OSGeo Journal

The Journal of the Open Source Geospatial Foundation

Volume 3 / December 2007

2007 FREE AND OPEN SOURCE SOFTWARE
FOR GEOSPATIAL (FOSS4G) CONFERENCE
VICTORIA CANADA SEPTEMBER 24 TO 27, 2007

Proceedings of FOSS4G 2007

Automatic Generation of Web-Based GIS/Database Applications

Nirut Chalainanont, Junya Sano and Toshimi Minoura

Abstract

We have been developing *web-based GIS/database* (WebGD) applications that allow users to *insert*, *query*, *update*, and *delete geographical features* and the data associated with them from standard Web browsers. The code shared by these applications is organized as the WebGD framework. The behavior of the map interface of a WebGD application is defined by *configuration files*. We have built also the WebGD application generator (WebGD-Gen) that *automatically* produces those configuration files from the database metadata, including those in tables `geometry_columns` and `spatial_ref_sys`. WebGD-Gen can generate also Web scripts that interact with the map interface and the database. Thus the WebGD framework and WebGD-Gen can significantly reduce the development time and the maintenance cost of a complex Web-based GIS/database application.

Introduction

The Internet has become the major venue for sharing information. A dynamic Web-based mapping application allows users without desktop-based GIS software to perform *spatial queries* and produce maps with standard Web browsers.

However, a typical web-based GIS application allows only the retrieval of maps and map-related data. A web server provides information to the client, but the client cannot feed information back to the server (3). This *unidirectional* flow of information is a major problem with a current typical map-server application. Furthermore, creating an interactive web application with a map interface is time-consuming. Commercial map servers and geographical database management systems are expensive. This situation hinders use of Web-based GIS applications in data gathering, analysis and decision-making.

We have been developing a set of tools that significantly reduces the cost of application development (14; 16; 9; 15). The code shared by interactive Internet GIS applications is organized as the Web-based GIS/database (WebGD) framework. Most of the complex workings for delivering GIS functions over the Web are included in this framework. With the WebGD framework, we can create a map interface through which users can *insert*, *query*, and *delete* geographical features and the data associated with them only by providing *configuration files*.

An important feature of a WebGD application is that it tightly integrates spatial data and associated tabular data, enabling analyses involving location-based data (13). These functions are available to users at different geographical locations for economical and timely data management.

We developed several years ago a *Web-form generator* that automatically generates scripts for traditional Web-forms from the *schema* of a relational database (4). This functionality was recently extended as the WebGD application generator (WebGD-Gen). The new application generator can produce most of the code for an entire WebGD application from the schema of a relational database and the information on *geometry columns*. With the map interface and the Web scripts automatically generated, geographical features (i.e., points, linestrings and polygons) can be *inserted*, *queried*, and *deleted*. Thus, when a relational schema and the GIS metadata for the map layers are available, a *non-customized* application can be quickly assembled with the WebGD framework and the WebGD-Gen application generator.

The automatic generation of the map interface and Web-form scripts make possible *incremental* and *iterative* development of complex web-based GIS applications. When a map layer is added or modified, we only need to create or update the configuration file for that layer and then regenerate the Web forms for that layer. When a database schema is modified, we can regenerate the entire set of Web scripts in several minutes. Therefore, even with an incomplete set of map layers and a database schema, we can generate a working prototype for an initial review. Furthermore, we can fix most of the software bugs by modifying only the shared code in the WebGD framework or WebGD-Gen and then by regenerating Web scripts for each application.

WebGD applications use the following *open-source* software components. PostgreSQL, an *object-relational* database, and PostGIS together manage geospatial data. PostGIS is an extension of PostgreSQL for GIS applications (11). MapServer (12) generates maps to be displayed on a web browser by using geospatial data provided by PostGIS. Web pages, including the one that displays the maps, are generated by server-side scripts written in PHP. The PHP MapScript module interacts with MapServer (6; 2). When a request to insert or delete a map feature is received by a PHP script, the script directly accesses the PostgreSQL database, using the PostGIS extension. An application developed runs on a PC without any licensed software.

We explain the features supported by the WebGD framework in Section **WebGD Applications**. The process of generating map-layer configuration files is explained in Section **WebGD Framework**, and that of generating Web scripts in Section 4. The process of automatic generation of map-layer configuration files is explained in Section **Automatic Generation of Map-Layer Configuration Files**. In Section **WebGD Development History**, we describe a brief history of the development of the WebGD framework and WebGD-Gen. Section **Conclusions and Future-Work** concludes this paper.

WebGD Applications

The Web interface of one of the WebGD applications, Natural Heritage Information System (NHIS) for North Carolina, is shown in Figure 1. This application provides a map interface for a copy of the Biotics 4.0 database maintained by the North Carolina Natural Heritage Program. Biotics 4.0 is a desktop GIS application built on the database developed by NatureServe. The key elements in this database are *element occurrences* (EOs), which are *areas* of land and/or water in which species are, or were present (7). EO records have both *spatial* and *tabular* data, and the database contain approximately 700 relational tables (5). The Biotics Mapper implemented with ArcView by NatureServe provides a map interface that allows EO representations and associated data to be created, updated, and deleted (8). In our implementation, we can perform these operations with standard Web browsers. Also, Web forms, approximately 3500 in total, are provided for all tables in the database.

The NHIS application enables *bi-directional* movement of *geospatial data* as well as ordinary data. Scientists and others with proper authentication can *insert*, *query*, and *delete* geographical features such as EO polygons, lines, and points, as well as the data associated with them. Queries can be executed by spatially *selecting an area* on the map or by using a traditional web form. In addition, one-meter resolution *digital orthographic quadrangles* DOQ, or aerial images, are included as a layer. When DOQ images are combined with other map layers such as highways, county boundaries, streams, and streets, locations can be easily pinpointed by taking advantage of features between map layers (16).

The major operations supported by the map interface of a WebGD application are as follows:

1. To retrieve information on the geographical features in the area of interest, the user can zoom in/out to that area by using the map navigation tools. If the user zoom-in enough, one-meter resolution aerial photos are displayed. The user can also go to a new area by selecting an entry in the **Quick View** menu.
2. To get information about a geographical feature, the user can select a layer in the legend and **Information** in the function menu, and then click the boundary of the feature.
3. Function **Insert** allows a geographical feature to be added with mouse clicks on the map. **Done** need be pressed after all points are entered.
4. Function **Search by Area** allows the user to retrieve the list of features that are within a *bounding box* specified on the map and that satisfy a search condition. The features that satisfy the search condition are *highlighted* on the map. Furthermore, the user can select features in the list by marking the checkboxes associated with them. Then, if the map is refreshed, the selected features are highlighted.
5. The data administration interface can be activated by clicking on the **Database** entry in the menu bar below the banner. A tree icon can be clicked to display a *treeview* for browsing. The treeview for **Higher Taxonomy** is the major one. To access the data of this application, a user must login with a password as some data on endangered species are confidential.

Several WebGD applications created with the WebGD framework and WebGD-Gen can be accessed from this page.[3]

[3]WebGD sample applications: `http://yukon.een.orst.edu/index_webgd.html` You may insert/query/update/delete data. However, please DO NOT disturb existing data. If login is required, use user name `cs540` and password `CSxyz540`. You may have to login twice, once for the server and then for an application.

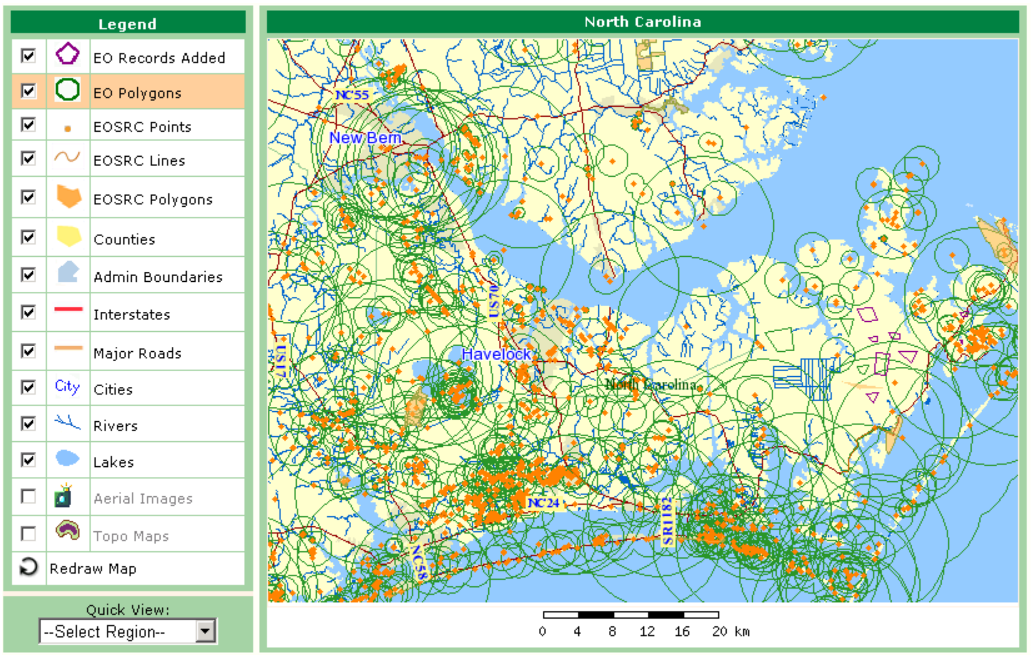


Figure 1: Interface of the NHIS Application for North Carolina

The first application is Natural Heritage Information System. Although this application can cover the whole USA or the world, the data are currently available only for North Carolina. The second application provides a map interface for an application that keeps track of conservation practices on land parcels. The third one is a Web-based mapping application for plant germplasm collection maintained at Western Regional Plant Introduction Station (USDA-ARS). The fourth one allows the soil information at the location where a mouse click occurs on the map interface to be retrieved.

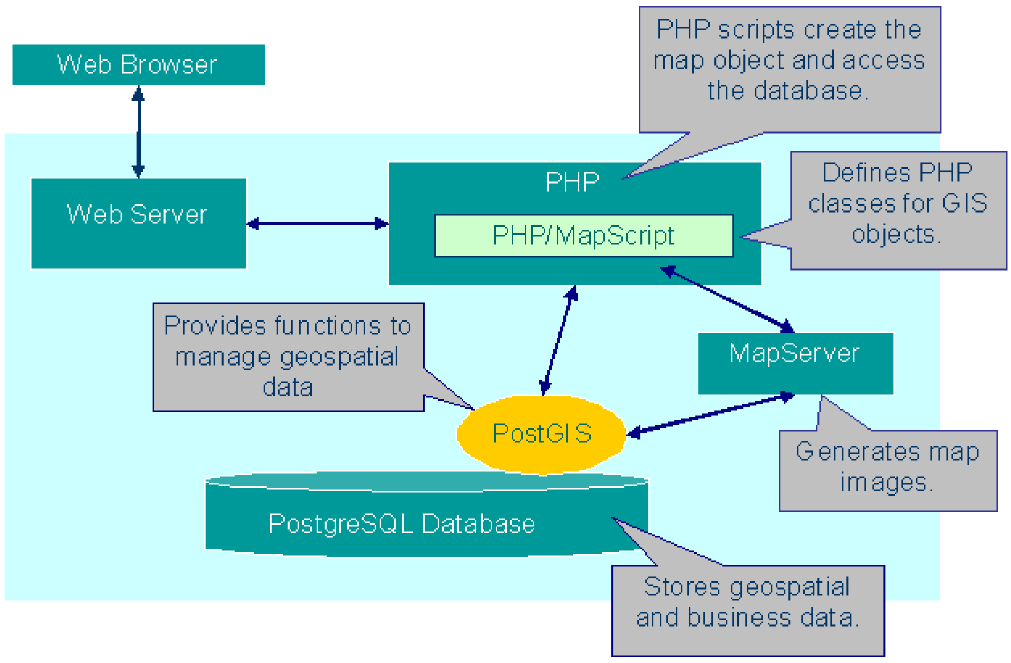
One salient feature of the current WebGD framework is *dynamic switching of spatial references*. Typically, different geographic regions and localities have preferred *map projections* in order to avoid distortions in the maps created (1). The framework allows the whole world to be covered with multiple-levels of maps, e.g., the world map, continent maps, and region maps. The map interface then automatically selects the most suitable projection for the region whose portion is displayed. For example, the world can use the *geographical coordinate system*, the United States the *Albers equal-area projection*, and Oregon the *Lambert conformal conic projection*. Thus, spatial analysis can be performed with the most appropriate projection for a particular area. The *dynamic switching* of the *spatial reference*, the *map file*, the *legend*, and the *quick view menu* supported by the current WebGD framework allows any part of the world to be covered with its own scale and spatial reference, including regions with one-meter resolution aerial images. This is a very important feature, especially now that the cost of storing aerial images for the entire US has dropped to affordable levels (10 terabytes needed to store aerial images for the entire US now cost around $10,000). Furthermore, many states are putting aerial images in the public domain.

WebGD Framework

The WebGD framework supports common features required by the map interface of a WebGD application such as zoom in/out, pan, and insert/query/update/delete operations of geographical features. The organization of a WebGD application is shown in Figure 2.

A map operation that does not manipulate geometry features, e.g., *zoom-in*, *zoom-out*, or *panning*, is processed as follows:

1. The user action on the map is transmitted from the Web browser to the Web server as an HTTP GET request.
2. The Web server activates a PHP script that handles the user action.
3. Inside the PHP script, various methods in PHP MapScript are called to prepare the new map



Web Browser
Web Server
PHP
PHP/MapScript
PHP scripts create the map object and access the database.
Defines PHP classes for GIS objects.
Provides functions to manage geospatial data
MapServer
PostGIS
Generates map images.
PostgreSQL Database
Stores geospatial and business data.

Figure 2: Organization of a WebGD Application

parameters, such as the special reference and the extent of the map and the names of the layers to be displayed. The map drawing method in PHP MapScript is then called to instruct the MapServer to create the map image.

4. The MapServer requests data for the map layers whose data are stored in the PostgreSQL database.
5. The data for the layers are returned.
6. The map image created is returned to the PHP script.
7. The HTML page generated by the PHP script is returned to the Web server.
8. The Web server transmits the HTML page including the new map image to the Web browser.

For a map operation that accesses or manipulates geometry features, i.e., *search by area*, *insert*, or *move point*, is processed as explained above, except between steps 2 and 3 the following operations are performed.

a The PHP script connects to the PostgreSQL database to perform a spatial operation with PostGIS-enabled SQL statements.

b The result of the spatial operation is returned to the PHP script by the SQL statements.

Furthermore, *configuration files* are user extensively as part of the WebGD framework for application customization. Major configuration files are *region configuration files*, *map layer configuration files*, and a *quick view configuration file*.

Region Configuration Files

In order to minimize map distortion, different map areas can be displayed with different spatial references. The *current region* is determined to be the *smallest* one that encompass the extent of the map to be generated. The *spatial reference* is then automatically switched to that of the new region. Then the layers in the map legend, the list of the map-navigation and data-manipulation commands, and the *quick view* list are reconfigured for the new region. These reconfigurations are necessary because different regions may require different sets of map layers, command, and quick view list.

A *region configuration file* for each map region includes the following definitions:

1. the special reference for the region,
2. the *name of the region* displayed on the map interface,
3. the *unit of distance measurement*,
4. the *name of the map layer configuration file* for the region, and
5. the *name of the quick view configuration file*.

For example, the region configuration file for the world is as follows:

```
$region = array(
   "gid" => 1,
   "name" => "world",
   "display_name => "World",
   "srid" => 4326,
   "rank" => 1,
   "units" => "MS_DD",
   "mapfile" => "gmap75_world.map",
   "quickview" => "world_gview.php",
   "legend" => "world_maplayers.php",
   "proj4text" => "+proj=longlat +ellps=WGS84
         +datum=WGS84"
);
```

The following options can be specified in a configuration file:

gid – the unique number assigned to each region, which is the primary key value of the row representing the region in table `regions` in the database.
name – the name of the region for programming.
display_name – the name of the region used on the map interface. This name may include capital letters, white spaces, and other punctuation marks.
srid – the *spatial reference identifier* for the map projection assigned with this region.
rank – the priority number used in determining the region for the map area to be viewed. The region with the highest rank number is selected among the regions that completely encompass the map area to be viewed. Regions with higher rank numbers cover smaller areas.
units – the unit of measurement associated with the region.
mapfile – the map file to be loaded when this region is selected.
quickview – the name of the quick view configuration file for the region.
legend – the name of the map layer configuration file for the region.
proj4text – the projection string used for the region.

Whenever the map region changes due to a user action on the map interface, the region configuration file for the new region is loaded dynamically. Then the map interface is customized according to the new region configuration file.

The region configuration file for Oregon, for example, contains the following definitions:

```
$region = array(
   "gid" => 150137,
   "name" => "oregon",
   "display_name" => "Oregon",
   "srid" => 6010,
   "rank" => 150137,
   "units" => "MS_FEET",
   "mapfile" => "gmap75_oregon.map",
   "quickview" => "oregon_gview.php",
   "legend" => "oregon_maplayers.php",
   "proj4text" => "+proj=lcc +lat_1=43.0
         +lat_2=45.5 +lat_0=41.75
         +lon_0=-120.5
         +x_0=400000.00000 +y_0=0.0"
);
```

Map Layer Configuration Files

The *map layer configuration file* provided for a region specifies the layers to be included in the legend and their characteristics. It is possible for multiple map regions to share one map layer configuration file.

The map layer configuration file for the Oregon region, for example, contains the following definitions:

```
$layer_groups = array (
  'grp_eo_py' => array(
     'geom_type' => 'polygon',
     'table' => 'eo_py',
     'layer_selectable' => true,
     'gid_column' => 'gid',
     'geom_col' => 'the_geom',
     'legend_label' => 'EO Polygons',
     'search_script' =>
      'forms/eo/eo_py_eo_search.phtml',
     'select_script' =>
      'forms/eo/eo_py_eo_select.phtml',
     'edit_script' =>
      'forms/eo/eo_py_edit.phtml',
     'normal_layer' => 'eo_py',
     'searched_layer' => 'eo_py_searched',
     'checked_layer' => 'eo_py_checked',
     'selected_layer' => 'eo_py_selected',
     'img_src' => 'images/eo_poly.png',
     'img_width' => 26,
     'img_height' => 26,
     'onclick' => 'activate_layer("grp_eo_py")',
     'data_srid' => 32119
  ),
  'grp_eosrc_pt' => array(
     'geom_type' => 'point',
     'table' => 'eosrc_pt',
     'layer_selectable' => true,
     'gid_column' => 'gid',
     'geom_col' => 'the_geom',
     'legend_label' => 'EOSRC Points',
     'search_script' =>
      'forms/eo/eosrc_pt_search.phtml',
     'select_script' =>
      'forms/eo/eosrc_pt_select.phtml',
     'edit_script' =>
      'forms/eo/eosrc_pt_edit.phtml',
     'normal_layer' => 'eosrc_pt',
     'searched_layer' => 'eosrc_pt_searched',
     'checked_layer' => 'eosrc_pt_checked',
     'selected_layer' => 'eosrc_pt_selected',
    'img_width' => 5,
    'img_height' => 5,
    'onclick' => 'activate_layer("grp_eosrc_pt")',
    'data_srid' => 32119
  ),
...
```

According to the above configuration file, the map legend shown in Figure 3 is produced.

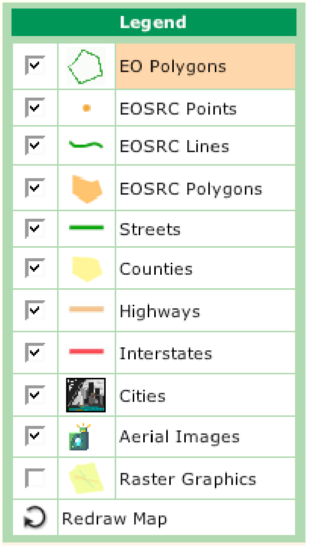


Figure 3: Map Legend of WebGD Application NHIS

We now explain each option used in map-layer configuration:

geom_type – the type of geometry features contained in the layer. The value can be `polygon`, `multipolygon`, `linestring`, `multilinestring`, `point` or `multipoint`. Exact spatial operations performed depend on this type. For example, if the type is `point`, a point is inserted on the map with an insert operation. On the other hand, if the type is `polygon`, a polygon feature can be inserted.

table – the name of the table in the database that contains the geometry column for the layer.

layer_selectable – the boolean option that determines whether spatial operations can be performed on the layer or not. Some layers, such as those for counties and highways in the above example, are static, and they are not selectable for spatial operations.

gid_column – the name of the column that contains the identifiers for the geometry features in the map layer.

geom_col – the name of the geometry column for the geometry features contained in the layer. The default name is `the_geom`.

legend_label – the name of the layer in the legend.

search_script – the name and the location of the search form associated with the layer.

select_script – the name and the location of the select form associated with the layer.

edit_script – the name and the location of the edit form associated with the layer.

normal_layer – the name of the map layer displayed when no highlighting occurs.

searched_layer, checked_layer, selected_layer – the names of the layers used to highlight the geometry features *searched for*, *checked*, or *selected*, respectively. The geometry features returned by a search operation is *searched for*, those whose checkboxes are turned on in search result form are *selected*, and the geometry feature chosen for editing is *selected*.

img_src, img_width, img_height – the file name, the width, and the height of the icon in the legend.

onclick – the name of the javascript event handler activated when the user selects the layer in the legend.

data_srid – the *spatial reference identifier (srid)* that designates the map projection used by the geometry features in the map layer. If this srid is different from that of the map region, then geometry features in the layer are reprojected before they are displayed on the map.

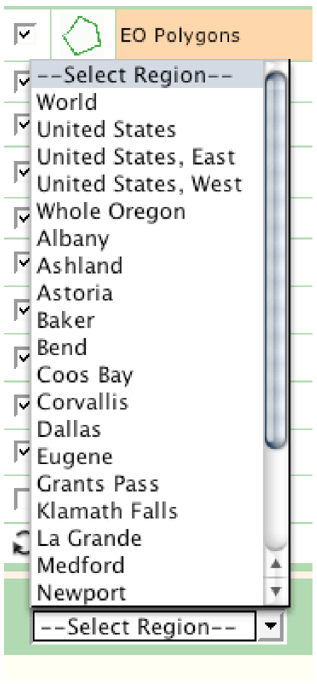
Quick View Configuration File

The *quick view* mechanism allows the user to select the map area of her interest. That is, the user can switch to a new map area quickly by selecting the map area from the quick-view dropdown list.

Each entry in a *quick view configuration file* describes the name of the map area represented by the entry, the map projection used by the *extent* of the map area, and the extent. The quick view configuration file for the Oregon region shown in Figure 4, for example, can be as follows:

```
$qview = array(
  array(
    'name' => 'World',
    'srid' => 4326,
    'extent' => '-180,-90,180,90'
  ),
  array(
    'name' => 'United States',
    'srid' => 4326,
    'extent' => '-125,13,-65,53'
  ),
  array(
    'name' => 'United States, East',
      'srid' => 4326,
      'extent' => '-102,22,-60,50'
    ),
    array(
      'name' => 'United States, West',
      'srid' => 4326,
      'extent' => '-135,30,-105,50'
    ),
    array(
      'name' => 'Whole Oregon',
      'srid' => 6010,
      'extent' => '46461.662375,-43912.968464,
        2487069.754184,1785176.331408',
    ),
...
```

The quick view list of regions as displayed in the map interface corresponding to the configuration file above:


EO Polygons

--Select Region--
World
United States
United States, East
United States, West
Whole Oregon
Albany
Ashland
Astoria
Baker
Bend
Coos Bay
Corvallis
Dallas
Eugene
Grants Pass
Klamath Falls
La Grande
Medford
Newport

--Select Region--

Figure 4: Quick View Selection of WebGD Application NHIS

Each entry in a quick view configuration file can specify the following options:

name – the name of the map area.

srid – the spatial reference identifier for the extent explained below.

extent – the `xmin`, `ymin`, `xmax`, and `ymax` values describing the positions of the lower left and upper right corners of the map area.

[4] 700 x 50

Automatic Generation of Web-Form Scripts

Several tools have been developed to augment the WebGD framework and simplify application development. The WebGD Web-site generator (WebGD-Gen) can create an entire WebGD application, including a web-based mapping interface. WebGD-Gen *automatically* generates a consistent set of Web scripts from *configuration files*, which are again automatically generated from a relational database schema. Since form generation is automatic, the cost of application development is greatly reduced. For a database such as Biotics that contains approximately 700 tables, programming all the required 3,500[4] forms manually can be very costly, even unfeasible.

WebGD-Gen is implemented as a collection of *templates*. Each template, combined with a corresponding *configuration file*, generates one of the following six types of Web scripts: *search*, *select*, *edit*, *information*, *action*, and *treeview* scripts. Templates and configuration files are written in PHP. The Web scripts generated by them are also in PHP. The generated scripts are executed on a Web server by a PHP interpreter. Each script, except for an action script, creates a Web form that is displayed on a client computer by a Web browser. Figure 5 illustrates the interactions among the Web scripts and forms.

Furthermore, WebGD-Gen can automatically generate the statements for inserting, searching, and deleting *geographical features* if the following lines, for example, are added to a configuration file:

```
 // type of geographical features
$web_gd = 'MULTIPOLYGON';
// layer group in legend
$layer_name = 'grp_eo_py';
// geometry column containing shapes
$geometry_column = 'the_geom';
//geographical feature IDs
$gid_column = 'gid';
// epsg spatial reference
$db_table_srid = 32119;
```

The forms generated for geographical features can perform the following additional functions compared to those for ordinary database tables:

1. A search form can be activated from a map interface. In this case, the extent of a search box specified on the map is passed as additional search parameters.
2. A select form includes additional JavaScript code for highlighting geographical features retrieved or selected by the user.

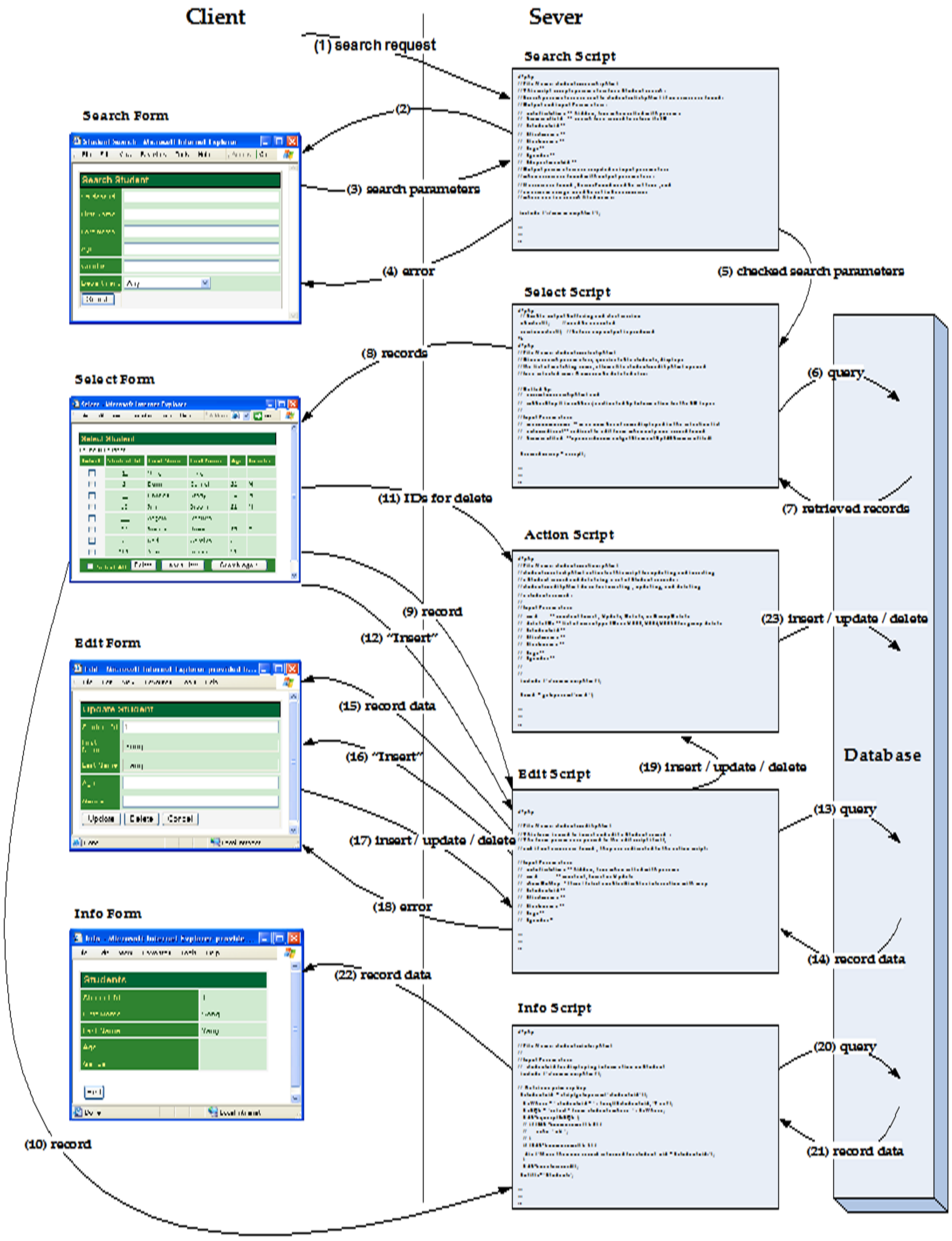


Figure 5: Interactions among Web Scripts and Forms

3. An edit form can insert a record for a geographical feature, after transforming the coordinate values from the spatial reference used by the current map interface to the one used by the geometry column for the record.

The forms related are automatically linked to each other. Figure 6 shows, as an example, the *edit* form for a Student table. From this edit form, the user can open the forms for the department and courses related to the student. The information needed to create the links are extracted from the *primary-key/foreign-key relationships* among the tables in the database.

In order to generate the Student edit form shown in Figure 6, the following entry is provided in the configuration file.

```
$edit_fields=array(
   array("column"=>"student_id",
     "label"=>"Student Id", "type"=>"numeric",
     "maxlen"=>"40", "size"=>"40"),
   array("column"=>"first_name",
     "label"=>"First Name", "type"=>"text",
     "maxlen"=>"40", "size"=>"40"),
   array("column"=>"age", "label"=>"Age",
     "type"=>"text", "maxlen"=>"40", "size"=>"40"),
   array("column"=>"gender", "label"=>"Gender",
     "type"=>"text", "maxlen"=>"40", "size"=>"40"),
   array("column"=>"department_id",
     "label"=>Department ID", "type"=>"to_one",
     "linked_table"=>"departments", "maxlen"=>"40",
     "size"=>"40"),
   array("label"=>"Courses Taken", "type"=>"to_many",
     "linked_table"=>"courses", "maxlen"=>"40",
     "size"=>"40"),
   array("column"=>"other_info",
     "label"=>"Other Information",
     "type"=>"textarea", "rows"=>"4",
     "cols"=>"32",
);
```

Each element in array `$edit_fields` represents a field in an edit form, with options `column`, `label`, `table`, `maxlen`, `size`, and `type`. Option `type` can be `numeric`, `text`, `time`, `date`, `email`, `phone`, `textarea`, `to_one`, or `to_many`.

textarea: The input is a string displayed in a text area.

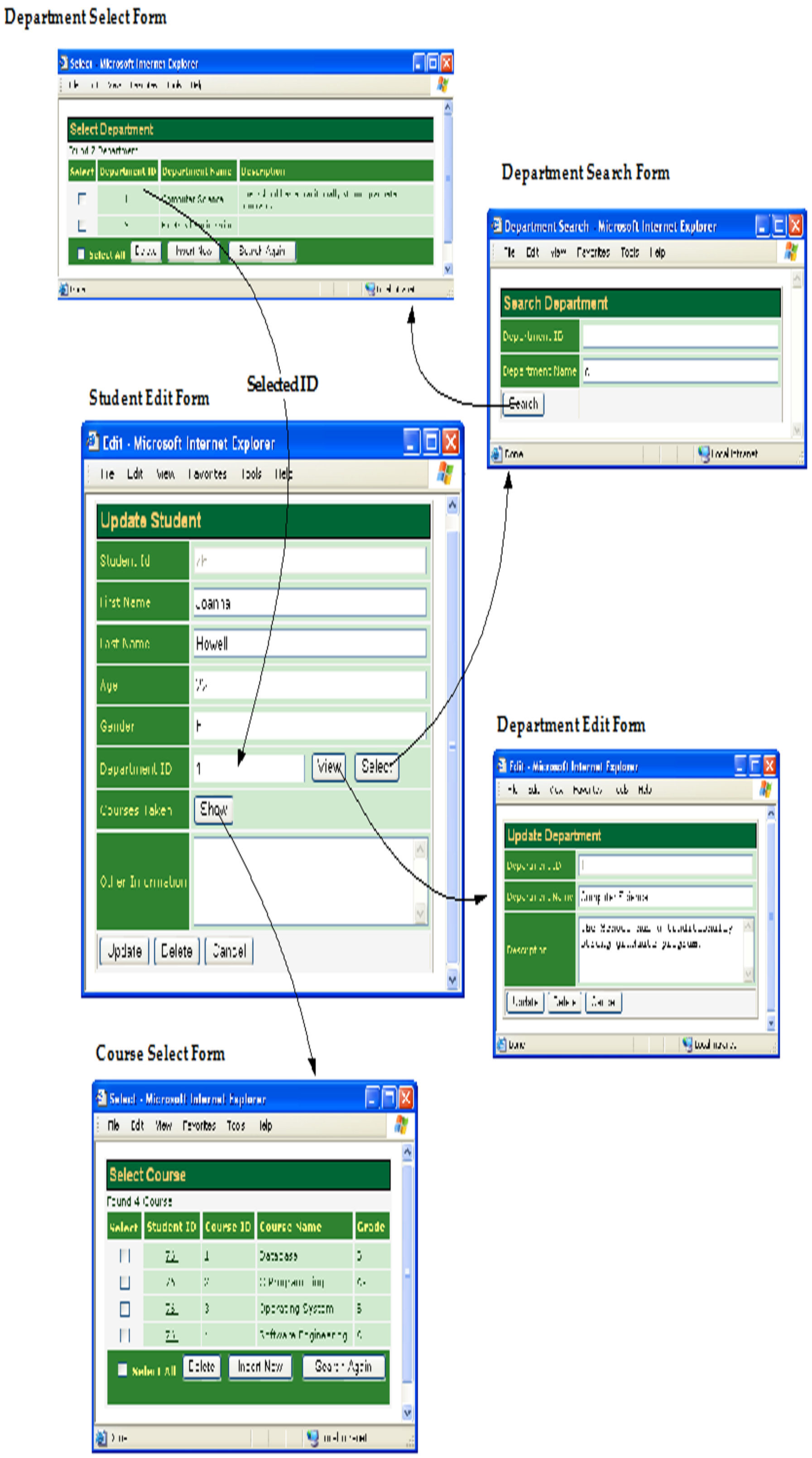
to_one: A field of type `to_one` allows a user to view or modify the record related to the current record via a *one-to-one* or *many-to-one* relationship type. In our example, column `department_id` in table `students` is specified as `to_one`. When the `View` button is clicked, the edit form showing the record of the student's department is displayed. Furthermore, a user can search and select a new department record to be linked. Option `linked_table` designates the name of the table storing the associated record. Option `child_column` indicates the column for linking in the associated table. If this option is omitted, the name of the column for linking is identical to that of the foreign key column in the current record.

to_many: The purpose of this type is to list the records related to the current record via a *one-to-many* or *many-to-many* relationship type. In our example, the field labelled `Courses Taken` is specified to be `to_many`. When the `Show` button is clicked, all the courses taken by the student are displayed in the select form for courses. This type needs options `linked_table`, `parent_column`, and `child_column`. Option `linked_table` and option `child_column` can be omitted, when the name of the foreign key column of the associated table is identical to that of the primary key column of the current record. Option `parent_column` indicates the *foreign key* column in the current table. If this option is omitted, the foreign key column is the primary key column of the current table.

Automatic Generation of Map-Layer Configuration Files

We can semi-automatically create a map-layer configuration file by using information in the table `geometry_columns` that stores GIS-related metadata and the layer names in a map file. Figure 7 shows this process for region *xxx*.

1. The default *map-layer meta configuration file* *xxx*`_maplayers.mconfig`, which lists all the layer groups, can be created by **map_mconfgen**. The map-layer group names are generated from the table names stored in table `geometry_columns` and the map-layer group names defined in the map file.
2. We can customize the default map-layer meta configuration file, whose format is identical to a map-layer configuration file, by selecting and reordering the layer groups. We can also provide customized values for the parameters in each layer group. The meta configuration file needs to keep only the definitions of customized parameters.
3. Finally, map-layer configuration file *xxx*`_maplayers.config` is generated by **map_confgen**. The information required by this step is



Select Department

Found 2 Department

Select	Department ID	Department Name	Description
	1	Computer Science	[illegible]
	[illegible]	[illegible]	

Search Department

Department ID	
Department Name	[illegible]

Update Student

Student Id	[illegible]
First Name	Joanna
Last Name	Howell
Age	[illegible]
Gender	F
Department ID	1
Courses Taken	
Other Information	

Update Department

Department ID	1
Department Name	Computer Science
Description	[illegible]

Select Course

Found 4 Course

Select	Student ID	Course ID	Course Name	Grade
	21	1	Database	5
	21	2	C Programming	[illegible]
	21	3	Operating System	5
	21	[illegible]	Software Engineering	[illegible]

Figure 6: Student Edit Form

retrieved from the meta configuration file, table `geometry_columns`, and the map file. The default values of the parameters determined by the latter two are overwritten if they are customized in the meta configuration file.

A map-layer meta configuration file can define any parameters that are allowed for a map-layer configuration file. However, the default one generated by **map_mconfgen** contains for each map layer group the definitions only for the following parameters:

layer_selectable – A boolean option that determines whether the layer can be selected for spatial operations or not.

legend_label – The name of the layer that will be displayed on the map legend.

An example of a default map-layer meta configuration file is shown below:

```
$layer_groups = array(
   'grp_eo_py' => array(
      'layer_selectable' => true,
      'legend_label' => 'Eo Py',
),
   'grp_eosrc_' => array(
     'layer_selectable' => true,
     'legend_label' => 'Eosrc Pt',
),
...
```

The user may modify the values of these definitions and add others. For example, the meta configuration file can be customized as follows:

```
$layer_groups = array(
   'grp_eo_py' => array(
      'layer_selectable' => true,
      'legend_label' => 'EO Polygons',
      'search_script' => 'forms/eo/eo_py_eo_search.phtml',
      'select_script' => 'forms/eo/eo_py_eo_select.phtml',
      'edit_script' => 'forms/eo/eo_py_edit.phtml',
      'img_src' => 'images/eo_poly.png',
),
   'grp_eosrc_pt' => array(
      'layer_selectable' => true,
      'legend_label' => 'EOSRC Points',
      'search_script' => 'forms/eo/eosrc_pt_search.phtml',
      'select_script' => 'forms/eo/eosrc_pt_select.phtml',
      'edit_script' => 'forms/eo/eosrc_pt_edit.phtml',
      'img_src' => 'images/eosrc_pt.png',
),
...
```

The following customization was done:

1. `legend_label` was changed to the desired layer name for the map legend.
2. The names and locations of the search form, select form, and edit form are specified in `search_script`, `select_script`, and `edit_-script`, respectively.
3. The source file name of the icon in the map legend was specified in `img_src`.

The order of the map-layer groups can also be changed to that desired in the map legend.

Then the map-layer configuration file shown previously can be generated by **map_confgen**.

WebGD Development History

The WebGD framework and WebGD-Gen were developed *incrementally* and *iteratively* during the last four years. We first implemented in 2000 an application that allowed point features to be inserted on a map by using ASP with ArcIMS and ArcSDE. In 2001, we re-implemented this application with ASP.NET as ASP.NET provides Web controls, which are better building blocks for Web pages. Based on this application, the first version of WebGD framework was created in 2002 in order to support multiple applications.(15)

In early 2003, we re-implemented an application called Motels Oregon with MapServer, PostGIS, and PostgreSQL(9) This version on Linux was more reliable and faster than the old one, as well as being built with free software. While implementing the next MapServer application, which was a germplasm resource management system (GEMGIS), we created the first version of WebGD framework for MapServer. This framework was then enhanced so that it can handle polygon features as well as point features.

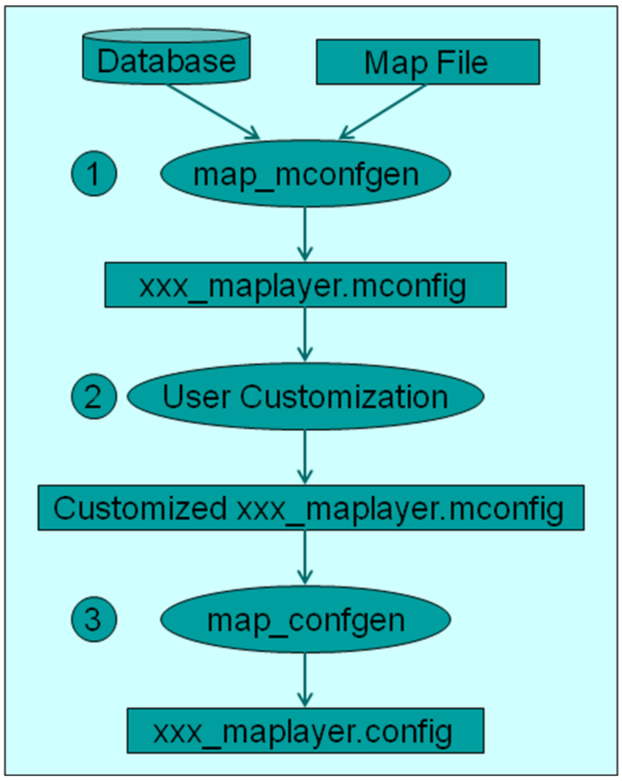


Figure 7: Process of Creating a Map-Layer Configuration File

The two major enhancements made to the WebGD framework in 2004 were *dynamic switching* of `spatial references` for different regions and *automatic generation* of Web forms that can be used to insert, query, and delete geographical features. Compared to an application that simply displays geographical features as points on a map, the current WebGD framework is roughly 20 times more complex in terms of the time we spent implementing the required features.

Conclusions and Future Work

We have developed the WebGD framework and the WebGD-Gen application generator for rapid development of Web-based GIS/database applications.

1. Geographical features, such as habitats of plants and animals, road-work sites, and waterlines, can be inserted, queried, and deleted with the map interface and Web forms displayed on a standard Web browser.
2. An application can be created without any programming. The map interface and Web scripts for data access can be automatically generated from configuration files, and those configuration files can be generated from the database schema and the GIS metadata, i.e. information stored in the table `geometry_columns`. Automatic generation of a Web-based GIS application not only reduces the development cost significantly, but it also facilitate incremental and iterative development of the application.
3. Dynamic switching of spatial references allows an application to cover different regions with different map files, projections, map legends, and quick-view lists. This is an important feature needed for an application that covers the entire USA or the world.
4. We created the WebGD framework by using only free open-source software. The software tools we use, such as the University of Minnesota MapServer, PostgreSQL DBMS, PostGIS, Apache, and PHP are all available for free. The GIS data used, such as those from USGS, TIGER/LINE, and Digital Chart of the World (DCW), are also in the public domain. Therefore, the framework is available for anyone for free use.
5. The cost of running our applications is extremely low. We could put copies of such large databases as Biotics, SSURGO2 soil data, and a part of National Germplasm Resource Information System on a $800PC.

Automatic code generation of a WebGD application will save a great deal of effort in the development of a spatial decision-support system. Although some manual customization is required, the time needed for customization can be lowered to weeks or months compared to the years required to build a *spatial decision-support system* from scratch.

The WebGD framework and WebGD-Gen are currently available upon request. In order to release them to the public as free and open-source software, we are looking for collaborators. We are also reimplementing the map interface by using OpenLayers to support smooth panning.

Bibliography

[1] P. H. Dana Map Projection Overview http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj_f.html

[2] DM Solutions Group Inc. PHP MapScript http://www.maptools.org

[3] R. Kingston (1998) Web-based GIS for public participation decision making. In *Procs* of NCGIA PPGIS Meeting, Santa Barbara, California. Retrieved Map 2003 from http://www.ncgia.ucsb.edu/varenius/ppgis/papers/kingston/kingston.html

[4] Eum D. and Minoura T. (June 2003) Web-based database application generater. *IEICE Transactions on Information and Systems, Vol. E86-D, No. 6.*

[5] Fogelsong, C. (2002) Biotics 4.0 data model version 1.0. Retrieved January 5, 2004, from http://whiteoak.natureserve.org/hdms/HDMS-DataModel.shtml

[6] McKenna, Jeff MapServer PHP/MapScript Class Reference - Versions 3.6, 4.0 & 4.2 DM Solutions Group Inc.

[7] NatureServe (February 2002) Element Occurrence Data Standard. Retrieved January 4, 2004, from http://whiteoak.natureserve.org/eodraft/all.pdf

[8] NatureServe (December 2003) Biotics 4.0 Getting Started Guide. Retrieved January 5, 2004, from http://whiteoak.natureserve.org/hdms/biotics-learn-more.shtml (now obsolete).

[9] Sano J., Wanalertlak N., Maki A., and Minoura T. (July 2003) Benefits of web-based GIS/database applications. In *Prosc. of 2nd Annual Public Participation GIS Conference* Portland, Oregon.

[10] USDA-ARS Western Regional Plant Introduction Station, USDA - Agricultural Research Service, Pullman, Washington http://www.ars-grin.gov/ars/PacWest/Pullman/

[11] Ramsey, Paul PostGIS Manual Refractions Research Inc.

[12] University of Minnesota (2003) MapServer http://mapserver.gis.umn.edu

[13] Sharma, A. (December 2003) Web-based analysis module for a germplasm collection. Master of Science report, School of Electrical Engineering and Computer Science, Oregon State University

[14] Wangmutitakul P, Li L, and Minoura T. (March 2003) User Participatory Web-Based GIS/Database Application. *Proc. of Geotec Event Conference*

[15] Wangmutitakul Paphun et al. (2004) Framework for Web-based GIS/database Applications *Journal of Object Technology 3*, 209-225

[16] Wuttiwat T., Minoura T. and Steiner J. (May 2003) Using Digital Orthographic Aerial Images as User Interfaces *Proc. of ASPRS Annual Conference*, Anchorage, Alaska

Toshimi Minoura, Nirut Chalainanont, Junya Sano
Oregon State University
http://engr.oregonstate.edu/~minoura/research/creeda/
minoura AT eecs.orst.edu

db4o2D - Object Database Extension for 2D Geospatial Types

Stefan Keller

db4o stands for "database for objects". It's a native object oriented database management system (OODBMS) written in Java and .NET and thus targeted towards these two platforms. The software was first released 2001 by db4objects, Inc., and since then it got a major market share among the so called second generation object databases. It is available under two licenses ("dual licensing model"), an open source licence of type GPL for personal and non-commercial use as well as a commercial license.

In this contribution we first introduce db4o. In the chapter *OODBMS and db4o* we discuss the advantages, limitations and differences from a conceptual and a programmer's perspective. Then we report

The Open Source Geospatial Foundation, or OSGeo, is a not-for-profit organization whose mission is to support and promote the collaborative development of open geospatial technologies and data. The foundation provides financial, organizational and legal support to the broader open source geospatial community. It also serves as an independent legal entity to which community members can contribute code, funding and other resources, secure in the knowledge that their contributions will be maintained for public benefit. OSGeo also serves as an outreach and advocacy organization for the open source geospatial community, and provides a common forum and shared infrastructure for improving cross-project collaboration.

Published by OSGeo, the OSGeo Journal is focused on presenting discussion papers, case studies and introductions and concepts relating to open source and geospatial software topics.

Proceedings Editorial Team:

- Angus Carr
- Mark Leslie
- Scott Mitchell
- Venkatesh Raghavan
- Micha Silver
- Martin Wegmann

Editor in Chief:
Tyler Mitchell - `tmitchell AT osgeo.org`

Acknowledgements
Various reviewers & the GRASS News Project

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this journal, the LaTeX 2_ε style source has been kindly provided by the GRASS and R News editorial board.

Journal online: `http://www.osgeo.org/journal`
OSGeo Homepage: `http://www.osgeo.org`
Postal mail: OSGeo
PO Box 4844, Williams Lake,
British Columbia, Canada, V2G 2V8
Telephone: +1-250-277-1621

ISSN 1994-1897