Thin Client Visualization

Stephen G. Eick, M. Andrew Eick, Jesse Fugitt, Brian Horst, Maxim Khailo, Russell A. Lankenau

SSS Research, Inc
600 S. Washington, Suite 100
Naperville, IL 60540
eick@sss-research.com

Abstract—We have developed a Web 2.0 thin client visualization framework called GeoBoost™. Our framework focuses on geospatial visualization and using Scalable Vector Graphics (SVG), AJAX, RSS and GeoRSS we have built a complete thin client component set. Our component set provides a rich user experience that is completely browser based. It includes maps, standard business charts, graphs, and time-oriented components. The components are live, interactive, linked, and support real time collaboration.

TABLE OF CONTENTS

1. INTRODUCTION

GeoBoost™ is a thin client collaborative visualization framework for building Web 2.0 browser-based geospatial applications. GeoBoost runs in both J2EE and .NET environments and leverages new programming techniques such as AJAX (Asynchronous JavaScript and XML) and browser-based graphics standards such SVG (Scalable Vector Graphics) to enable developers to build rich, interactive thin-client geospatial applications that support collaboration. The advantage of this approach to building applications is that thin client applications run natively within a browser without the need to install any client software, applets, or plug-ins. This advantage translates into low-cost deployment to many users and the ability to support many devices for real-time operational decision-making.

One of the surprising aspects of our GeoBoost framework is its performance. GeoBoost visual components support interaction, tooltips, dynamic queries, filtering, selection, panning, and zooming. The interactive performance GeoBoost obtains as thin client is comparable to some of the best implementations of desktop visualization software.

The GeoBoost system consists of three components. First, GeoBoost visualization components are built using a JavaScript Visualization Framework that we call *thinc* Interface™. *thinc* Interface is a thin client AJAX visualization framework we built that uses Scalable Vector Graphics as its rending API. *thinc* Interface ingests live RSS and RSS extensions and renders the information in browser-based visual components. Using *thinc* Interface we have built a set of a set of reusable visual components including:

- Interactive Maps that display image, feature, and live GeoRSS data from multiple sources
- Business Charts including Bar, Pie, and Line Charts
- Network and Graph Displays
- Hierarchical, Tree, and Tree Map Displays
- Time Lines and Time Wheel

These components are interesting because they have the capability to connect to live RSS streams, ingest the real-time data, and display the resulting information. The end-user programming model for GeoBoost visual components is also interesting and involves JSP custom tags and .NET Web controls. As a result it is easy for programmers to include GeoBoost components in their web applications using standard HTML tags.

The GeoBoost map component is particularly interesting. It is an OGC® (Open Geospatial Consortium) WMS (Web Mapping Service) and WFS (Web Feature Server) - compatible client that supports rich interactivity such as panning, scrolling, and zooming. For dynamic interactive operations raster data is asynchronously requested from WMS servers using AJAX requests. Feature data is rendered on top of the raster data using SVG layers. It is also a collaborative geospatial wiki. Users may mark, annotate, edit, and draw arbitrary shapes onto the map. The polygons corresponding to these shapes and annotations can be propagated to other connected users to allow collaboration over the marked area of interest.

Secondly, GeoBoost provides a series of servers to stream imagery, feature data, and RSS for consumption by GeoBoost visual components. Thin clients have limited

processing power and systems built around thin client architectures need more server support than equivalent desktop systems. Thus GeoBoost server components enable flexible data ingestion, image and feature data tiling, tile caching, performance optimization, and provide built-in support for tracking and collaboration features. These server components are necessary to support thin client interfaces and applications.

GeoBoost uses an open extensible architecture and supports relevant standards. GeoBoost is built around open standards such as GML (Geography Markup Language), RSS, GeoRSS and other RSS extensions. GeoBoost consumes imagery and feature data from both OGC compliant sources such as WMS (Web Mapping Service) and other native formats such as NGA's [3] RPF and LizardTech's [4] MrSID.

The third interesting component of GeoBoost is the GeoBoost AJAX portal. This portal provides a live framework for content, e.g. web pages, to be integrated in, customized, and tailored to specific use cases. Users with different job functions may customize GeoBoost content for their specific needs and thereby incorporate task-specific workflow.

The benefits of the GeoBoost platform include:

- Provides the user with rich desktop user interface functionality in a browser
- Includes visual components that are ready for use and may be incorporated into web applications
- Supports real-time collaboration without proprietary client software
- Enables inexpensive and large deployments since no client software is required to be installed
- Provides web developers with a platform for rapidly building geospatial and thin client applications
- Supports the relevant open standards including OGC WMS, WFS, GML, RSS, GeoRSS

2. TECHNOLOGY BACKGROUND

Applications built using the GeoBoost™ platform are described as Rich Internet Applications [5] (RIAs). Although Macromedia coined the term, a RIA today is understood to be an application that runs in a web browser and provides an interactive and responsive user interface that has traditionally been found in desktop applications. Until recently, the only software applications that could provide this level of responsiveness desired were the standard "fat client" applications that require client-side installation and run as a native client applications. Recently, a new class of RIAs has emerged that run completely within a web browser. These applications offer many benefits over traditional fat client applications. In this section we will describe some of these benefits and discuss other attempts to provide this functionality.

Fat Client Application vs. Web Application

To understand the benefits inherent to a web application, it helps to first understand how a typical fat client application might work. After being installed, the fat client application is launched by the user. The user interface provides interaction with the rest of the application and the app loads data that it needs from the user's hard drive. Many times, the data used by the application is in a proprietary format that is specific to the application. Although the richness of the user interface is often very good, the drawbacks of a fat client application are numerous. It is difficult to get new data or upgrades to the application without having another install process. Storing the data in a proprietary format makes it difficult to ingest different types of data or work with other applications without introducing specific interfaces to support the new functionality.

A web application solves several of the problems apparent with a fat client application. Since a web application is stored on a web server and launched by browsing to a specific URL in a web browser, the entire application deployment process is greatly simplified. Upgrading the application is also easier as the user will automatically see the new application when they browse to the URL, if the web app has been upgraded since the last time they used it. Data can be retrieved from a local hard drive as with a fat client application or it can be retrieved by requesting it from remote URLs on other networks.

To be able to ingest data from different sources, the need for data standards in web applications becomes very obvious. Groups such as the Open Geospatial Consortium (OGC) have defined geospatial related standards that many web applications implement. Another important difference is that extending a web application's functionality is often much easier than extending a similar fat client application. The server architecture of a web application can be designed in such a way that it supports a plug-in based approach so that individual pieces of functionality can be easily added or removed. However, the missing component for most web applications until recently has still been the responsive user interface that is common in fat client applications.

Moving from Web 1.0 to Web 2.0

Traditional web applications also known as Web 1.0 use a client-server model. The client, a web browser, issues

an http request to a server for a new page when the user clicks on a link. The web server, usually Apache or IIS, does some processing, retrieves information from legacy systems, does some crunching, and sends a formatted page of hypertext back to the client for display. This approach is the simplest technically, but does not make much sense from the user perspective. The reason for this is the latency, one to ten seconds, between when the user requests the page and when it finally loads. Because of this latency it is not possible to use *direct manipulation* user interfaces [1]. This class of user interface is greatly preferred by users [2].

The new model enabled by a set of technologies that are broadly called Web 2.0 eliminates the start-stop-start-stop nature of web applications. Instead, information is asynchronously downloaded to the browser using XML. JavaScript code in the browser caches this information when it is received from the server and displays it upon user request. Since the information is cached locally, the system can provide instantaneous responses and thereby support direct manipulation operations. JavaScript code in the browser handles interactions such as panning, zooming, scaling, and data validation. The advantage of the asynchronous requests for XML data is that users can continue working with the application's responsive user interface without losing their focus on the screen while data is downloading.

Historical Attempts at Creating a Rich User Interface

To casual web users the recent appearance of direct manipulation web interfaces may appear to be a sudden discontinuity. However, creating a rich user experience has been a goal for web developers since Sun first introduced Java and Applets back in 1995. From a historical perspective, JavaScript and DHTML were introduced as lightweight ways to provide limited client side programmability and improve user experiences. Macromedia introduced Flash and along with the term "Rich Internet Applications" in another attempt to enhance user experience on the web. Even though the vision for these capabilities has been clear for a while, these techniques have not caught on because of cross-browser incompatibilities as well as the difficulty in producing Rich Internet Applications. Developers have been unable to keep up with these browser changes, and a critical mass of applications have not be attained. What has emerged from an intense period of competition is the closest there has been to a de facto browser standard. By programming to this standard, it now is possible to deliver a direct manipulation interface in a web application.

Scalable Vector Graphics

Scalable Vector Graphics is an XML markup language for vector graphics. Using SVG it is possible to build both static and animated graphical displays in browsers. SVG is an open standard created by the W3C (World Wide Web Consortium) that is intended to compete with Flash, a closed proprietary technology. Adobe's Flash has been a successful graphics standard and is widely used to deliver multimedia content, timeline animations, and advertising content. Because of programming limitations, however, Flash has not caught on for application user interfaces. On the other hand, nearly all browsers now provide native support for SVG (or support through a browser plug-in). SVG also appears to be the graphics standard of choice for cell phones and mobile devices. Microsoft is expected to promote a different XML-based language called XAML for creating user interfaces. Although our components are written for SVG, our approach works equally well with either SVG or XAML. By abstracting our graphics layer, we are able to produce interactive components based on whatever rendering language the browser supports. This approach completes the missing piece in web applications. By providing the user with a responsive user interface in a web browser, the ability to create Rich Internet Applications is truly possible.

3. GeoBoost System Architecture

The GeoBoost system architecture is shown in Figure 1. GeoBoost servers ingest geospatial raster data, feature data, and business data from a wide variety of common data sources and then transform the imagery and data sources into protocols formats that GeoBoost thin client components can accept, image tiles and RSS, and perform database and other services for GeoBoost applications.

The problem these servers are solving involves restrictions in the thin client programming model. Thin client code is written in JavaScript which is a difficult programming language, has no access to databases, is restricted by security rules, and cannot maintain information from page to page as the user navigates through a site. These operations must be enabled using support from various other services. The AJAX JavaScript and browser programming model does make two types of operations easy. First, it is easy on a web page to issue asynchronous image requests by assigning a url to image src tags. As the images are retrieved, the browser automatically renders them on the page. Second, it is possible to asynchronously request XML files which the JavaScript may use to manipulate the web page. Because of the difficulty in programming JavaScript, the servers do the heavy lifting. It is more efficient to do protocol conversions, calculations, and as much computation as possible on servers rather than within the client. The reason is that thin client code is restricted and particularly difficult to debug.

Data Ingest
GeoBoost's strategy for data ingest is to accept data in as many formats as possible, with particular focus on open standards. These include raster image data, geospatial feature data, and generic business and geospatial data.

Server Layer
GeoBoost provides two types of servers. *Image Services* stream image tiles, either drawn from image repositories or created by rendering feature data as image tiles, to the client. *Application Services* stream XML to the client. The XML is usually formatted RSS, a simple XML protocol, or RSS namespace extensions. As mentioned above, the problem that Image Services imagery and Application Services solves for raw data involves thin client computational limitations. Browser-based code written in JavaScript is severely limited and it is not possible to do many computations that are easily done in fat-client or applet code.

GeoBoost Web 2.0 AJAX Client
The GeoBoost thin client layer is a Web 2.0 AJAX client and includes a development environment consisting of a set of visual components for building web-based graphical applications. Using a common data infrastructure, these components ingest image and RSS data and provide visualization and collaboration services. To make these components easier to use for non-programmers, GeoBoost provides JSP and ASPX tags that enable the visual components to be readily used on web sites.

GeoBoost's second client is a Windows Mobile application that ingests image tiles, feature data, and uses services provide by GeoBoost Servers for tracking. The application is aimed at GeoSpatial tracking, alerting, and collaboration involving mobile devices.

Application Layer
Using the GeoBoost platform, we have built a series of vertical applications focused around geospatial problems. The first application, called FUSION, focuses on Air Traffic Management and display plan positions, restricted airspace, and weather patterns. The second application shows the positions of employees and company assets wearing RFID-tagged badges within a building.

4. GEOBOOST VISUAL COMPONENTS

GeoBoost Visual Components
GeoBoost provides a broad set of visual components that includes maps, business charts, networks and graphs, trees and hierarchical displays, and a timeline. Each of these components has been developed using the *thinc* Interface framework, described below, and, when wrapped with the appropriate HTML code, may be independently positioned and moved on the browser screen. See Figure 2.

GeoBoost Graph Visualization
GeoBoost's graph visualization, shown in Figure 7, displays the nodes and links of a graph ingested from RSS. Being completely thin client, the graph is generated by JavaScript and displayed in the browser when new information arrives. The nodes, which are simply standard RSS items, are represented by any of several built in geometric shapes or custom icons as well as custom size. The edges, which are also RSS items, require only a from item id and to item id tag which correspond to the ids of the two connected nodes. The edges are drawn as connecting lines and have the ability to show direction by having an arrow pointing at the target node.

The graph has a robust set of functionality that allows user interaction and customization. It is capable of displaying the network as a radial or tree graph. In both drawing modes, the nodes and links have built in mouse events such as hovering to see a tooltip and right clicking for a menu of additional options. The graph also has the ability to zoom in or out for a closer look and pan by clicking and dragging the mouse. Labels for individual nodes are drawn on the graph for quick recognition of what RSS item is represented by a particular node. By default, the graph will find the minimum spanning tree of the network and use the middle item as the root node. However, any node can be specified as the root of the graph in order to see the relationships of that item more clearly. The graph allows dynamic changing of the root node by simply right clicking on an item and selecting the option to set as root. Tooltips for the graph may be text or images.

One issue for thin client programming is scalability. JavaScript is an interpreted language and is not particularly fast. We were pleasantly surprised with the performance of our graph drawing algorithms. The approximate time to layout and display a graph with 300 edges and 200 nodes on a standard laptop is approximately 1 second. Drawing labels increases the time by a factor of four. Interactive operations such as tooltips and selections are essentially instantaneous.

GeoBoost Map Visualization Component
The GeoBoost Map is a rich visualization that allows dynamic interaction with raster imagery and vector feature data. It supports animated zooming and panning as well a number of different API functions that allow programmatic interaction with the Map. Multiple Maps can be synchronized together so that they pan and zoom

at the same rate, which is used to create "Overview" Maps and "Magnification" Maps. See Figure 2 andFigure 3.

Raster imagery can be ingested from OGC compliant sources (WMS and WFS) or from custom sources. Multiple raster sources can be "layered" on a Map and re-ordered to combine street maps and satellite imagery from different sources. In addition, the transparency of each raster source can be individually adjusted to allow other layers to show through. See Figure 4.

Feature data is ingested using the standard GeoRSS format which translates very cleanly to and from other popular geospatial formats. Many useful interfaces exist to allow real-time and historical data to be pulled in and updated on the Map incrementally, providing optimal performance for web applications. Multiple overlays of feature data can be placed on a Map, each having a different data source. A variety of styles can be applied in real time to feature data to encode meaning, including color, width, opacity, etc. The advantage of drawing each data item as a vector feature is that the item is able to be highlighted and selected in a very intuitive way. The Map supports sweep selection to select multiple items at a time. The unique aspect of this interaction model is that it causes the events to propagate to other linked Maps and visualizations (Timelines, Bar Graphs, Pie Graphs, etc) which are displaying the same data.

GeoBoost Time Line and TimeWheel

GeoBoost Timeline, shown in Figure 8, is an interactive visualization that plots temporal data based on their date and time. Like the Map, the Timeline supports dynamic zooming and panning so the user can scroll through time and find data. Multiple overlays also allow data from different sources to be stacked on the Timeline and then shown or hidden. The Timeline supports six different scales (Year, Month, Day, Hour, Minute, Second) and several different drawing modes (Circles, Ticks, Spheres, Custom Icons) and is very easy to configure and customize. Another unique feature is its ability to plot data according to a specified property along the Y axis rather than using the default "Best Fit" algorithm, which essentially turns the Timeline into an interactive Line Chart with Time as the X axis. The data on the Timeline supports tooltips and can be highlighted or selected, which will cause the events to propagate to other linked visualizations. Synchronizing two separate Timelines that display the same data is also useful because they will pan at the same rate and provide a summary and detailed view of the same data over time.

TimeWheel is an interactive visualization that plots temporal data according to time of day and day of week. It is useful for identifying patterns in data that is linked to a Timeline or Map. The TimeWheel is shaped like a clock with each segment representing a different hour of the day. Concentric rings are labeled to designate a different day of the week. Once the data items are plotted, it is very easy to select or highlight clumped data items that meet certain criteria ("All data that occurred on Saturdays and Sundays between 9:00 and 12:00" or "All data that occurred on Fridays at 10:00pm"). In addition, the segments can be drawn equally or can be drawn proportional to the number of items that occurred in that slice (hour of the day). This provides a simple means of volume analysis to determine which hours of the day represent the most activity. A day/night layer can be added to the TimeWheel to show which hours of the day represent daytime and which represent nighttime so that it becomes obvious whether data items occurred during the day or night.

GeoBoost Portal

GeoPortal, shown in Figure 9, is an AJAX-enabled portal site for customized workflow. It enables users to include content of interest, customize the content, and automatically updates the content. Content is, of course, just web pages that are included via a simple interface. Widgets use the GeoBoost visualization library to create a consistent view of data through multiple displays. GeoPortal content widgets are stored in a Ruby on Rails backend which maintains user preferences, manages data feeds, and provides the logic required to include content.

In a simple use case, a user might display information from a data feed in several ways by including bar-chart, line-chart, and geospatial widgets on a single page. Additional tabs can be added to the display to allow the user to build situation-specific applications designed to include only the information necessary to the task at hand. These applications can then be made public for dissemination to others working in the same area. Simple collaboration features built into the Ruby on Rails backend allow users working on the same application to share information.

5. JAVASCRIPT VISUALIZATION FRAMEWORK

GeoBoost visual components are built using a JavaScript AJAX visualization framework that we call *thinc* Interface. The framework uses the browser's native rendering layer, SVG currently and in the future WPF/E (Windows Presentation Foundation Everywhere or XAML), for building thin client visualizations. In contrast to open source efforts such as DOJO [6], Prototype [7] and Microsoft's Atlas [8] that focus on user interface components, our focus is on visualization. The SVG graphics API that *thinc* uses is a W3C standard that is natively supported by many browsers including Firefox.

RSS and GeoRSS Data Ingest

Each of these components ingests data to be displayed based upon RSS and namespace extensions of RSS that provide the components with specialized information. RSS is a simple protocol for publishing information. The basic content of an RSS stream is a set of metadata tags organized into items that describe content. The required tags in each item are <title>, <link>, <description>, <pubDate>, <guid>. The <link> tag is generally a hyperlink to retrieve the item content.

GeoRSS extends RSS by adding location information. The location may be a point, e.g. *<georss:point>45.256 -71.92</georss:point>*, or more complex polygon. We have extended the GeoRSS specification to add specific shapes of interest such as ellipses, sectors, and slices using a *thinc* namespace extension. For example, the specification

```
<georss:where>
   <thincml:sector center="38.82, -77.12"
   radius="4000" insideradius="3000"
   startangle="90" arcangle="45"/>
</georss:where>
```

Represents a sector of a circle or a pie wedge if the inside radius is zero.

Linking Between Visual Components

RSS organizes information into sets of *items* within a channel. Our GeoBoost visual components have the capability to ingest RSS feeds and display the information using different visual presentations. Our Map Components, for example, display GeoRSS by geo-positioning entities in the browser over a map. Tooltips and other mouse operations are automatically linked among different visual components showing the same item sets. This approach toward visual components makes it easy to create rich visual applications built around RSS.

Supporting Multiple Browser Graphics APIs

Rendering 2D vector graphics in a web browser is challenging for many reasons. Lack of IDE support for JavaScript makes it difficult to debug and step through the JavaScript code needed to render vector graphics that manipulate the browser's DOM (document object model). In addition, cross-browser coding is even more difficult because all browsers do not support the same type of vector graphics. At the current time, the latest browser releases of Mozilla Firefox, Opera, and Safari all have native support for SVG, which is a W3C standard for vector graphics. However, Microsoft Internet Explorer supports VML natively, which is a similar vector graphics library designed by Microsoft. By using a plug-in, Microsoft Internet Explorer can also support SVG documents that are included through the use of an "embed" or "object" tag. The current varying state of vector graphics capability between common browsers introduces substantial complexity for the typical web developer who typically just needs to do simple things like drawing points, circles, rectangles, or other polygons.

The *thinc* Graphics classes and API solve this problem by abstracting the details of programming against the individual browsers and allow the web developer to program against a single graphics API. The *thinc* Graphics classes interrogate the browser's rendering environment at run-time to determine the vector graphics it supports, and then conditionally execute the appropriate JavaScript code to create the vector objects. For example, a user browsing with Firefox to a page that had been created with the *thinc* Graphics classes would see the native SVG that the code generated. However, a user browsing to the same page with Internet explorer would see SVG in an embed tag if they had a plug-in that supported SVG otherwise, the *thinc* Graphics classes would fall back to generating the native VML that is supported in Internet Explorer. Of course, this behavior can be overridden by the developer to force specific rendering engines to be used.

This approach allows a web developer to make simple calls to the API such as "DrawRectangle" or "DrawCircle" and not be concerned with the specifics of each browser's rendering capability. The *thinc* Graphics classes provide this layer of abstraction and make it much easier for the developer to use native vector graphics in their application. In addition, this model is flexible and easy to extend so that when new browsers are released, the same application code should work without requiring any changes. New classes can be added to the underlying *thinc* Graphics code without changing the existing signature of any of the *thinc* Graphics API calls.

JSP and ASPX Server Tags

GeoBoost exposes functionality to the integration developer through HTML server tags. Server tags allow the developer to define GeoBoost widgets inline using an HTML syntax rather than pure JavaScript code. As shown in Figure 5 and Figure 6, HTML tags enable developers to use sophisticated visual components by simply including them on web pages.

For each of GeoBoost's visual components we have developed both JSP and ASPX custom HTML tags. For ASPX these tags are also referred to as .NET Web Controls. The tags, which all start with <*thinc*> provide a programming abstraction layer on top of the JavaScript library for html programmers. When processed by the web server, the <*thinc*> tags in the web page are converted to the corresponding JavaScript on the web

server when the page is loaded.

The advantage of GeoBoost's HTML tag functionality is that it greatly simplifies authoring web pages. Developers can program against the JSP Custom Tags or .NET Web Controls for fast integration into their application, or they can program directly against the JavaScript API for optimal flexibility. The tags are identical across platforms. Thus, a page created using JSP Custom Tags will run in the .NET environment, and vice versa. The tag library exploits HTML's inherent notion of containment, 'embedding' map data sources within a <*thinc*:map /> tag. In this way the complex underlying JavaScript is shielded from the end developer, enabling rapid prototyping and quick integration

Benefits to End Developer

The HTML syntax is more readable and therefore more maintainable for the end developer. Server tags also allow intellisense and syntax checking when used in the standard programming environments Eclipse and Microsoft's Visual Studio. By integrating with the popular IDE's, "Drag and Drop" support is enabled, allowing the developer to design and code the application using typical Rapid Application Development (RAD) techniques. Syntax errors are caught at integration time rather than during run time. GeoBoost Server Tags provide meaningful error messages for the integration developer. Regular HTML error messages provide only generic error messages and codes (for example, Error 500, Server Error), whereas GeoBoost server tags provide detailed error messages.

Hierarchy within Tags

Many of the GeoBoost widgets express a hierarchal relationship. For example the Map widget might contain 3 overlay regions which to display, or the BarChart widget might contain 3 data series to plot. Unique to GeoBoost, the server tag library expresses these relationships using XML's natural syntax, familiar to all developers. The standard way to express the relationship is by embedding ID's and using lookup indirection to deduce hierarchy. GeoBoost tags express this naturally, as in the following sample:

```
<thinc:map latitude="35.0" longitude="45.0">
   <thinc:ItemCollection>
     <thinc:GeoRssDataConnector url="source1" />
     <thinc:GeoRssDataConnector url="source2" />
   </thinc:ItemCollection>
</thinc:map>
```

In the above sample code, a map is drawn centered at latitude 35, and longitude 45. Drawn in the map is the data from "source1" and "source2". This hierarchal relationship is clear from the code.

Using hierarchical containment, it is possible to include several different image and datasets into *thinc* components. For example, Figure 6 shows an example where our Map control is attached to two different WMS image servers. Both sets of imagery are retrieved by the client and combined within the browser. The result is shown in Figure 4 where NASA imagery is combined with political boundaries by merging the images in the browser.

Annotating and Editing Visual Components

As shown in Figure 10 GeoBoost provides the capability for users to edit and annotate the visualizations. The edits and annotations automatically propagate using AJAX messages and are persisted in GeoBoost's collaboration database. In Figure 10 two users of the Map component are simultaneously editing the map in different browsers. The each of the browsers polls the collaboration database every second or two and picks up any new changes. The result is that of a real-time geospatial wiki. The same capability may be used with any of GeoBoost's visual components.

5. *GeoBoost* Mobile Client

Screens on mobile devices are typically small, have differing orientations and dimensions, have varying pixel resolutions, and thus require special attention. Additionally, mobile devices typically have slower processors, minimal battery life, and constrained local storage. These factors must be weighed when integrating a mobile appliance into the GeoBoost platform. Creating a compelling mobile client requires careful consideration of all these constraints. To fully take advantage of mobile applications, the GeoBoost server platform has been extended to process and deliver data in new ways. By off-loading this processing to the server, the mobile device resources were conserved and the user experience was enhanced.

Screen resolutions and aspect ratios (orientation)

Mobile devices have varying screen resolutions and physical dimensions. Some devices (notable the Palm Treo) have a square screen with a resolution of 320x320. Other smart-phones have rectangular dimensions, with orientations in both Landscape and Portrait. On the far end of the device scale, small handhelds have screen resolutions of 800x600 with a physical dimension of 5". A strategy of delivering "a standard set" of images to clients, and letting the mobile device handle the difference would be disastrous since the devices do not have the processing power required for intense image manipulation. Rather, GeoBoost server allows image manipulation to be done server side, and passing on images with resolution and dimensions specific to the device.

Processor Power, Battery Life

On a mobile device, battery life is directly proportional to processor power and processor workload. If we program the device to do heavy computations or image manipulation, we have a double effect on the device: slowing performance overall, and draining the battery. Battery life is at a premium on the devices, and manufactures are trying to increase the battery power by continually reducing the processor power. Recognizing this constraint, the GeoBoost server platform was extended to serve images "pre-cooked" for the specific device. This allows the device to simply display the map image, rather than needing to manipulate the image for its hardware.

Color Depth

Current mobile technology delivers limited color depth and typically only displays 24-bit color (as opposed to 32-bit color standard on a desktop). This limited GeoBoost's ability to finely calibrate data points on the map. GeoBoost server side changes were made to accommodate the reduced color depth.

Glyph Size

Mobile devices are designed to be used within 5"-6" from the users eyes, whereas desktop monitors are 17" - 25" from the eyes. Further, some handhelds are specifically designed with touch interfaces (the Windows Mobile platform), whereas others are designed for stylus use (TabletPC). Fingertip use requires bigger icons with larger hotspots to be activated, while with stylus use, a smaller hotspot is possible. Some mobile devices (Smart Phones) allow neither, and rely on the telephone input digits or joysticks (blackberry pearl) for input. GeoBoost Mobile has algorithms and tolerances defined to allow varying sized hotspots depending on device. The larger hotspots require larger icons, so icon collision detection algorithms are needed to space hotspots accordingly.

GeoBoost Mobile

A prototype rich client was developed as proof points for our theories. The prototype consumed map images from GeoBoost server, and delivered them to the device to allow zooming, panning, and scrolling on the client. The architecture of our mobile application is similar to that of the GeoBoost client: a Model-View-Controller pattern is used to create several different interfaces to a single central data model.

The mobile application connects to Map Services over the internet and fetches map tiles for the currently displayed area. Feature data is provided by Data and Application services in the form of GeoRss. The GeoRss is parsed by our RSS library and imported into the application's central data store. The application then uses its native graphics libraries to represent the feature data on the map.

Mobile applications are especially well suited for low-bandwidth or sporadic internet access. Since the application does not depend on a web browser, additional optimizations such as local tile caching can be introduced to counteract the limitations of the network.

6. SUMMARY AND CONCLUSIONS

We have created a Web 2.0 AJAX thin client mapping framework that we call *GeoBoost™*. Using our framework we are able to build browser-based geospatial applications that run on top of open geospatial standards, RSS and GeoRSS, and provide rich geospatial visualizations built using Scalable Vector Graphics. The framework is unique in that applications built using it have the richness of traditional desktop applications and the reach of browser-based systems. It provides an open platform for thin client web mapping.

7. REFERENCES

[1] Stuart Card, Jock Mackinlay, Ben Shneiderman, *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann, 1999.

[2] Ben Shneiderman, D*esigning the User Interface*, Addison Wesley, Third Edition, 1998.

[3] National Geospatial Agency, *Military Standard Raster Product Format*, 6 October 1994.

[4] www.lizardtech.com

[5] Dona Maurer, "Usability for Rich Internet Applications", *Digital Web Magazine*, 20 February, 2006.

[6] Dojo Foundation, "dojo, the Javascript Toolkit", available at http://dojotoolkit.org/.

[7] Sam Stephenson., "prototype JavaScript Framework", available at http://prototype.conio.net/.

[8] Martin LaMonica, "Microsoft gets hip to AJAX," CNET, 27-June 2005, available at http://news.com.com/Microsoft+gets+hip+to+AJAX/2100-1007_3-5765197.html

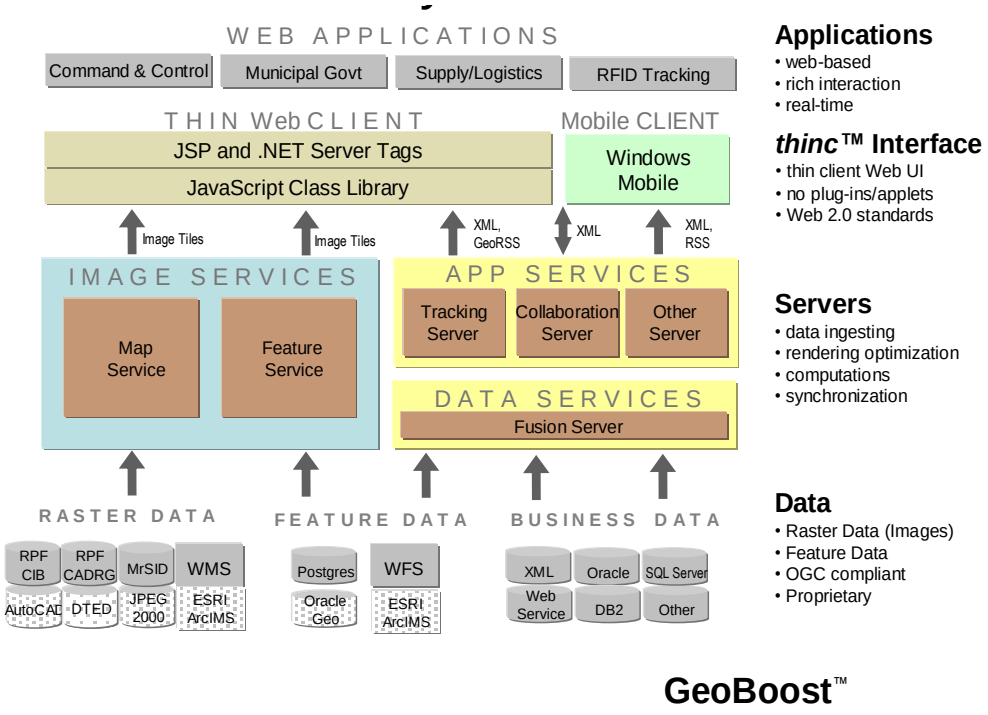


Figure 1 GeoBoost System Architecture.

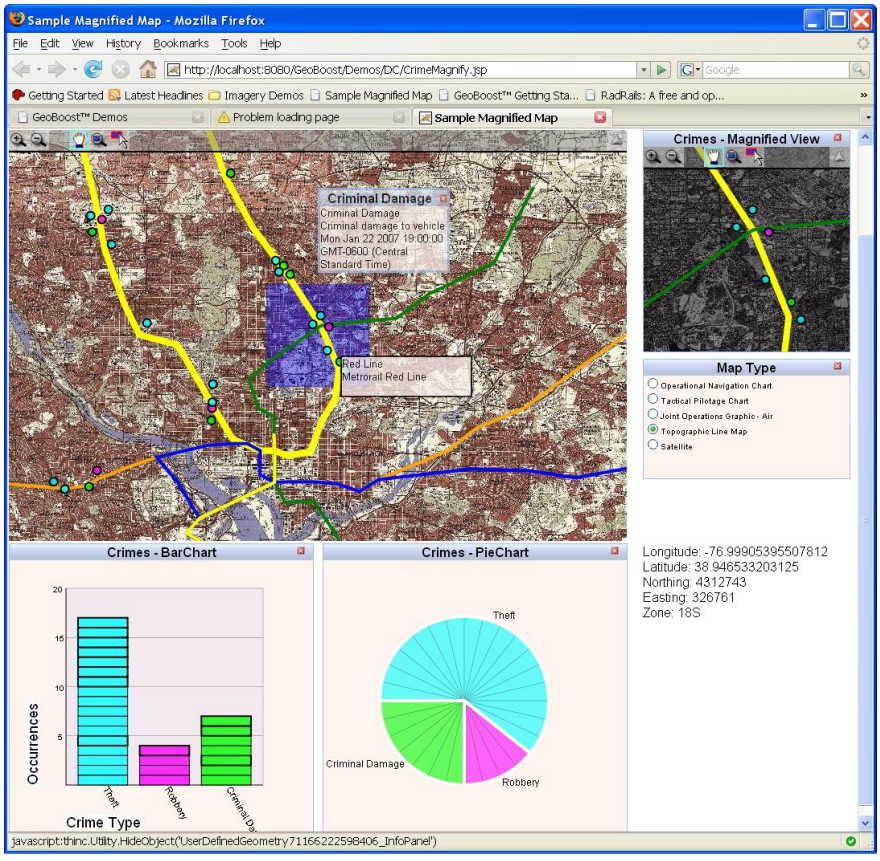


Figure 2. Linked GeoBoost Maps with Standard Business Charts.

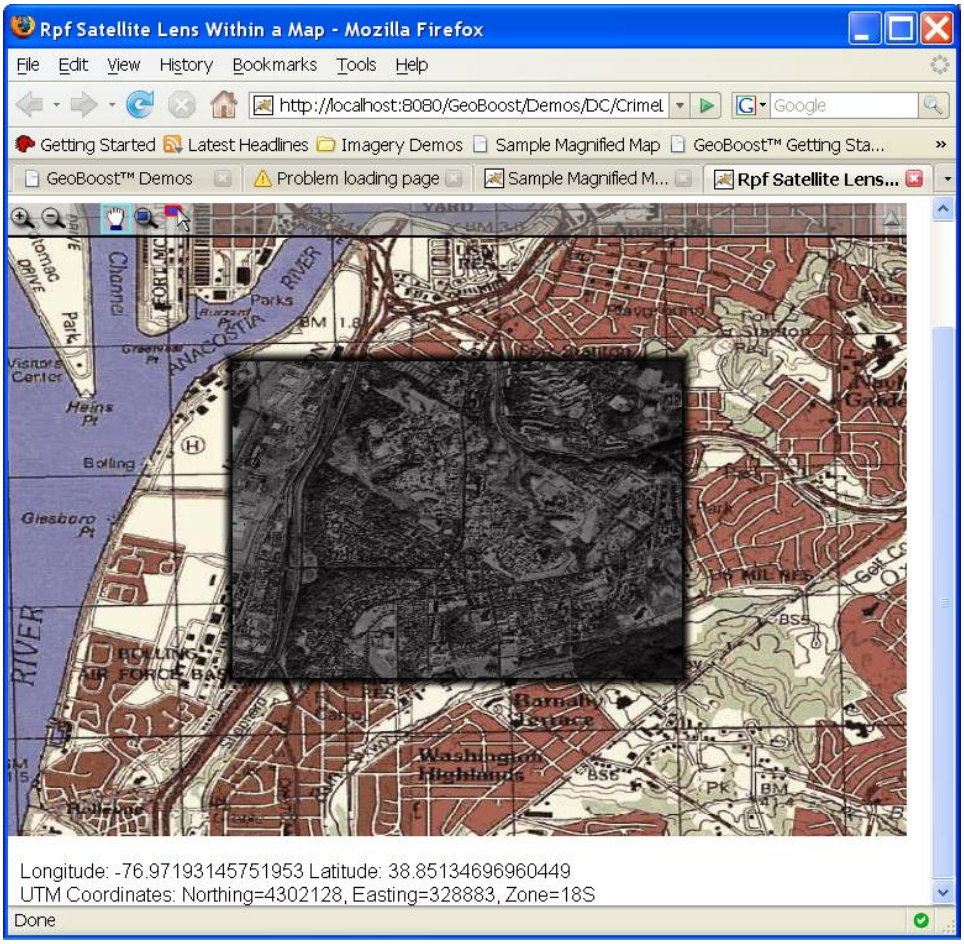


Figure 3. GeoBoost Map Component Merging Raster and Map data for Washington DC.

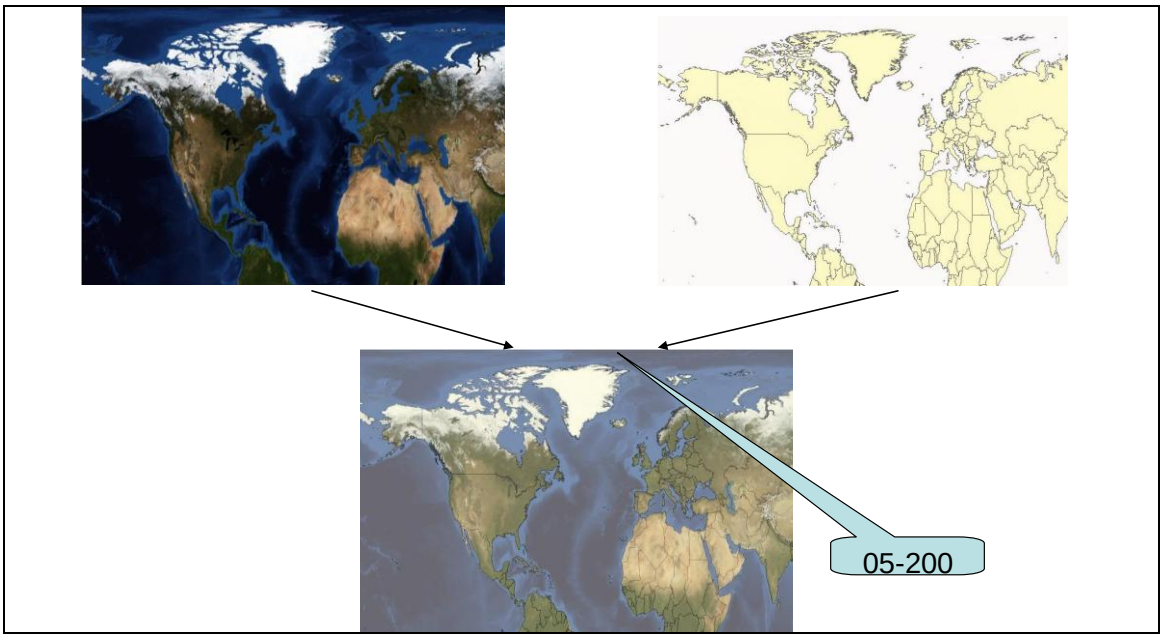
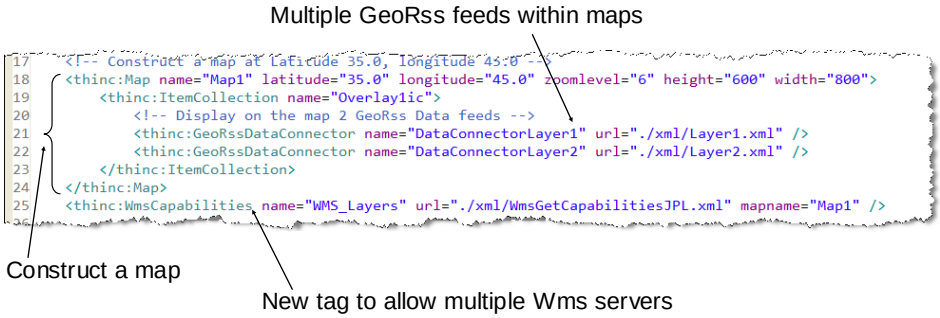


Figure 4 NASA image data is combined with feature data rendered as geopolitical outlines from two different web servers using GeoBoost's Map Visualization Component.



Tags that are contained within other tags are feed automatically into the “parent” tag. This allows a natural syntax, and obviates the need to ‘hardcode’ the name in the data tag.

Figure 5 JSP and ASPX Server Tags simplify programming using *thinc* Interface Components. These six lines of HTML code put a map on the web page and attach it to two GeoRSS feeds.

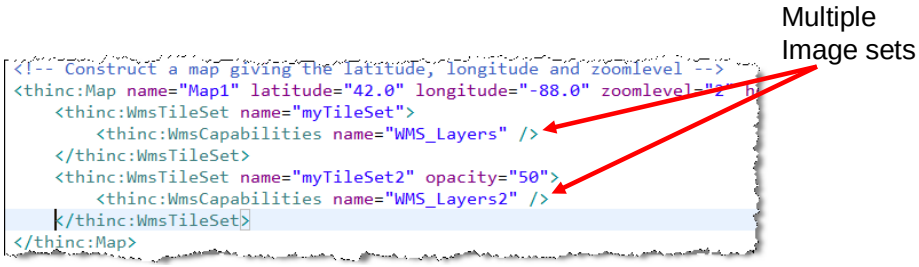


Figure 6 *thinc* HTML tags for including multiple image sets on a map.

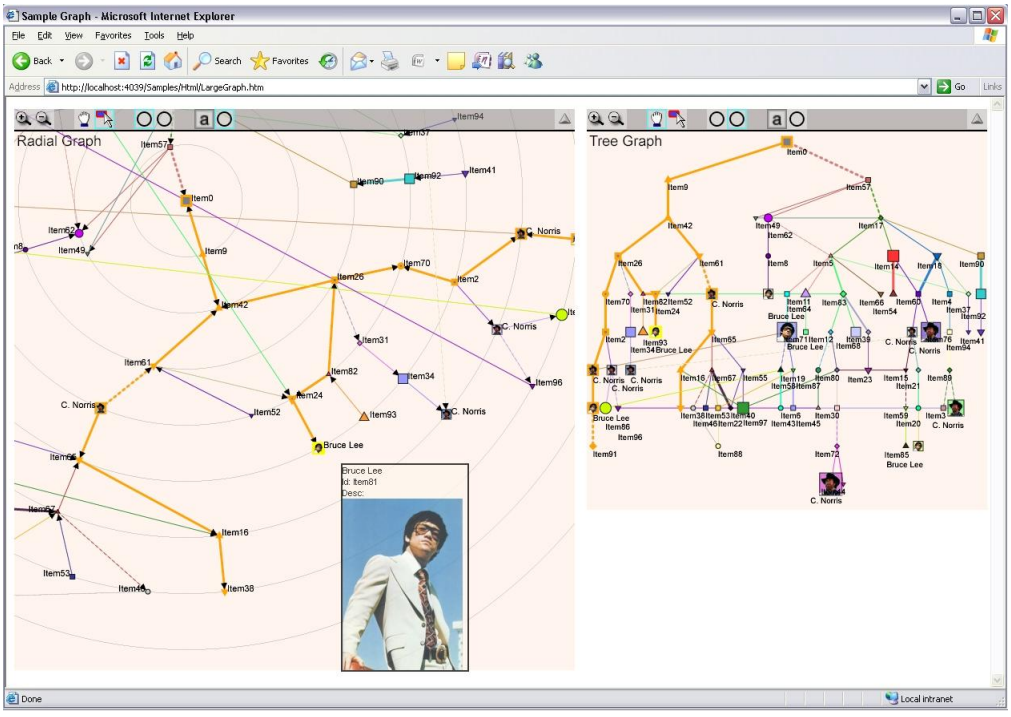


Figure 7. GeoBoost Graph Visualization Component.

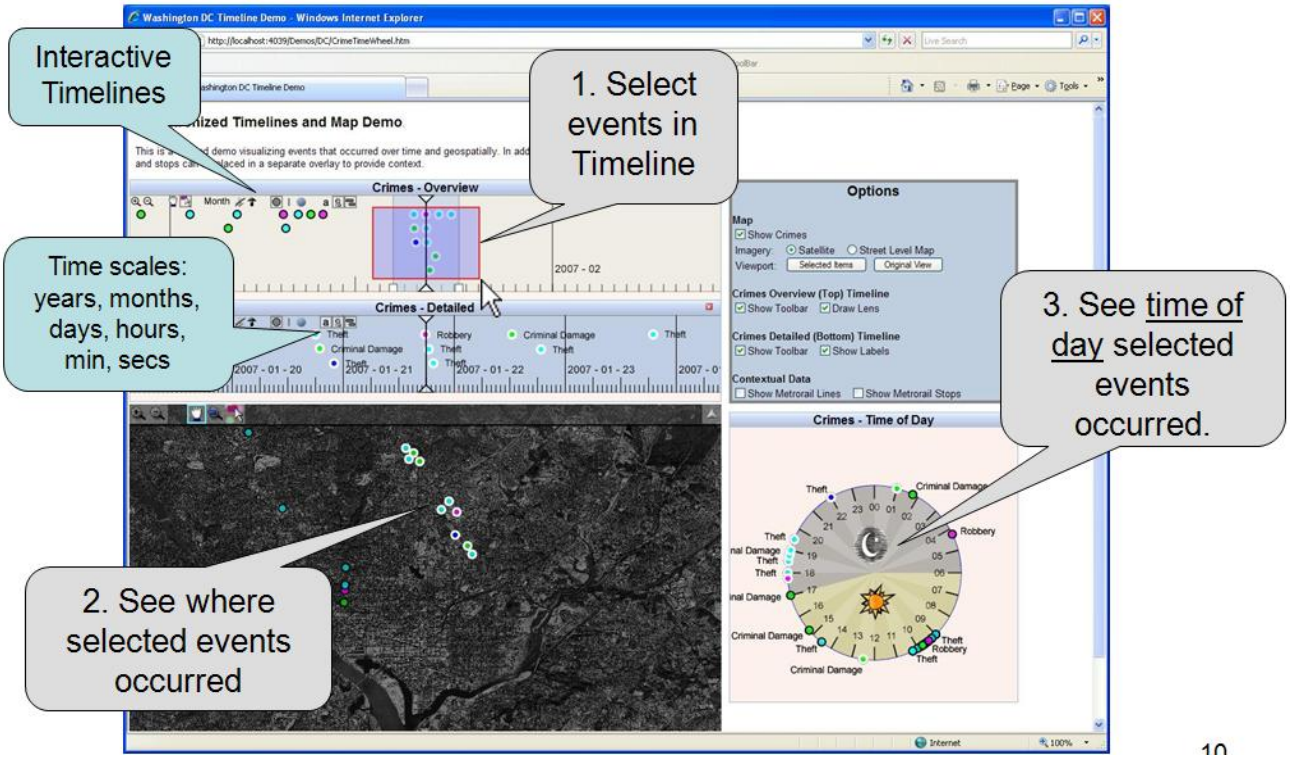


Figure 8. GeoBoost Time Line and TimeWheel linked to map showing crime locations in Washington DC.

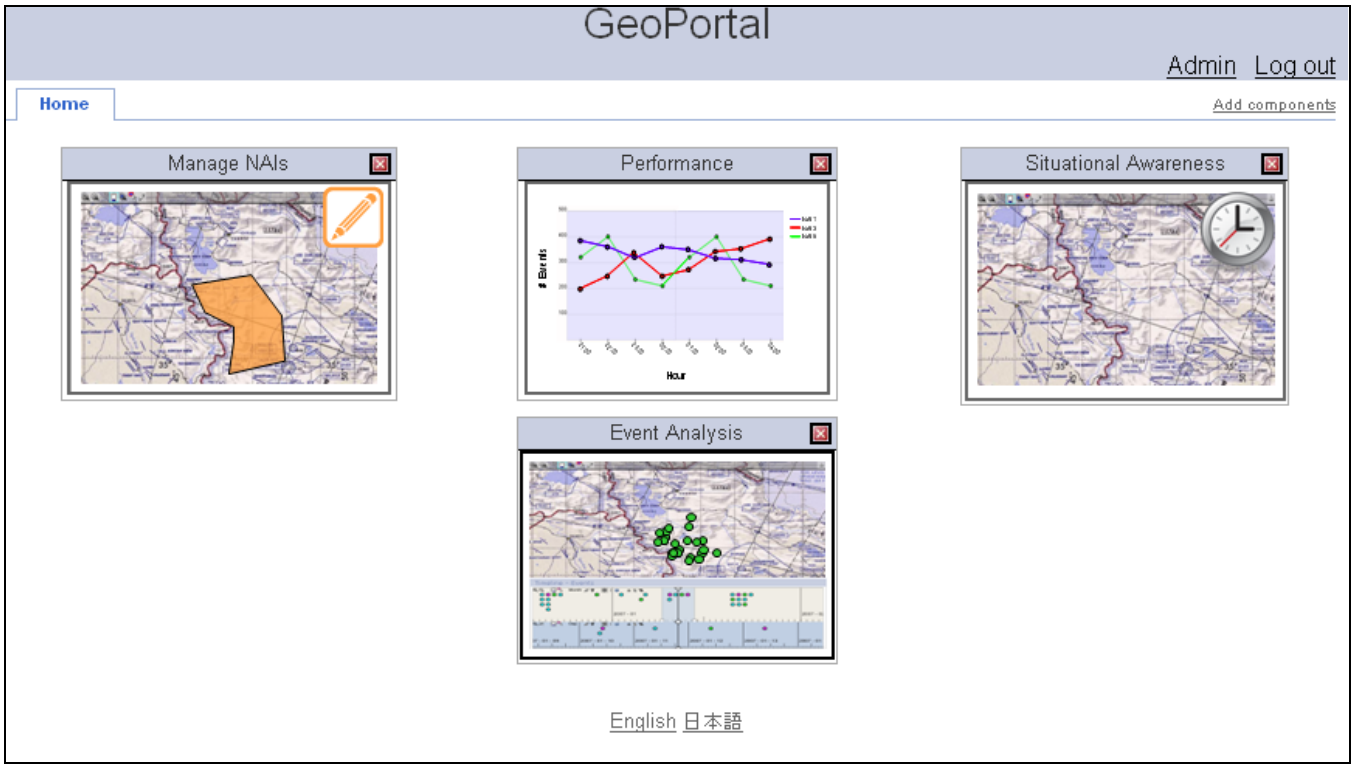


Figure 9 GeoBoost's live AJAX portal.

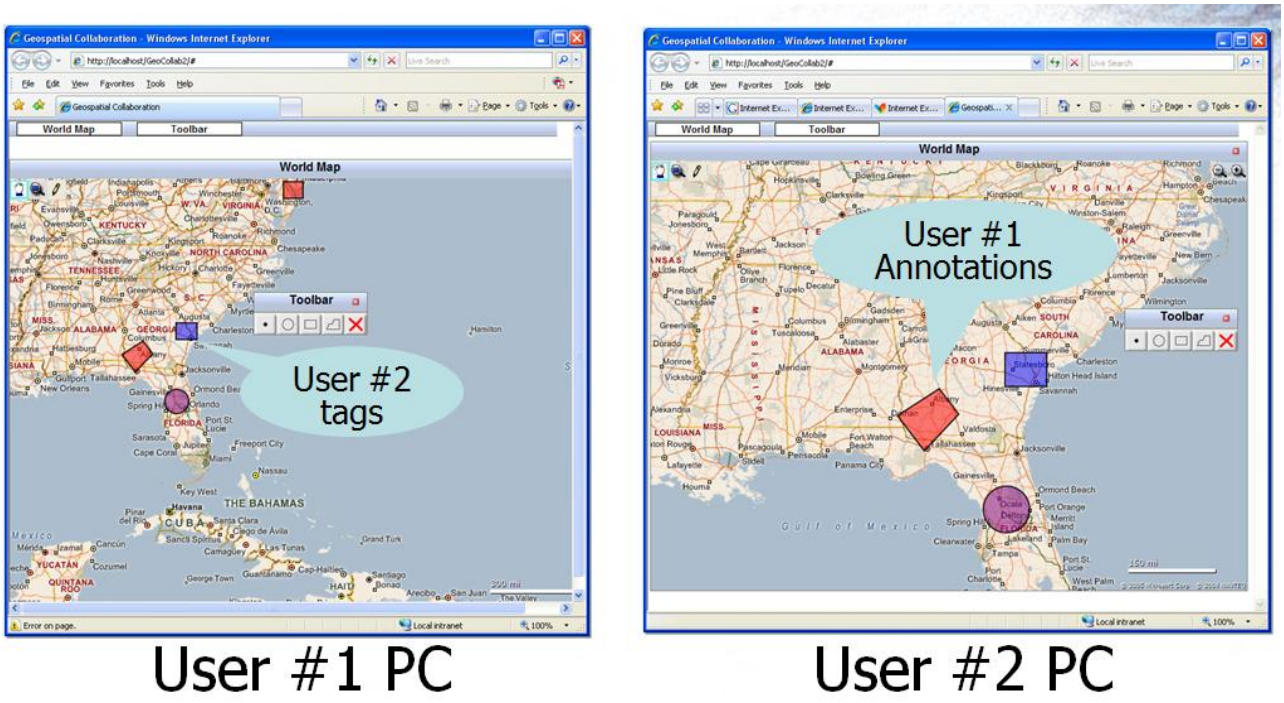


Figure 10 GeoBoost real-time collaboration. Edits and annotations on the map and other visual components propagate using AJAX within a couple seconds.

Figure 11 GeoBoost Tracking Application on Mobile Device.