIS software vendors such as ESRI and Intergraph quickly implemented WMS and WFS services in their products. Since OGC's WMS was formalized before SOAP emerged, WMS and WFS do not refer to SOAP.

3. Related Works

Leading vendors have also been implementing SOAP-based GIS web services. For example, ESRI's ArcWeb Services have demonstrated a commercially hosted web service site that sells spatial data and GIS functionality. It has illustrated the possibility for users to avoid hosting GIS data sets but to rely on commercial GIS web services.

Microsoft TerraServer Web Service, also called TerraService (terraserver.microsoft.com), is an example of map imagery Web services. TerraService has been demonstrating a programmable interface to an on-line database of high resolution USGS aerial imagery and topographical maps. It is well-documented and free. TerraService's underlying storage scheme is a database approach with a hierarchical tiling technique [7]. TerraService created 200 by 200 pixel, compressed image "tiles" that are aligned to specific coordinates on the globe. Then, an image pyramid is pre-computed at predictable image resolutions. This approach makes it possible for the users to navigate and browse the vast amount of imagery (3.0 Terabytes of compressed image tiles created from 12 Terabytes of uncompressed image data files) over slow speed lines with a standard web browser without special software. TerraService's services can be broken down into three categories: the *search* methods for finding geographic and image coordinates by place name, the *projection* methods for converting coordinates from one projection system to another, and the *tile* methods for fetching tile meta-data and image-data.

4. Applications Based on Integration of Web Services

In this section, we report three projects that utilize web services for GIS applications. Even though the requirements from the three clients are vastly different, they all involve map images. Our design strategies for them are similar.

4.1 Overlay insect trap tapes location on aerial images

There has been substantial research on the behavior of insects that are harmful to agriculture, which theoretically can help farmers to spray insecticide intelligently. To realize the benefit, farmers need to monitor the density of certain insects in the fields. An effective and affordable way to do so is to place bug-trap tapes and count the bugs that are stuck to it. A company specializing in bug-trap tapes also wants to provide intelligent insect control service based on a number of prediction models. At the very beginning, they need to overlay the insect distribution over the field images. This helps farmers visualize the distribution of insects from the viewpoint of the infected area.

The company has developed software on PDA for farmers to collect tape readings and send the data to a small database. However, after we imported the data as features into their GIS database (an ArcSDE installation), we found that the company was not ready to acquire all the map images covering the customers' fields nationwide. A legitimate concern was about the necessary investment for upgrading hardware. As a temporary means, we decided to use the DOQ (Digital Orthophoto Quadrangles) map images from Microsoft's TerraService as described in Section 3. The architecture for the insect distribution Internet service is shown in Figure 1.

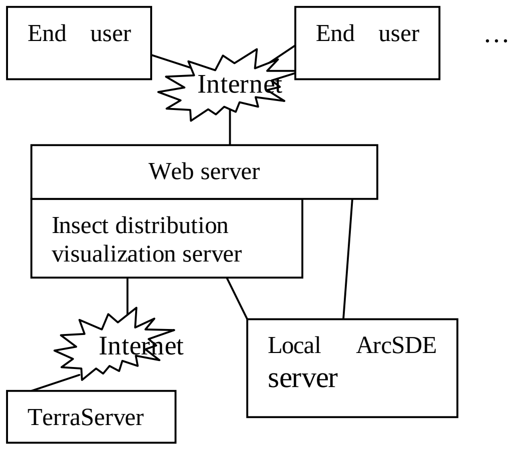

Fig. 1 The Insect Distribution Visualization Server

The Insect Distribution Visualization Server (IDVS) resides in the same computer that runs the web server. IDVS is a client of both the TerraServer and the local ArcSDE server. Upon a user's request, IDVS obtains the feature data (the bug-trap tapes' readings and their locations) from the ArcSDE server. According to the bounding box of the selected tapes' locations, IDVS determines the geographic bounding box of the background map image as well as the desired map resolution. With this information, IDVS queries TerraService to find out which image tiles are needed to cover the background because TerraService delivers images as fixed-sized tiles. This is done by calling TerraService GetAreaFromPt.

```
public AreaBoundingBox GetAreaFromPt
    (LonLatPt center, Theme theme, Scale scale,
    int displayPixWidth, int displayPixHeight)
```

The LonLatPt point parameter identifies the Geographic center of the rectangle of interest. The Theme and Scale parameters identify the type of imagery and resolution of interest. The displayPixWidth and displayPixHeight parameters identify the needed image size.

The *GetAreaFromPt* service returns an object of AreaBoundingBox that contains the identifiers (TileId) of the four titles that cover the four corners (NorthWest, NorthEast, SouthWest, SouthEast) of the bounding box, as well as the pixel locations of a specific longitude and latitude value (LonLatPtOffset) within each of the four corner tiles. By choosing the values of Scale, displayPixWidth and displayPixHeight properly, we can ensure that the number of required tiles never exceeds nine. Once IDVS determines the set of tiles to fetch, it calls the *GetTile* service repeatedly to get all the images. *GetTile* is the only service provided by TerraService that actually delivers map images.

```
public Byte[ ] GetTile(TileId id)
```

The *GetTile* service returns a Byte array containing the compressed image data for the requested tile. Note, *GetTile* returns only one tile upon each call.

IDVS then draws the feature points onto the background image. Finally, the entire image is sent to the end user. By maintaining a high-speed Internet connection for IDVS, IDVS could always complete a resulting image within four seconds in our tests. The surprisingly good result has made us change our mind about hosting the company's own image services. The cost of using a paid map image service will be offset by the saved hardware/software investment as well as the maintenance cost.

The TerraService provided by Microsoft's TerraServer (www.terraserver-usa.com) has exemplified excellent design. The rich (coarse-grained) metadata approach helps clients reduce the number of requests effectively. For instance, the AreaBoundingBox object returned by the *GetAreaFromPt* service virtually satisfies all the conceivable needs for information of image maps including possible image cropping. By assigning each data object (tile) a unique identifier (TileId) with application-level meaning (row and column), the data requests based on the metadata provided by the metadata services (such as *GetAreaFromPt*) are completely independent from the previous metadata request. With such an arrangement, the server (TerraServer) only needs to handle stateless requests, which is a key to scalability.

4.2 A fly-through simulation using images from web services

As good as it is, the TerraService is suitable for web browsing in which few tiles will be needed at a time. However, when a GIS application requires many image tiles at the same time, the large number of calls to GetTile will cause linearly increasing communication time cost when a high power user wants many tiles, since GetTile delivers data one tile at a time. This was confirmed by a simple test on accessing time to fetch tiles from TerraService. The test results have been summarized the table in Figure 2. We propose an additional service that returns an assembled image with multiple tiles.

# of tiles	secs
2	1.2
3	1.5
4	2.0
5	2.3
10	3.6
20	8.2
30	14.3
40	18.3
50	24.3
60	28.4
100	50.4

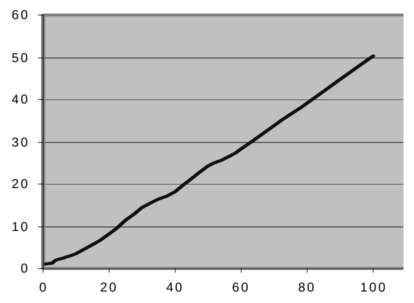

Figure 2 Accessing time to TerraService

public Byte[ ] GetImageFromPt (LonLatPt center,
Theme theme, Scale scale, int displayPixWidth,
int displayPixHeight)

The parameter list of this suggested service is the same as that of the *GetAreaFromPt* service. The difference lies in the returning value. Rather than returning the rich metadata of the boundingbox, the *GetImageFromPt* service returns a large byte array that holds the image assembled from all the tiles surrounded by the four corner tiles returned by the *GetAreaFromPt* service. In this way, the delivery will cost one connection time only. If the client needs to accurately crop the image, a call to the *GetAreaFromPt* service will provide all the needed information. This suggestion should improve the transmission time for users who request large images.

To realize our suggestion regarding performance improvement, we implemented a set of web services that provide map images in multiple-tile groups using a local ArcSDE map server. Figure 3 illustrates the GUI of experimental client, a fly-through simulation. The flying speed, map resolution (plane's height), flying direction, and the display size can all be set by the user. Setting to high resolution with a large display size will require many tiles per row. To achieve a smooth flying view, we programmed a buffer on the client side. Choosing a high speed will further require fetching as many titles as possible. Using our home-made web services, the simulator fetches tiles row by row by calling GetImageFromPt and achieves satisfactory results. A nearly identical simulator could not be carried out when the chosen display size and the resolution required five tiles per row. To make our comparison between our image server and the remote TerraService, we used a network simulation package that mimicked the Internet delay in the communication channel between the local server and the client. Still, the GetImageFromPt service of our server supported a satisfactory simulation on the client side.

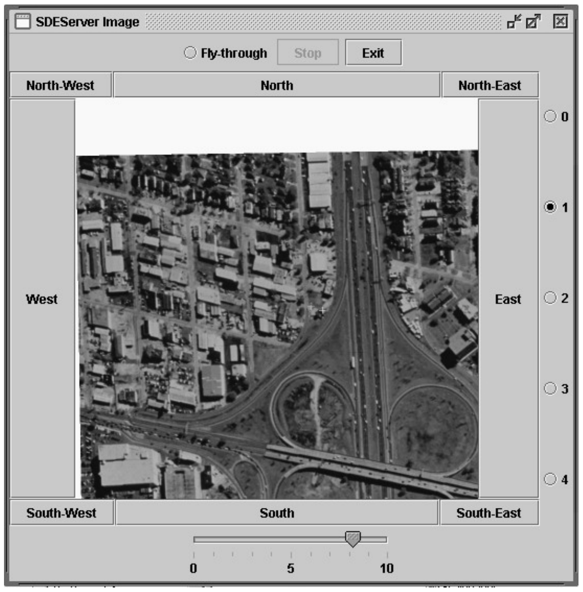

Figure 3 The GUI of the Fly-Through Simulator

4.3 Marking addresses from HR software on maps

The geographic information adheres to nearly every physical object. As the software systems are integrated into more and more inclusive cycles, geophysical information naturally becomes the pivot point of integration. In recent years, homeland security tasks have urgently needed comprehensive and real-time information resources. Building new systems for these tasks would be costly. Integrating existing systems can deliver economical solutions. For example, to effectively stop spreading a disease from a known location, the public health agents need to immediately identify the potentially contaminated areas. Because each individual system contains its own information separately, the fragmented information cannot be utilized without integration. Considering that the municipal business bureau's database has the names of the companies around the contaminated area; the personnel information system of each of the companies has the address of every employee; and the general public GIS has the map to mark the potentially contaminated houses. In principle, web services should fit into this kind of integration well.

As a feasibility study of a project that requires the inter-enterprise integration as described above, we carried out an experiment on integrating the GIS map imagery web services (TerraService) into the PeopleSoft Human Resource (HR) system. Our goal was to integrate the map imagery capability into the PeopleSoft HR system. By clicking on a PeopleSoft HR page showing an employee's information, we want the PeopleSoft page to trigger the display of an image map showing the employee's residence. This function is needed during an emergency situation for the homeland security agent. To achieve this, we need to do two things: (1) get the address information out from PeopleSoft and find its geocode; (2) fetch images from a map image service such as TerraService and display them. In Sections 4.1, we have shown how to fetch and display map image tiles surrounding a specific location identified by its longitude and latitude. Thus, we will focus on Step (1).

To describe this process, we need a little knowledge about PeopleSoft HR, the most popular Commercial Off-The-Shelf (COTS) Human Resource management software in governmental organizations in the U.S. PeopleSoft uses a large database to store data and a set of comprehensive web-based interfaces to access the data. A user can access much of the information about employees through web forms. Developers can add or modify functionality to customize the system for their needs using the Application Designer.

One of our design principles was to minimize the modification to the PeopleSoft applications. This is a way to make our integration sustainable through PeopleSoft software upgrades. For instance, we decided to send the whole Rowset XML document (rowset) generated by PeopleSoft with no modifications. The consequences of this decision are explained below.

As a first attempt, we decided to send a message to an Address Storage web service we created which stores name and address information. We configured the PeopleSoft applications to send NAME_AND_ ADDRESS_MESSAGE (a message type we created) to the Address Storage web service. Next, we had to actually send the message. To do that, we added code to a standard PeopleSoft event called *SavePostChange*. This event is called whenever a user clicks the *Save* button on a PeopleSoft screen with new data. To achieve this, we found a built-in page in PeopleSoft HR that contained address information (it is called PERSONAL_DATA). We added the event handling code (event handler) to this page. This event handler received the rowset for PERSONAL_DATA, then converted it into XML and inserted it into a SOAP message destined for the SOAP service we set up. It was a simple operation that took 15 lines of PeopleCode. The advantage of this method is the additional code of the event handler could have been inserted into any *SavePostChange* event for any PeopleSoft component; it would have worked the same way. A severe drawback of this method was the XML message generated for the generic Rowset was excessively verbose. The XML message was almost two megabytes in length for each person's address. Most of the XML document was unneeded.

Another solution was to add People code to the PeopleSoft event handler and let it extract the data from the rowset. Fortunately, PeopleCode had XPath capabilities (XPath is a way to represent a path to an XML node in an XML document), so with a few more lines of code we reduced the size of the message dramatically while still retaining the needed data. This eliminated most of the unused portion of the XML message, and dramatically reduced the size of the message. The drawback to this working solution was that we lost code reusability. We had to write different event handler for different data sets. Such a case-by-case customization would make the integration code less and less manageable.

Our final solution has regained high code reusability. We let the event handler query the whole rowset as it is generated by the built-in PeopleSoft page, PERSONAL_DATA. Instead of sending the rowset to the Address Storage web service, the event handler code writes the rowset to a file that is accessible by the web server. Then, the event handler sends the URL of the file to the Address Storage service. The Address Storage service correspondingly does an HTTP GET on the URL, and obtains the complete XML file. It extracts the name and address information from the XML file, and converts the address into a latitude and longitude by looking it up from the TerraService. Then the Address Storage service stores the name and coordinates in an XML file. The main advantage of this approach is that we can always use a uniform event handler for any rowset; no message specific code is needed.

5. Lessons Learned

In this section, we summarize the lessons that we have learned from the above three projects. In the first project, we particularly appreciate the rich metadata operation approach, represented by the GetAreaFromPt operation that returns an object AreaBoundingBox. This is a reflection of the well-known CORBA design pattern called “secondary object identities” [6]. The essential idea of this design pattern is to use a data structure to enclose a combination of the object identifiers along with adequate information to support the user’s common queries. Applying this pattern to web services turns out to be very effective in reducing remote requests. It is also convenient because the underlying XML is ideal for packing structured data.

A lesson learned in the fly-through simulation project regards the coarseness of messages. For unstructured data such as compressed images, the larger message size usually leads to more efficient transmission. On the other hand, a design choice has to be made carefully if a larger message will require the server to carry out more computation.

For structured data, the message size is subject to more constraints. This is because the size of the XML documents will have impact on the parsers in the message channel. For instance, in the PeopleSoft-GIS integration project, we encountered multiple-megabyte SOAP messages. Many systems use the DOM model to reconstruct the object in the memory, which lead to very heavy memory usage. When many concurrent requests involving large messages hit the SOAP messaging channel, the server can be overwhelmed.

A very important consideration regarding integration with COTS software is to assure the manageability of COTS software upgrades for long term. The COTS vendors upgrade their COTS products regularly, to enhance features and performance. Facilitating clients’ data into a new version is typically an obligation of COTS vendors. However, vendors do not support the transformation of clients’ customization code. Thus, massive or complex code customization in COTS deployment will hinder future COTS upgrades. Our strategy was to rely on the most fundamental functions (*SavePostChange*) and stable generic data structure (rowset) as explained in Section 4.3. In order to enable integration, adding code to the COTS software is inevitable. By using a uniformed add-on piece for creating any outbound message from PeopleSoft, we greatly reduced the complexity of integration. This way will help us achieve better manageability.

6. Conclusion

The SOAP-based web services have been opening a wide avenue for software integration, especially for COTS software integration. Through reporting three GIS related web applications, we have shared our experiences regarding the rich metadata message approach, as well as the choice of message size. In determining message sizes, we should differentiate the structured from unstructured messages. Following the principle of minimizing customization inside COTS, we found a uniform add-on piece for creating any outbound message from the COTS. We hope that our paper can garner sufficient attention from web service designers to take note of the design for web services that are involved in intensive computation or enormous data transmission.

7. References

[1] T. Bellwood, et al., “UDDI Spec Technical Committee Specification”, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 2002.

[2] R. Chinnici, et al., “Web Services Description Language (WSDL) Version 1.2”, http://www.w3.org/TR/wsdl12/

[3] P. Fremantle, S. Weerawarana, and R. Khalaf, “Enterprise Services”, CACM, 45:10, 77-82, Oct. 2002.

[4] K. Larsen and P. Bloniarz, “A Cost and Performance Model for Web Service Investment”, CACM, 43:2, 109-116, Feb. 2002.

[5] Open GIS Implementation Specifications, http://www.opengis.org/techno/implementation.htm

[6] D. Slama, J. Garbis, and P. Russell, *Enterprise CORBA*, Prentice Hall, Upper Saddle River, NJ, 1999.

[7] S. Tu, X. Li, X. He, and J. Ratcliff, “A Systematic Approach to Reduction of User-Perceived Response Time for GIS Web Services”, Proceedings ACM GIS 2001, Atlanta, GA, pp 47-52.

[8] R. Wilson, M. Cobb, F. McCreedy, R. Ladner, D. Olivier, T. Lovitt, K. Shaw, F. Petry, M. Abdelquerfi, “Geographical Data Interchange Using XML-Enabled Technology within the GIDB(TM) System”, invited in edited manuscript: A B. Chaudhri (ed), *XML Data Management*, John Wiley & Sons, 2003.

[9] World Wide Web Consortium, “SOAP Version 1.2 Part 1: Messaging Framework”, http://www.w3.org/TR/2003/PR-soap12-part1-20030507/

[10] Digital Mapping, Charting and Geodesy Analysis Program (DMAP) Team, http://dmap.nrlssc.navy.mil.