JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 4 (April 2004)
Special issue: TOOLS USA 2003

WebGD: A Framework for Web-Based GIS/Database Applications

Paphun Wangmutitakul, Toshimi Minoura, and Alec Maki, Oregon State University, U.S.A.

Abstract

We have developed a framework for Web-based GIS/database applications which allow users to insert, update, delete, and query data with a map interface displayed by Web browsers. The framework was designed so that a Web-based GIS application that uses ArcIMS as a map server can be easily created, customized, and maintained. In order to achieve this goal, we have created our framework as a collection of ASP.NET *custom server controls*. In creating a Web-based GIS application, instances of our custom server controls can be picked and placed on ASP.NET Web pages like standard Web controls.

1 INTRODUCTION

We designed and implemented a framework for creating Web-based GIS/database applications. This framework, called WebGD, enables users to share spatial and non-spatial data across the Internet. Using an interactive map interface, users can view any particular area of the map. Also, information associated with geographical features on the map can be accessed and manipulated. Due to their Web-based nature, WebGD applications can be accessed from anywhere on the Internet using a standard Web browser.

WebGD uses ArcIMS for providing interactive map images and ArcSDE for managing spatial data. We developed WebGD as a foundation for rapid development of Web-based GIS/database applications. WebGD includes a set of ASP.NET *custom server controls*. A server control is a class that can be instantiated as a user interface component and placed on an ASP.NET Web Forms page. When an instance of a server control is added to an ASP.NET page, its look and behavior can be customized by setting property values. We developed a collection of custom server controls that encapsulate functionality needed for a Web-based map interface. These controls provide user interface elements such as a map, a layer list, and a toolbar. Using WebGD server controls, a map interface can be created by simple pick-and-place operations.

The WebGD framework also uses our *ArcSDE.NET* assembly, which allows a .NET application to access and modify ArcSDE-managed data. Since the .NET Framework executes applications in a *managed* environment, it does not allow direct access to the *unmanaged* C programming interface of ArcSDE. Managed code is executed within the .NET environment, which provides such services as garbage collection and security restrictions, whereas unmanaged code is executed without these services. To get around this restriction, *ArcSDE.NET* was written in managed C++, which allows managed and unmanaged code to be mixed. Thus, *ArcSDE.NET* enables WebGD applications to perform operations not supported by ArcIMS, such as inserting, updating, and deleting geographical features in a spatial database.

In this report, we focus on custom server controls we developed for use with the ArcIMS *ActiveX Connector*. Section 2 provides an overview of the organization of WebGD applications. In Section 3**,** we describe components of the WebGD interactive map interface by using a sample WebGD application, *Yolo County Soil Viewer*. In Section 4, we give implementation details of WebGD custom server controls. In Section 5, we give an example WebGD business application, *Motels in Oregon*, that demonstrates many features covered in this paper. Finally, we summarize the major benefits of the WebGD framework and give a direction for further development.

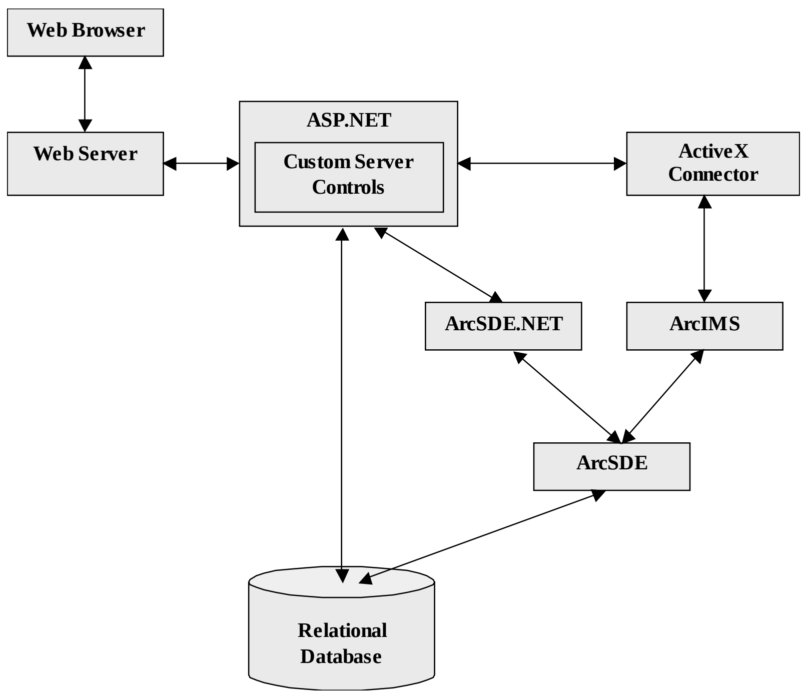


Fig. 1: Organization of WebGD.

2 WEBGD ORGANIZATION

The organization of WebGD applications is shown in **Fig. 1**. The relational database and the ArcSDE server in combination manage geographical data. The ArcIMS Internet map server generates maps to be displayed on a Web browser by using geographical data provided by the ArcSDE server. The Web pages are dynamically generated by the server-side scripts written in ASP.NET. When a server side script receives a request to generate a map, the script forwards that request to the ArcIMS server through the *ActiveX Connector* COM component. The *ArcSDE.NET* assembly was developed to allow ASP.NET scripts to directly access ArcSDE spatial data. Direct access provides such additional functions not supported by ArcIMS as inserting and deleting map features.

ASP.NET

ASP.NET, a subsystem of the Microsoft .NET Framework, is for developing dynamic Web-based applications. ASP.NET provides an integrated set of .NET classes, including those for data access, XML, graphics, and Web server controls. In addition to the .NET Framework class library, .NET allows COM components to be used in .NET programs. Thus, to access an ArcIMS application server, an ASP.NET script for an ArcIMS map interface can use ArcIMS *ActiveX Connector*, which is a COM component.

Furthermore, ASP.NET has introduced *server controls*. A server control is a class that encapsulates both a view and behavior of a user-interface component. An instance of a server control can be embedded in an ASP.NET `.`*aspx* page, which can contain ordinary HTML controls and ASP.NET server controls. Server controls are converted into HTML elements when an ASP.NET script containing them is executed by a Web server.

ArcIMS Server

ArcIMS, a software package from ESRI, can be used to create a map server. An ArcIMS server provides GIS functionality for a Web-based map interface. When a user performs an operation that requires a new map image, ASP.NET uses the *ActiveX Connector* to send a request for a map image to the ArcIMS server, which then returns the URL of the newly generated image. When a user clicks on the map to select a geographical feature, the *x*- and *y*-coordinate values of the clicked point are provided to the ASP.NET application. Based on these coordinates, the application defines a spatial filter for locating the selected geographical feature.

Custom Server Controls for ArcIMS ActiveX Connector

Our custom server controls for ArcIMS *ActiveX Connector* were developed as ASP.NET server controls, and they encapsulate common functions and interfaces found on ArcIMS map interfaces. We currently provide three server controls: `ArcImsMap`, `MapToolbar`, and `LayerList`. An `ArcImsMap` displays a map image and provides methods to manipulate the map. A `MapToolbar` allows the user to select a tool to operate on a map interface. A `LayerList` displays the list of the layers currently shown on the map, and it also allows the user to set layer visibility and to select an active layer.

ArcSDE Server

ArcSDE supports an object view of such spatial data as map layers and geographical features stored in tables in a relational database. Business data can also be stored in a relational database along with the spatial data. Although both spatial and business data managed by ArcSDE can be retrieved through ArcIMS, ArcIMS does not support such functions as inserting, deleting, and updating data.

ArcSDE.NET Assembly

ESRI, the company that developed ArcSDE, provides C, Java, and ArcObjects APIs for client programming. Although the ArcObjects COM component provides the most complete functionality, ArcObjects cannot be used by Web-based applications because of an ESRI license restriction. Therefore, the ArcSDE.NET assembly has been developed to allow scripts written in .NET languages, such as C#, to access an ArcSDE server through the ArcSDE C API.

The ArcSDE.NET assembly calls the ArcSDE C API by using Microsoft .NET C++. C++ is the only language in .NET Framework that can be used to call *unmanaged* code directly. To create the ArcSDE.NET assembly, a pair of *managed* and unmanaged C++ modules have been created. When receiving a request from a user, an ASP.NET script calls an appropriate function in managed C++ modules in ArcSDE.NET. The managed module uses the unmanaged module to access to ArcSDE API. By calling C API functions from unmanaged C++ functions, unmanaged structured data from the C API can be properly passed to a C function. Moreover, an overhead of switching between managed and unmanaged codes can be reduced by combining several C API function calls into one function call in an unmanaged class.

3 INTERACTIVE MAP INTERFACES

Each of our custom server controls provides a user-interface element for a WebGD map interface. A unique feature of our map interface is that geographical features can be inserted and deleted from the map and that information associated with them can be retrieved from a database through the interactive map interface. In describing components and functions of a WebGD map interface, we will use the *Yolo County Soil Viewer* application as a running example.

Map Interface of the Yolo County Soil Viewer

The map interface of *Yolo County Soil Viewer* consists of four Web Forms pages: `tools.aspx`, `tools2.aspx`, `map.aspx`, and `contents.aspx`. They are contained in four frames as shown in **Fig. 2**. Both `tools.aspx` and `tools2.aspx` contains a `MapToolbar`. `map.aspx` contains an `ArcImsMap`, and `contents.aspx` contains a `LayerList`.

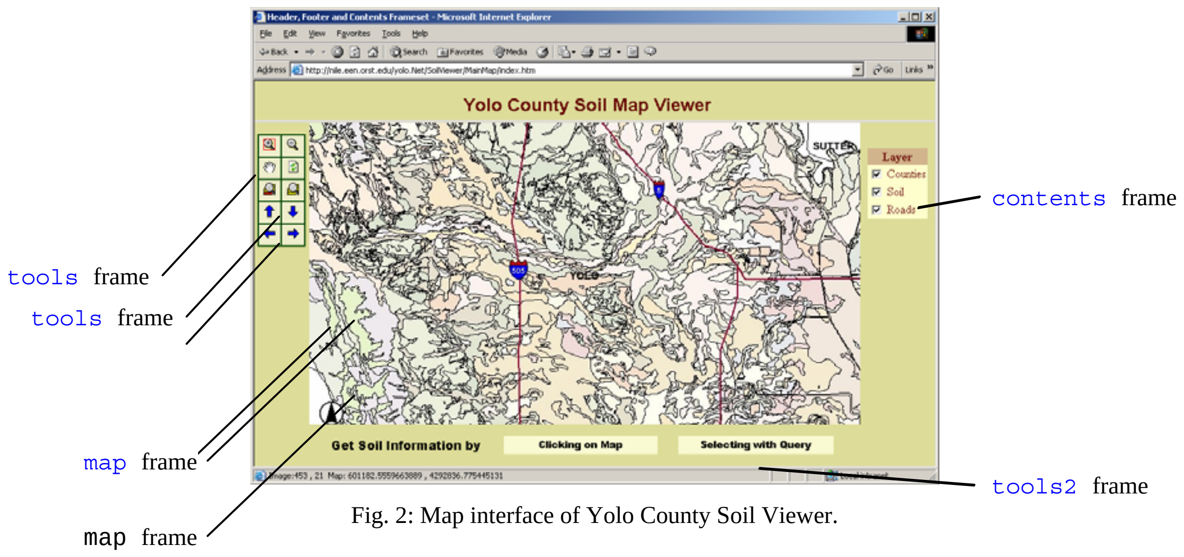


Fig. 2: Map interface of Yolo County Soil Viewer.

Using Yolo County Soil Viewer

To zoom into an area of interest, the user needs to select the *zoom-in* tool. Once she selects the zoom-in tool, she can press the left mouse button at the top-left corner of the area to zoom in and drag the cursor to the lower-right corner of the zoom-in area, drawing a rectangle, as shown in **Fig. 3**, that represents the area for zooming.

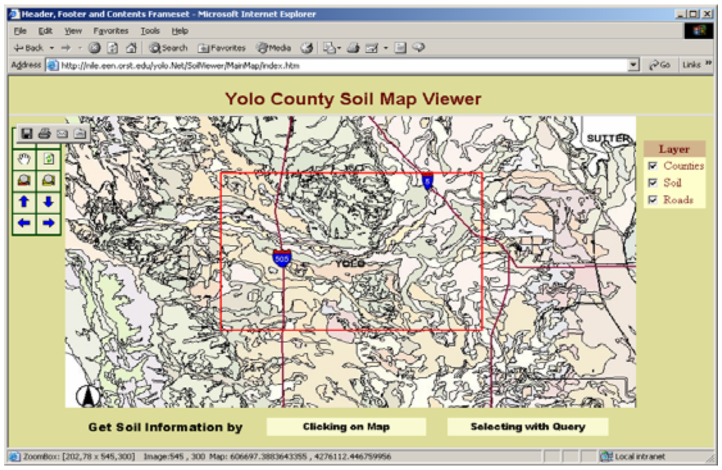

Header, Footer and Contents Frameset - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://nile.een.orst.edu/yolo.Net/SoilViewer/MainMap/index.htm
Yolo County Soil Map Viewer
SUTTER
YOLO
Layer
Counties
Soil
Roads
Get Soil Information by
Clicking on Map
Selecting with Query
ZoomBox: [202,78 x 545,300] Image:545 , 300 Map: 606697.3883643355 , 4276112.446759956
Local intranet

Fig. 3: Zooming in to an area of interest**.**

On the Web server, the executed ASP.NET script requests a new map image from the ArcIMS server. The ArcIMS server generates the image and sends its URL back to the ASP.NET script. The script then includes the URL in the page displayed in the `map` frame. The new map shows an orthographic photo image in the background as shown in **Fig. 4**. The layer list shows the Ortho Photo layer when the map scale becomes smaller than 1:200000.

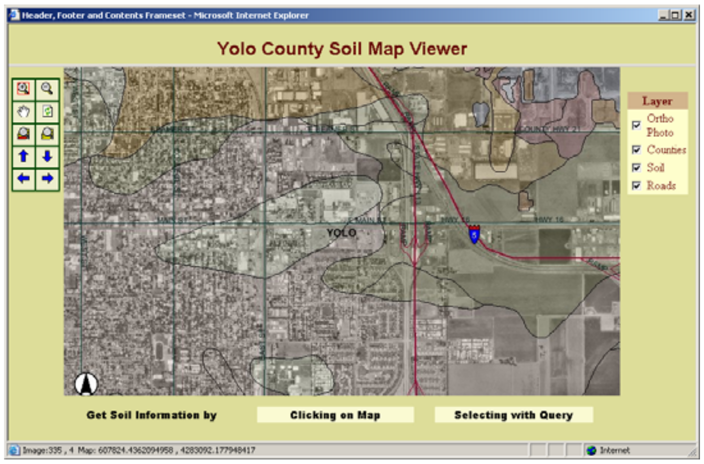

Header, Footer and Contents Frameset - Microsoft Internet Explorer
Yolo County Soil Map Viewer
YOLO
Layer
Ortho Photo
Counties
Soil
Roads
Get Soil Information by
Clicking on Map
Selecting with Query
Image:335 , 4 Map: 607824.4362094958 , 4283092.177948417
Internet

Fig. 4: Map with the orthographic photo layer.

4 WEBGD CUSTOM SERVER CONTROLS

We developed a set of custom server controls to be used as map-interface components. In this section, we show how those server controls can simplify development of an interactive map interface.

Developing a Map Interface with Custom Server Controls

We designed and implemented a set of custom server controls, including `ArcImsMap`, `LayerList`, and `MapToolbar`, for rapid implementation of an interactive map interface. Instances of these server controls can be added to an ASP.NET `.`*aspx* file with XML tags. Development enviroments like Visual Studio .NET allow these server controls to be picked and placed on the Web Form. T appearance and behavior of a server control can be managed by setting property values. The map interface shown in **Fig. 5** was created using WebGD server controls. This figure also shows the `ArcImsMap`, `LayerList`, and `MapToolbar` server controls.

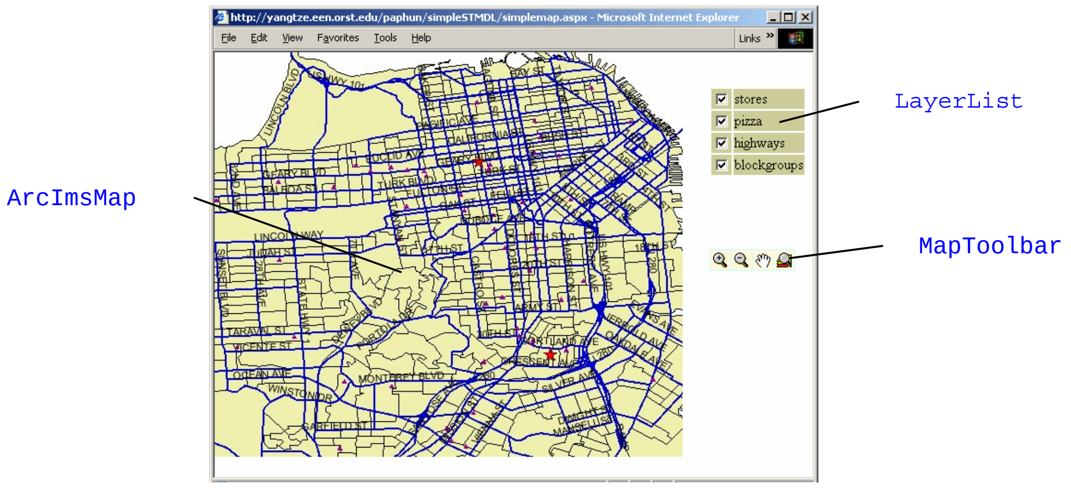

http://yangtze.een.orst.edu/paphun/simpleSTMDL/simplemap.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
stores
pizza
highways
blockgroups
LayerList
ArcImsMap
MapToolbar

Fig. 5: Interactive map interface generated by WebGD.

`Control` is the base class of all the ASP.NET server controls. `Control` defines events, properties, and functions common to every server control. In each phase of the lifecycle of a server control, the virtual function provided for that phase of the server control is called. To define its own appearance and behavior, a custom control can override virtual functions inherited from `Control`. For example, each of our custom server controls overrides the `Render()` method to define its appearance.

Our custom server controls are derived from the `WebControl` class, and hence the properties of `WebControl` are inherited by WebGD server controls. `WebControl` provides the properties common to every Web server controls, and we can change the appearance and the behavior of a control by setting new values to its properties. For example, we can set a new value to the `BackColor` property to change the background color of the control. In this section, we describe implementation details of each of our custom server controls.

ArcImsMap

An instance of `ArcImsMap` is used to render the map. As such, `ArcImsMap` forms the core of a WebGD map interface. The user can perform various actions, such as zooming, panning, and querying, with a map interface. Actions on a map image are handled by the

ArcImsMap. We encapsulate functionality of each action in a subclass of MapAction and store instances of MapAction subclasses in the Actions collection of an ArcImsMap. When an ArcImsMap receives a request posted back from the Web browser, it searches through the Actions collection for the MapAction instance that matches the action posted back. If the matching MapAction is found, its function Act() is invoked. In addition to the server-side function, a MapAction subclass may include also a client-side JavaScript script, which allows the user to interact with the map. For example, a ZoomInAction provides the client-side script for drawing a box for zooming.

We implemented actions with classes instead of implementing them as functions of ArcImsMap, since in this way we can add an action to an ArcImsMap or remove an action without modifying the ArcImsMap class. To implement a map action, such as highlighting areas in the map being displayed, we can create a new MapAction subclass and add its instance to the Actions collection of the ArcImsMap. Thus, functionality of a WebGD map interface is not limited to the classes currently provided by our framework. In addition, as a map action is encapsulated in a class, we can reuse it in multiple applications.

As shown in **Fig. 6**, the following abstract functions are defined in MapAction.

- GetAction(): This function returns the string representation of the action. For example, GetAction() of a ZoomInAction returns the value of “zoomin”. In order to determine the action requested by the user, the ArcImsMap uses the returned value of GetAction() to compare with the string posted back in the hidden action field.
- Act(): This function is called by the ArcImsMap to perform the action requested. For example, Act() of a ZoomInAction changes the extent of the map to the extent selected by the user. A Boolean value indicates whether or not the map scale has been changed. If the scale has changed and if the LayerList is in another page, then the ArcImsMap generates the client-side JavaScript script to post back the page containing the LayerList so that the new layer list is rendered.
- GetClientMouseHandlerName(): The ArcImsMap calls this function when it generates the switch statement in the client-side function setTool() to register the mouse-down event handler provided for the action selected. The function GetClientMouseHandlerName() returns the name of the mouse-down event handler for the action supported by this MapAction object.
- GetClientMouseHandlerMethod(): This returns the client-side script of the mouse-down handler. The client-side script is in JavaScript. The ArcImsMap registers the script returned from this function to the Web Forms page. As some actions, such as zoom-in and zoom-out, share a client-side script, the ArcImsMap uses the name of the handler as the key to prevent the same script to be registered redundantly for the same page.

```
public abstract class MapAction {
    public abstract string GetAction();
    public abstract string GetClientMouseHandlerName();
    public abstract string GetClientMouseHandlerMethod();
    public abstract bool Act( ArcImsMap pMap );
}
```

Fig. 6: Class `MapAction`.

MapToolbar

A `MapToolbar` can contain several tools for use on a map. A `MapToolbar` lays out tools in a table, storing each tool in a cell. Tools are represented by the images like those in the `tools` frame within the map interface given in **Fig. 7**. In addition, tools can be represented by texts as done in the `tools2` frame.

Some map operations, such as zooming, require the user to interact on the map image. When the user selects the zoom-in tool, the tool button changes its background color to indicate that it is currently selected. When another tool is selected, the original background color is restored. If a tool is represented by an image button then the image can be changed to another image to indicate that the tool is selected.

A `MapToolbar` is laid out as a table, and the `ArcIMSMap` control can have several `MapToolbars` associated with it. A developer can arrange tools and a map image according to her preference. **Fig. 7** shows another map interface with four `MapToolbars`, each of which has one tool for panning the map in a different direction.

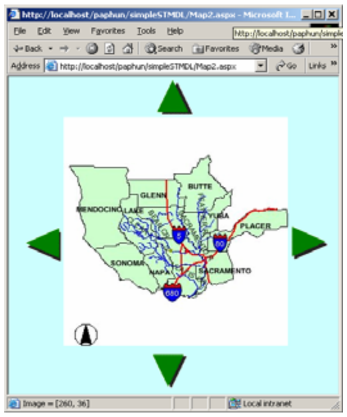

http://localhost/paphun/simpleSTMDL/Map2.aspx - Microsoft I...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/paphun/simpleSTMDL/Map2.aspx
Go Links
MENDOCINO
LAKE
GLENN
BUTTE
YUBA
PLACER
SONOMA
NAPA
SACRAMENTO
5
80
680
Image = [260, 36]
Local intranet

Fig. 7: Map interface with four **`MapToolbars`**.

Like an `ArcImsMap`, a `MapToolbar` has a collection of `Tools`. Each object in the collection is an instance of a subclass of the `Tool` class. A `MapToolbar` has the attribute `[…, ParseChildren(true, "Tools"), …]`, which tells the page parser to create objects from nested elements and to add them to the `Tools` property of the `MapToolbar`.

The `Tool` class contains data specifying the appearance of the tool and the behavior of the map operation represented by the tool. The behavior can be defined with four member variables of `Tool`.

- `Action`**:** This variable stores an action string to be set to the hidden `action` field of an `ArcImsMap`.
- `isMapAction`**:** This variable indicates whether or not the user needs to interact with the map to perform the operation. If `isMapAction` is true when the tool is selected, the tool changes the background color of the image, and the client-side function `setTool()` of `ArcImsMap` registers a mouse handler associated with the operation.
- `needNewImage`**:** This variable specifies whether or not the operation requires the new image. If the new image is needed, the form of the `ArcImsMap` is submitted to the Web server.
- `funcName`**:** This variable stores the name of a client-side function of the tool. While most operations that require new map images do not have a client-side functions, those operations that do not need new images need client-side functions. For example, a tool for retrieving information uses a client-side function to open a Web page to display the requested information.

We can get the values of the above four variables, but we cannot change their values. A `MapToolbar` renders HTML elements with attribute `onclick` associated with the `setTool()` function call made with arguments `action`, `isMapAction`, `needNewImage`, and `funcName`.

`Tool` also has other variables for specifying the appearance of a tool. Examples of such variables are the `showImg` variable that defines whether or not the tool is an image button and the `enable` variable that defines whether the tool should be initially enabled or disabled. The `Tool` class provides the public properties for these variables. Hence, values of these variables can be specified as attributes in declarative tags, as shown in **Fig. 8**. As the behavior of the operation represented by a tool can be defined with its member variables, we created `CustomTool`, which exposes some member variables as properties. Thus, we can define a tool for an operation by specifying the values of `CustomTool` properties.

```
<ARCIMS:MapToolbar id="MapToolbar1" runat="server"
 BackColor="#ccffff" mapid="myMap" numcols="4">
       <ARCIMS:ZoomInTool ShowImg="true"/>
       <ARCIMS:ZoomOutTool ShowImg="true"/>
       <ARCIMS:PanTool ShowImg="true"/>
       <ARCIMS:ZoomActiveTool ShowImg="true"/>
</ARCIMS:MapToolbar>
```

Fig. 8: Declarative tags for a `MapToolbar`.

LayerList

The `LayerList` displays the list of the layers currently shown on the map, and it also allows the user to change visibility of the layers and to select the active layer. In this section, we describe the user interface of a `LayerList` and explain how different types of layers are handled by `LayerList`.

The `LayerList` shows the list of the layers displayed on the `ArcImsMap`. Each layer occupies one row in the table provided for the `LayerList`, and each row has two columns for the name and the visibility checkbox. The user can make a layer visible or invisible with this checkbox. When the user checks or unchecks the checkbox, the client-side JavaScript code generated by the instance of `LayerList` submits the Web Forms page containing the state of the `ArcImsMap` to the server for a new map image. The table cell displaying the name of a layer has a JavaScript function registered with the `click` event. When the user clicks on the name of a layer, the `LayerList` highlights the layer to indicate that it is the active layer. A map operation, like *zoom-to-active-layer*, uses information on the active layer when the action is performed on the map.

In the load phase, a `LayerList` steps through the layer list of the ArcIMS `Map` object, creating a `LayerItem` for each layer and adding it to the `Layers` array. Then, in the render phase, the `LayerList` renders the `LayerItems` in the `Layers` collection. The declaration of `LayerItem` is shown in **Fig. 9**. We add the `Layers` array to `LayerList` to allow modification of the property values of layers before they are used in the render phase.

```
public sealed class LayerItem {
  private string id;
  private bool visible; //whether the layer is shown on map
  private LayerType type; //feature, image or clone
  public string Name; //layer name
  public bool Active; //whether the layer can be active
  public bool Show;//whether the layer is shown on list
  public string ID { get{ return id;}}
  public bool Visible {get{ return visible;}}
  public LayerType Type {get {return type;}}
  public LayerItem (string pName, string pId, bool pVisible,
                    bool pActive, LayerType pType )
  {
    Name = pName;
    id = pId;
    Active = pActive;
    visible = pVisible;
    type = pType;
    Show = type == Clone ? false : true;
  }
}
```

Fig. 9: Class `LayerItem`.

A `LayerList` handles layers differently, depending on their types. ArcIMS supports three types of layers: Feature, Image, and Acetate. In addition to those layer types, an instance of `LayerList` may contain a clone of a Feature layer.

- **Feature layer:** A Feature layer represents a vector dataset. ArcIMS allows us to get or set many properties of a Feature layer. For example, we can get the extent of the dataset, and we can apply a filter to the features in the layer. A Feature layer can be selected as the active layer on a `LayerList`.
- **Image layer:** An Image layer displays raster data on the map. For an Image layer, we can only read the values of the values, such as `name`, `minimum scale`, and `maximum scale`. As an ArcIMS server does not provide any useful function on an image layer, it cannot be selected as the active layer.
- **Acetate layer:** An Acetate layer is a custom layer to which we can add objects, such as a scale bar, a north arrow, and polygons. As an Acetate layer does not represent any dataset, a `LayerList` does not add it to the `Layers` collection.
- **Cloned layer:** A cloned layer is a copy of the Feature layer. We can use a cloned layer to implement such a feature as highlighting. To highlight geographical features, we create a new layer as a clone of the layer to be highlighted, keeping only the features that need be highlighted, and change the symbol of the cloned layer. By default a cloned layer will not be shown in the `LayerList`.

Supported Web Browsers

As described earlier, client-side JavaScript code is used in many functions. These client-side scripts provide functions that cannot be performed with static HTML. However, client-side scripts cause browser compatibility problems. We developed and tested our custom server controls to support the following major Web browsers: Microsoft Internet Explorer 6.0, Netscape Navigator 6.2, and Mozilla 1.1.

5 WEBGD APPLICATION: MOTELS IN OREGON

To illustrate the utility of WebGD, we examine *Motels in Oregon*, an example business application that allows customers to search for information and make reservations for motels in Oregon. With a standard Web browser, users can perform various operations related to motel business processes. **Fig. 10** shows the interface for *Motels in Oregon*, which, in this case, displays the State of Oregon.

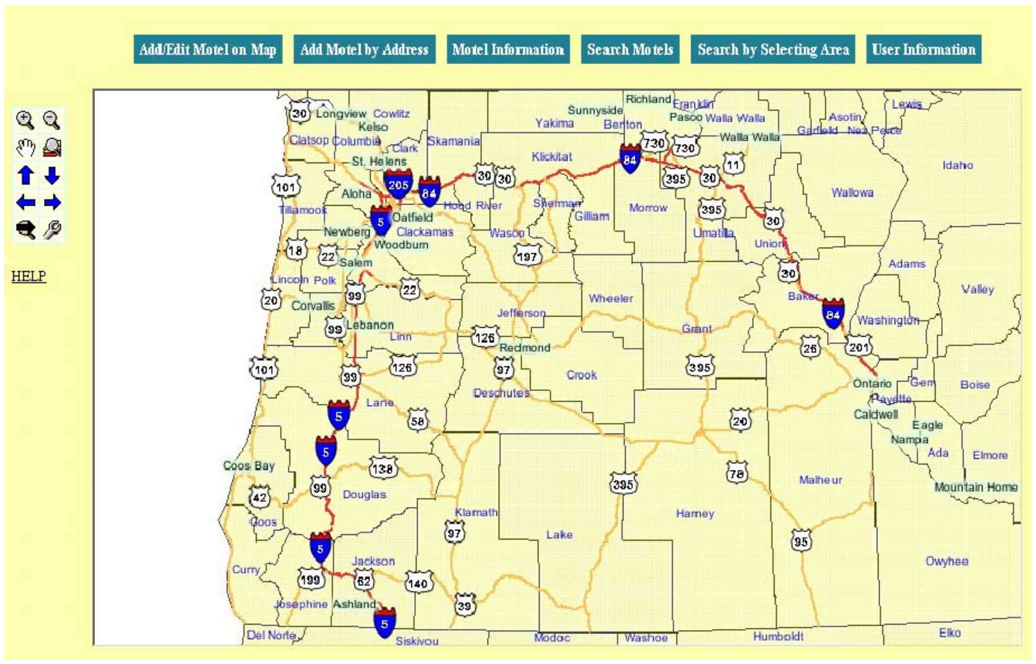


Fig. 10: Motels in Oregon

Motels in Oregon allows users browse the map, by panning the map and zoomming in or out. When zoomed in far enough, the Ortho Image layer becomes visible, giving users an aerial view of the map. The result in **Fig. 11** shows a section of Corvallis, OR in which the Super 8 Motel and Econo Lodge are highlighted.

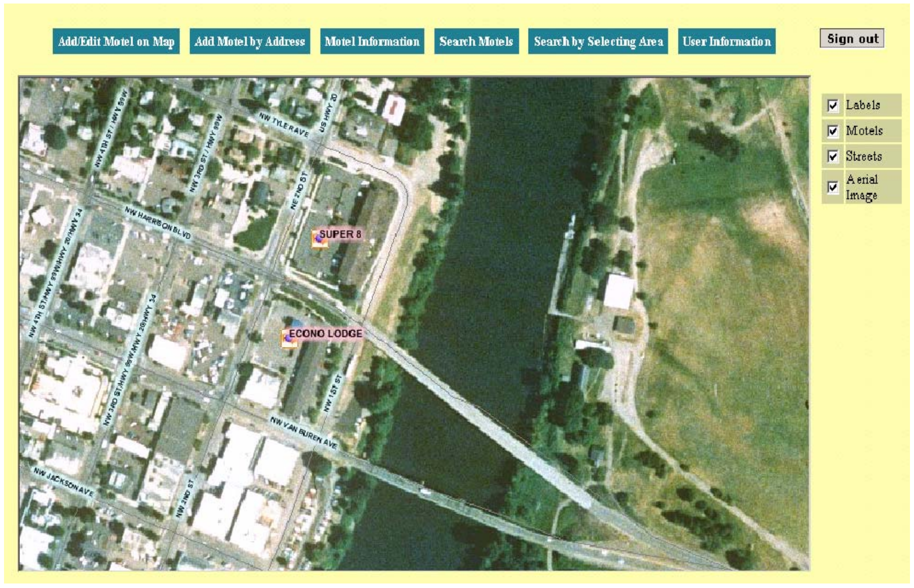
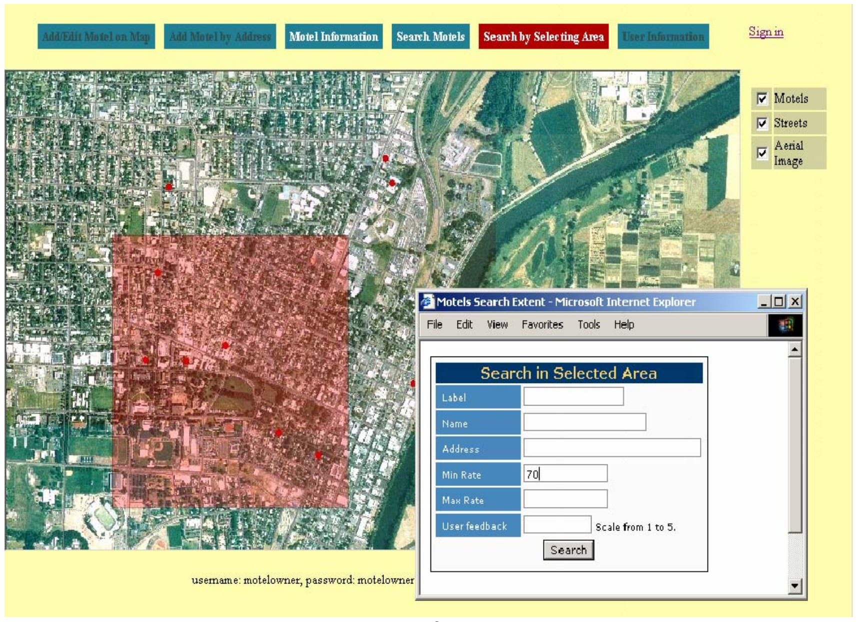


Fig. 11: Zoomed in to Corvallis, OR.

Motels in Oregon has four levels of authorization: the public, customers, motel owners, and administrators. The public can browse the map to locate motels and retrieve information. When members of the public register, they become customers, who can, in addition to searching and information retrieval, set and modify reservations. Motels can be searched for by address or by selecting an area, as shown in **Fig. 12**

.

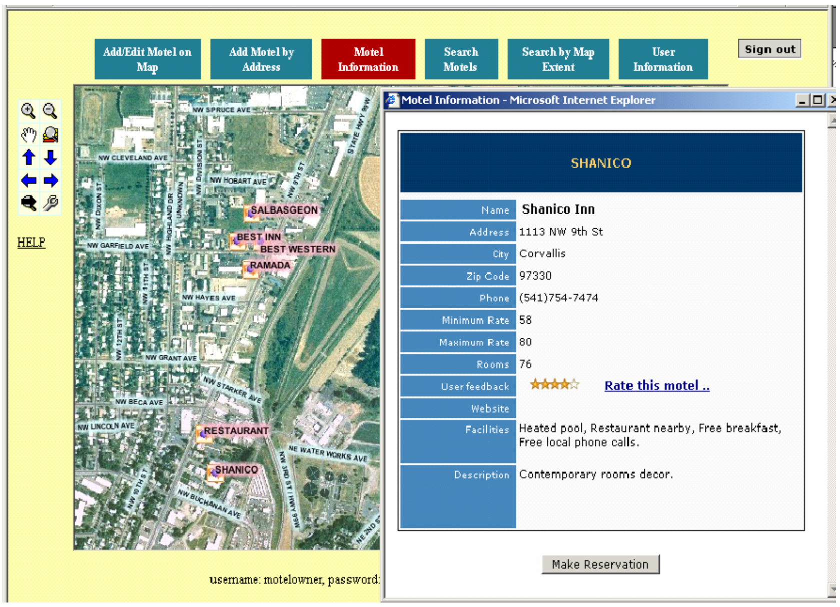


Min Rate	70
User feedback	Scale from 1 to 5.

username: motelowner, password: motelowner

Fig. 12: Searching for motels by area.

Once a motel has been found, users can retrieve relevant information such as room rates, number of rooms, and other amenities. This is done by simply clicking on the motel on the map. The results of such a query are depicted in **Fig. 13**.

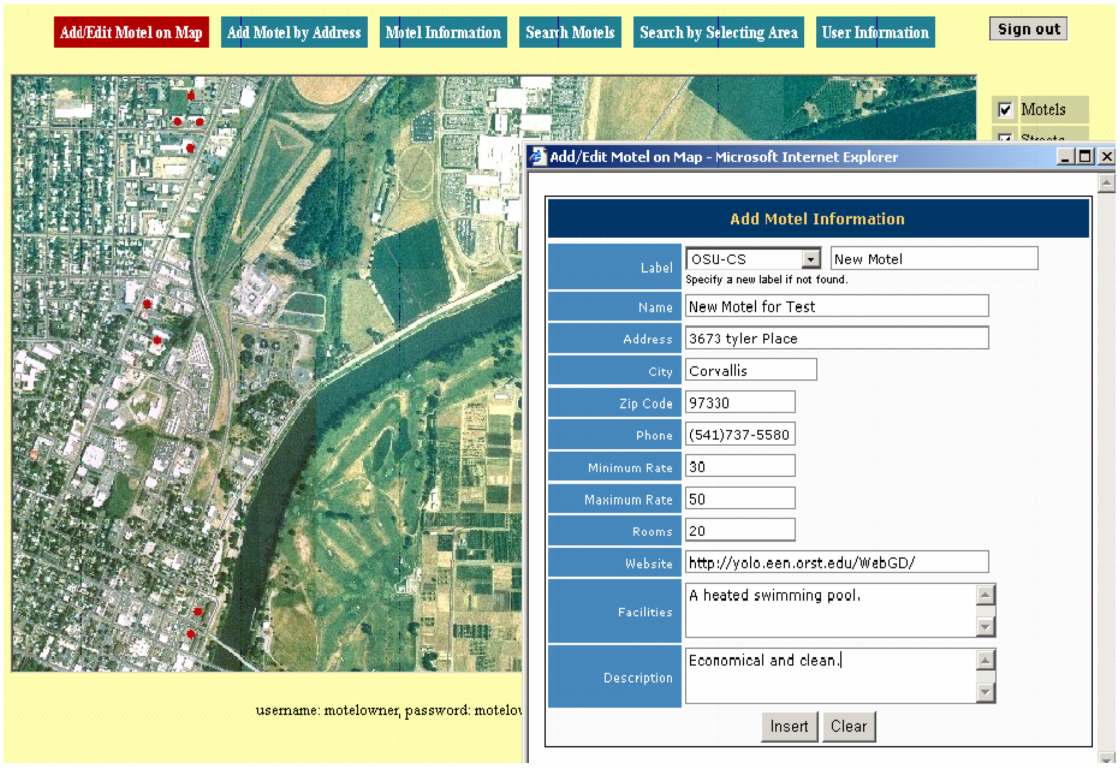


SHANICO

Name	Shanico Inn
Address	1113 NW 9th St
City	Corvallis
Zip Code	97330
Phone	(541)754-7474
Minimum Rate	58
Maximum Rate	80
Rooms	76
User feedback	★★★★☆ Rate this motel ..
Website	
Facilities	Heated pool, Restaurant nearby, Free breakfast, Free local phone calls.
Description	Contemporary rooms decor.

Fig. 13: Retrieving motel information.

Motel owners utilize the full power of WebGD, as they can insert and delete motel symbols on the map, and insert, retrieve, update, and delete non-spatial data associated with each motel. **Fig. 14** gives a view of the interface for inserting a motel.



Add Motel Information

Label	OSU-CS New Motel Specify a new label if not found.
Name	New Motel for Test
Address	3673 tyler Place
City	Corvallis
Zip Code	97330
Phone	(541)737-5580
Minimum Rate	30
Maximum Rate	50
Rooms	20
Website	http://yolo.een.orst.edu/WebGD/
Facilities	A heated swimming pool.
Description	Economical and clean.

username: motelowner, password: motelo

Fig. 14: Inserting a motel.

Motels in Oregon provides a powerful interface for motel operators and customers. This application offers motel operators both a means of advertisement and sales, while providing customers with a convenient platform for locating motels and reserving rooms.

6 CONCLUSIONS AND FUTURE WORK

WebGD is a framework for developing Web-based GIS/database applications. This framework makes possible rapid development of a Web-based application that supports the insertion and deletion of geographical features shown on the interactive map interface, as well as the access and retrieval of information associated with those geographical features.

The building blocks of WebGD are ASP.NET custom server controls. We have observed the following benefits of the WebGD custom server controls.

1. WebGD custom server controls enable us to create a Web-based GIS application rapidly. These instances can be added to an ASP.NET page as declarative elements. In addition, a development tool, such as Visual Studio .NET, allows us to pick and place instances of those server controls on a Web Forms page.
2. The appearance and behavior of a custom server control can be customized for each application. We can set the values of server control properties in declarative

tags as attribute values or change those values in ASP.NET scripts with executable statements.

3. We can implement new map actions and new tools, and add their instances to our custom server controls.

WebGD has been used to develop several Web-based applications that have map interfaces. *Yolo County Soil Viewer* and *Motels in Oregon* are two example applications discussed in this paper.

We are also developing WebSiteGen, a tool that uses the schema of a database to automatically generate a set of Web forms for accessing data stored in the database. We will combine WebGD with WebSiteGen so that Web-based applications including both map interfaces and data forms can be created with little up-front programming. The information in the configuration file of a map service can be used to relate geographical features displayed on the map interface to the business data associated with them. In this way, we are creating tools and a framework that can be used to automatically create Web-based GIS/database applications.

REFERENCES

[Challa02] Challa Siva and Artur Laksberg: *Essential Guide to Managed Extensions for C++*, Apress, 2002.

[Duthie02] G. Andrew Duthie and Matthew MacDonald: *ASP.NET in a Nutshell*, O'Reilly, 2002.

[ESRI02] ArcObjects Developer Help, ESRI, 2002.

[ESRI02] ArcXML Programmer's Reference Guide, ESRI, 2002.

[Microsoft02] *MSDN Library*, Microsoft, 2002.

About the authors

Paphun Wangmutitakul is a software engineer at Terra Genesis, Inc., AZ. She currently works on developing GIS application programs for transportation system maintenance and analysis.

Dr. Toshimi Minoura is an associate professor of Computer Science at Oregon State University. His research group has designed and implemented frameworks for creating web-based GIS/database applications, by using ArcIMS and ArcSDE on Windows and by using MapServer on Linux. With this technology, the group has created applications for environmental impact assessment and wildlife conservation.

Alec Maki is a graduate student in computer science at Oregon State University. He studies under Dr. Minoura's guidance. His research focus is on the design and development of web-based GIS/database applications.