<table>
<thead>
<tr>
<th>Title</th>
<th>Java-based digital library portal for geography education (Main article)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Liu, Zehua; Yu, Hai; Lim, Ee Peng; Yin, Ming; Goh, Dion Hoe-Lian; Theng, Yin-Leng; Ng, Wee Keong</td>
</tr>
<tr>
<td>Date</td>
<td>2004</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10220/6173">http://hdl.handle.net/10220/6173</a></td>
</tr>
<tr>
<td>Rights</td>
<td>Science of Computer Programming @ copyright 2004 Elsevier. The journal's website is located at <a href="http://www.sciencedirect.com/science?_ob=ArticleURL&amp;_udi=B6V17-4CK7XM0-3&amp;_user=10&amp;_rdoc=1&amp;_fmt=&amp;_orig=search&amp;_sort=d&amp;_docanchor=&amp;view=c&amp;_acct=C000050221&amp;_version=1&amp;_urlVersion=0&amp;_userid=10&amp;md5=21326aa6b78235ac17dc78603e096f56">http://www.sciencedirect.com/science?_ob=ArticleURL&amp;_udi=B6V17-4CK7XM0-3&amp;_user=10&amp;_rdoc=1&amp;_fmt=&amp;_orig=search&amp;_sort=d&amp;_docanchor=&amp;view=c&amp;_acct=C000050221&amp;_version=1&amp;_urlVersion=0&amp;_userid=10&amp;md5=21326aa6b78235ac17dc78603e096f56</a>.</td>
</tr>
</tbody>
</table>
Java-based Digital Library Portal for Geography Education

Zehua Liu a Hai Yu a Ee-Peng Lim a, * Ming Yin a
Dion Hoe-Lian Goh b Yin-Leng Theng b Wee-Keong Ng a

a Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore 639798, SINGAPORE
b School of Communications and Information, Nanyang Technological University, Singapore 639798, SINGAPORE

Abstract

G-Portal is a Java-based digital library system to manage the metadata of geography-related resources on the Web. In addition to providing a flexible repository subsystem to accommodate metadata of different formats using XML and XML Schemas, G-Portal organizes metadata into projects and layers, and supports an integrated and synchronized classification and map-based interfaces over the stored metadata. G-Portal also includes a classification subsystem that creates category structures and classifies metadata resources into categories based on user-specified classification schemas. Furthermore, G-Portal users can annotate resources and make their annotations available to others. In this paper, we describe the design and implementation of G-Portal and elaborate how Java is used to implement its features. G-Portal has been designed to be modular and some of the modules can be used as standalone tools. In this paper, we use UML notations to describe the detailed design of G-Portal and highlight some of the design decisions.

Key words: Digital Libraries, Geography Education, Metadata Management

PACS:

* Corresponding author

Email addresses: liuzh@pmail.ntu.edu.sg (Zehua Liu), ashyu@ntu.edu.sg (Hai Yu), aseplim@ntu.edu.sg (Ee-Peng Lim), asmyin@ntu.edu.sg (Ming Yin), ashlgoh@ntu.edu.sg (Dion Hoe-Lian Goh), tyltheng@ntu.edu.sg (Yin-Leng Theng), awkng@ntu.edu.sg (Wee-Keong Ng).

Preprint submitted to Elsevier Science 17 October 2003
1 Introduction

1.1 Background

The ubiquity of World Wide Web has brought about significant changes to the role of libraries in schools, universities, public and private institutions. The Web provides enormous amount of information in web pages that can be browsed for free. Compared to a decade ago, students and researchers today are more likely to patronise the Web instead of visiting the physical libraries for information.

While the Web is preferred over traditional libraries in terms of volume and scope, the tools for selecting, organising, indexing Web content remain scarce. Users may be able to use Web search engines to locate web pages by keywords but the search results can be inaccurate and the information found in web pages may not be authoritative. In contrast, a traditional library provides more authority control over its library catalog data in the following ways:

- Library catalog data are formatted using a standard catalog record format, e.g., MARC [9];
- Every library catalog record is further assigned a subject category which have been standardized across public libraries; and
- Finally, the search interfaces of traditional libraries are usually very similar and are relatively easy to use. The search accuracies are often very high.

In this paper, we therefore introduce a digital library portal known as G-Portal that attempts to provide better selection, organization, and indexing of Web content of educational values. More specifically, G-Portal is designed to manage metadata of geography-related Web resources which can be web pages or any other Web objects that may be relevant to the geography education in junior and senior high schools. Examples of such Web resources include a web page describing the mountains in East Malaysia, or a detailed map of China. Metadata here refers to data about data. Similar to library catalog, G-Portal’s metadata provide description about the actual Web resources and the URL references to the latter. The similarity between G-Portal metadata and library catalog however ends here as the referenced Web content are found within the global Web and are therefore not within the control of G-Portal.

Moreover, unlike other Web portals, the design of G-Portal must address three important characteristics unique to its metadata content:

- **Spatial attributes**: Many of G-Portal’s metadata can be associated with locations and shapes on a map. These are known as the *geospatial metadata*. For example, for web pages containing the census data about dif-
ferent states in USA, each state’s census data web page can be associated with a polygon representing the state boundary. It is therefore necessary for G-Portal to support digital library services dealing with geospatial metadata.

- **Diverse metadata structure**: The broad range of geography-related Web content suggests that there is no one single structure that can be used for representing G-Portal’s metadata. This is very different from the usual Web portal and library catalog data where a single set of attributes suffice.

- **Lack of a standard set of subject categories**: There is currently no standard set of subject categories defined for Web content. Hence, it is not feasible to expect metadata records created in G-Portal to be assigned subject categories similar to library category records.

Due to the above unique metadata characteristics, the design of G-Portal has incorporated several novel ideas into the digital library services provided to its users. These include:

- A flexible classification scheme for categorizing metadata resources;
- A versatile mechanism to logically organize metadata resources;
- A resource visualization module that allows the same metadata resource to be navigated in both a map-based and a classification-based user interfaces;
- A query module that supports both spatial and non-spatial queries on metadata of different formats.

To allow users to easily access to the above features, we have chosen to implement G-Portal in Java. Apart from being object oriented, Java is ideal for implementing graphical user interfaces required by G-Portal. For example, the viewing and manipulation of maps can be made much simpler if users can carry out zooming, panning and other operations interactively with G-Portal. Furthermore, by implementing the G-Portal client as a Java Applet, G-Portal can run on any Java-enabled Web browser without any installation efforts. In other words, G-Portal can become platform independent making itself available to any user on the Web.

At the server side, the decision to also use Java to implement the server program is mainly for the ease of communication with the client, since the client is implemented using Java. Portability is another important consideration that affect our decision. In fact, the project has taken great advantage of the platform-independent feature of Java by having the development on Windows platform and the deployment on a Sun Solaris server. In addition, Java is favoured because of its faster application development time due to the wide availability of high level API and third party packages.
1.2 Objectives

In this paper, we will describe the design and implementation of G-Portal. We will place much emphasis on some interesting ideas realised in G-Portal making it a unique digital library portal. The objective here is to share our G-Portal experience with the researchers and developers of other similar applications.

At the system level, G-Portal consists of several essential functional modules which are designed to be reusable for other similar applications. EXtensible Markup Language (XML) and other XML facilities also have been used extensively in G-Portal. At the component level, the Java packages and classes for realising the functional modules will be described. We will also outline the implemented methods.

While G-Portal can be viewed as a learning tool for users to acquire geography-related knowledge, its current design is being extended to support e-learning activities. Nevertheless, due to the rather preliminary findings and space constraints, we will exclude the the e-learning aspects of G-Portal from this paper.

1.3 Related Work

There have been several ongoing research efforts making Web content available to users in a more structured and organized way [1–4,7,8,13–17,19]. In this section, we give a brief review on those that are most closely related to our G-Portal project.

Semantic Web, an extension over the current Web to associate well-defined meanings to web pages and other web objects, is an initiative advocated by Tim Berners-Lee [1]. In Semantic Web, the objective is to make Web information “consumable” by software agents by representing Web data together with their meta-level descriptions marked up using XML or other markup languages. Ontologies are also developed to make it possible for Semantic Web information to be interpreted in a consistent manner. The Semantic Web activities, however, largely focus on metadata representation and interpretation. Research in digital library services such as query, classification and annotation are largely missing. In G-Portal, we will instead have to emphasize on the above services.

The Alexandria Digital Project (ADEPT) [16] is a project initiated by the University of California, Santa Barara to host data related to earth science. The ADEPT research focuses on the creation and management of data content. The research further supports the maintenance of metadata, their organization, and spatial (and non-spatial) query processing on them. The ADEPT
prototype system, however, differs from G-Portal by not adopting flexible metadata representations and classification. In fact, ADEPT assumes that all its data are hosted by a computer server and they are compliant to a single metadata format.

The DLESE Project [17] is very similar to our G-Portal project since both DLESE and G-Portal recognise the importance of tapping content from the Web. DLESE maintains metadata about web entities identified by some user community established by teachers, students and researchers. These metadata unfortunately must be compliant to a single metadata format that does not include the notion of spatial attributes. Furthermore, DLESE’s implementation is not Java-based. Hence, complex user interactions cannot be easily supported by DLESE.

1.4 Outline of Paper

The remaining sections of this paper present G-Portal in detail. We first give an overview of the system by going through a use case in Section 2. The system architecture is presented in Section 3. Section 4 discusses how resources are represented and organized in G-Portal. Section 5 and Section 6 cover how resources are visualized and classified, respectively. Query processing of resources is described in Section 7. Finally, we give the concluding remarks and our future work in Section 8.

2 Overview of G-Portal

As an educational digital library portal for geography-related information, G-Portal must provide a meaningful way to organise its content and services allowing users to effortlessly look for information. In this section, we therefore give an overview of G-Portal by describing a typical example scenario involving a student user. As we walk through the scenario, important terms used in G-Portal and its functional modules will be introduced. More detailed descriptions of these terms and modules will be given in the latter sections.

The content maintained by the G-Portal system are metadata resources that describe some pre-existing Web resources located on public domain Web sites, which we call the raw resources. For convenience, we shall just call the G-Portal metadata resources as resources in the rest of our discussion, unless otherwise stated.

In this example scenario, the user is a high school student who would like to
prepare for his/her GCE O Level (high school equivalent) geography examination. The student first enters the URL of the G-Portal system in a web browser and logs on to the system. The system then presents the user with a list of projects. Project is a new concept introduced to define the collection of resources that are relevant to a use case or a specific task. In this case, the user selects the project named GCE Exam that contains resources related to geography examination. These resources include examination questions, answers, countries, mountains and other geography resources gathered beforehand by the project creators who are the geography teachers. Within the project, resources are further grouped into layers for fine-grained organization as shown in the left bottom window of Figure 1. As shown in the figure, question resources are grouped into the Questions layer. Answers and supplemental materials are grouped into the Answers & Sup layer.

After the project is loaded, two main user interface windows are shown (see Figure 1). On the left, resources are drawn as objects on the map in the map-based interface. This is a primary point-of-access to the geospatial resources allowing users to locate a resource with known location. Usual map navigation aids such as zooming and panning are provided within this map-based interface. Alternatively, a classification-based interface is also provided to browse resources classified in taxonomies of categories, especially those resources without spatial features. In this example, since the student is interested in exam questions, which do not have spatial attributes, he/she can begin by browsing examination questions classified by theme or by year using...
the classification-based interface. It should be noted that the classification-based interface is supported by a flexible classification scheme that allows resources to be automatically classified into various taxonomies. For example, examination question resources in the figure have been classified by year and by theme in this project. This is in contrast with the single taxonomy approach used in almost all the existing digital libraries.

Suppose the user has selected one question from, say, the population category. To view the content of the selected resource, the user double-clicks on the question. The content, represented in XML, will appear in an XML viewer that we have developed, as shown in Figure 2(a). Using the XML viewer, user can view metadata elements nested in XML tags which provide both the semantics and structure meanings to the data. G-Portal represents all its resources in XML and requires resources (including annotation resources) of the same type to be compliant to a resource schema that defines the internal structure of the resources. For example, an examination question schema is defined for all examination questions in this project to ensure that all question XML resources provide the same structure. The self-describing and semi-structured nature of XML data model makes it simple to represent a wide range of metadata resources. However, as a high school student, the user may not be familiar with XML. He/she can choose to view the resource content in a more familiar format. In G-Portal, resource contents can also be transformed into HTML webpages using the underlying resource content transformation mechanism. Figure 2(b) shows the HTML counterpart of the resource content in

Fig. 2. Visualization of Resource Content
Figure 2(a). In this example, the transformation involves extracting the URL source of the selected question metadata and returning the raw question resource in HTML using the URL.

While reading the details of the question, the student may notice that the question is about the comparison of information about China and USA. Since countries are also available as resources in the project and are viewable on the map, the student can use the map-based interface to quickly find out the two countries and view the contents associated with these two country resources.

Using the population information obtained from these two country resources, the student may arrive at an answer and would like to compare it with the standard answer. Answers in this project are created as annotations of the question resources. Resource annotation in G-Portal provides a mechanism to attach additional knowledge to metadata resources and the annotations can be shared. To keep our resource data model simple, we represent annotations as a special type of metadata resources. To view the answer of the selected question, the student finds the answer to the question by asking the system to show the annotations about this question that are of the type answer. Like the normal resources, the answer annotation content can be viewed using the XML viewer or web browser. The student can then check against his answer and read the detailed explanation in the standard answer.

Having successfully attempted a question, the user may want to find out more.

1 This requires resource content transformation.
questions about USA. He/she can use the query interface to look for questions containing the word “USA” as shown in Figure 3. Using the query interface, the student can specify search criteria on the resource attributes, or a bounding area to constrain the resources to be returned, or both. In this example, only the keyword word “USA” is specified against the country name attribute. Suppose a few questions are returned as the query result and the student double-clicks on the first one which compares the population of USA with that of, say, Ethiopia. Since Ethiopia is a country not known to the student, he/she can again use the query interface to search for a country named “Ethopia”. The synchronization between the search results and the map-based interface will display Ethiopia’s location in Africa. After viewing the resource content of this country, the user may find it interesting and would like to contribute a new comment. This can be done by creating an annotation about this country resource with the help of a built-in XML editor. Unlike other editors, this XML editor is aware of the schema of the resource (or annotation resource) and is able to guide editing process such that the final resource created will be compliant to the required resource schema.

This concludes our brief overview of G-Portal and a use case example demonstrating how a typical user can use G-Portal to explore, learn and share knowledge about geography related information.

3 System Architecture

3.1 Design Architecture

The overall system architecture of G-Portal is shown in Figure 4. There are three types of users: administrator, project administrator, and users.

A G-Portal administrator manages all databases in G-Portal using the project management and user management modules. He/she can create new user accounts, delete existing ones and define users’ roles that in turn determine the privileges to access the functional modules in G-Portal. All user information are stored in the user database.

Each project administrator uses the project management and resource management modules to create and configure projects that can be used by ordinary G-Portal users. The project management module allows project administrators to define projects, create layers that bundle various types of resources, group these layers into the defined projects and setup appropriate access control. The resource management module lets them register new types of resources (i.e., new resource schemas) and populate the projects with re-
source instances. The project and layer information are stored in the **project and layer database**. Resource instances and schemas are stored in the **resource database** and **schema database**, respectively. A project administrator can also use the **resource classification** module to associate classification schemas with resources and classify resources.

The rest of G-Portal users with ordinary access rights constitute the majority of targeted users. These users make use of the resource management, **resource annotation**, **query engine** and resource classification modules to perform various tasks related to geospatial resources, as described in Section 2. The visualization module allows them to access the resources through both a map-based and a classification-based interfaces and view the resource contents in raw XML format as well as transformed formats such as HTML. The classification module classifies resources on the server and displays the resulted taxonomies in the client’s classification interface. Users can query resources based on spatial and non-spatial attributes using the query engine. They can also contribute new annotations about existing resources.

As the focus of this paper is on the implementation issues, we only give brief descriptions of the modules. A more detailed coverage can be found in [10].

### 3.2 System Deployment

The design architecture described in Section 3.1 has been implemented in a standard client-server model. The client is a Java applet realizing the visualization module (the map-based and classification-based interfaces) and front-end interfaces of other modules. On the server-side is a Java server program that communicates with the client applet and performs necessary operations on the
back-end databases to support the front-end interfaces. Therefore, most of the modules depicted in Figure 4, except the visualization module, have both a client and a server component.

Figure 5 shows the deployment diagram of the system. There are two main components: **client applet** and **server program**. The client applet initially resides on the Web server and is downloaded by the user into the client machine using a Web browser (e.g., Internet Explorer or Netscape Navigator). To support the operations and commands invoked by the user, the applet sends requests to and expects responses from the server program running on the Web server.

One decision that we have made in implementing the client is to use Java 1.1 instead of the more commonly used Java 2. The main reason is that we want to allow users to directly access the digital library portal using a Web browser with minimal installation requirements. As the majority of Web browsers are still installed with only the Java 1.1 virtual machine, restricting the client to using only Java 1.1 features eliminates the need for users to install a Java Plug-in.

The main role of the server program is that of a gateway between the client and the two database servers, namely the **Tamino XML Server** and the **Informix Relational Database Server**. In addition, several modules shown in Figure 4 (e.g., resource classification and query engine) require part of their operations to be performed in the server program to keep the client applet simple.

The relationship between the server program and the two database servers can also be considered as a client-server model, with the server program now acting as a client to access the data managed by the database servers. The Tamino XML Server is used to host the resource database and the schema database. The Informix RDBMS Server on the other hand has been used to
store the project and layer database and user database. To facilitate spatial queries on geospatial resources, we also keep the spatial element of resources in the Informix server. Section 4 will elaborate in further detail how the resources are represented in Java classes and stored in the XML and Informix databases.

3.2.1 Client Applet & Server Program Packages

All classes in the client applet are arranged within the client package and its sub-packages. The client package contains the classes for displaying the map-based interface and handling the flow of control of the entire client applet. The client.core package includes classes for the representation of the basic concepts in G-Portal, such as resource, layer and project. Classes for visualizing and editing the content of resources are in the package client.resourceinfo and client.xmleditor. Classes related to the classification-based interface are grouped into the client.classification package and those designed to facilitate querying of resources are in the package client.query. The client.connection has all the classes that deal with the communication with the server side. There are a few other packages of classes mainly for the GUI; thus not discussed here.

The classes responsible for the majority of the server functionalities and communication with the client reside in the server package. The classes in the server.db and server.tamino packages deal with connections with and storing / retrieving data to/from the Informix Database Server and Tamino XML Database Server, respectively. The server.classification package contains classes for classifying resources and the server.query package contains those for performing queries on resources residing in local databases as well as re-
mote repositories.

The util package contains some common classes that both the client and server packages require. This includes the classes for representing geography objects (util.spatial) and XML helper functions (util.xml).

3.2.2 Important Detailed Design Issues

In the course of designing and implementing the system, some of the important issues that we have encountered include:

- **Resource Representation.** The unique requirement of combining spatial attributes together with XML representation for resources poses a challenge to devise appropriate data structures to represent resources, layers and projects. This involves considerations on their memory-based data representation (for the client applet) and the database design (for the server program).

- **Visualization and Editing of Resource Content.** Due to the use of XML to represent resource content, G-Portal provides XML editor and resource viewer with transformation option to display and modify resources in a more visually appealing way. The design and implementation of these components will have to be studied.

- **Resource Classification.** A flexible classification scheme has been devised to classify the heterogeneous resources in G-Portal. However, supporting such scheme requires a new interface for visualizing the classified resources and a classification engine for performing the classification.

- **Resource Querying.** We have proposed an X-Query based query language for querying resources based on spatial and non-spatial attributes. The design of query engine and its query evaluation strategy have to be determined.

4 Resource Representation and Organization

4.1 Resource Data Model

As mentioned earlier, G-Portal focuses specifically on geography related metadata resources and some may be associated with locations on the map. This calls for the explicit inclusion of location in the resource definition. This location attribute, together with several other attributes, including id, name, owner, source, constitute the core attributes that every resource must have. In addition to these mandatory core attributes, each type of resources is al-
allowed to define other new attributes to better describe the resources. The set of mandatory core attributes together with the customized attributes enable the resource data model to represent a heterogeneous range of metadata. The complete attribute structure shown by resources of the same type is defined by a resource schema. To allow flexible representation of resources with semi-structured attributes, we adopt XML as the resource representation format and XML Schema [18] for defining resource schema.

Figure 7 depicts the resource of a geography examination question, which also exists as a webpage. This resource contains a set of XML elements such as ID, ResourceName, Location, Creator, Source that serve as the core attributes. The XML elements under the Content element are those that are specific to this type of resource, i.e., ExamQuestion resource.

In G-Portal, we allow users to share their knowledge and contribute content to the system by allowing them to create annotations on existing resources [12]. Annotations are represented as a special type of resources, with an additional attribute that locate the annotated resources. Due to space constraints, we shall not elaborate the annotation scheme further in this paper.

To implement the resource data model described above, there are several issues to be considered:
• The display and manipulation of resources in the visualization module and other front-end interfaces of other functional modules requires some classes to be carefully designed for representing resources. This requirement also applies to project and layer information.
• To keep the projects, layers and resources persistent on the G-Portal’s databases, the design of resource database, project and layer database and user database will be required.

The first requirement is mostly relevant to the client applet of G-Portal while the latter is an issue to be addressed at the server end. In the following, we discuss the approaches taken by G-Portal to address both.

4.2 Resource Class Design

G-Portal client applet, as described in Section 2, supports several graphical user interface features to visualize the resources in both map-based and classification interfaces. To support the spatial related representation and visualization, we have adopted GeoTools\(^2\), a public domain GIS package. As a general purpose GIS package, the GeoTools’ API is too primitive and does not fit well into the resource data model in G-Portal. On the other hand, although XML has been used to represent the resource information, the Java libraries for XML lack the support of GIS features and can not be directly applied. In G-Portal, we, therefore, adopt a hybrid approach to take advantages of both the GeoTools package and the Java libraries for XML.

Figure 8 shows the design of resource related classes. These classes are mainly found in the `client.core` package and the Java packages in the GeoTools. A set of resources of the same type under the same layer are represented by the `GPResourceGroup` class. This class contains a member of the `GPTheme` class, derived from the `Theme` class in GeoTools, that stores the geometry shapes of resources, through the inherited `Layer` interface\(^3\) from `Theme`. The shapes are actually stored in the subclasses that implement the `Layer` interface, depending on the types of the shapes. Other core attributes such as id, name and owner are found in the `GeoData` class, which is also from the GeoTools package. Besides these classes, other utilities from GeoTools, such as those for rendering shapes on the map and for basic map navigation (e.g., zoom-in, zoom-out and pan), have also been directly incorporated (not shown in Figure 8).

Instances of the `GPResourceGroupKey` class are used throughout the client

\(^2\) [http://www.geotools.org/](http://www.geotools.org/)

\(^3\) Note that the term `layer` used in GeoTools is different from the concept of layer in G-Portal.
applet to maintain the identifiers of \texttt{GPResourceGroup} instances each representing a set of resources sharing the same resource schema. Each \texttt{GPResourceGroup} also has a reference to its resource schema in the \texttt{GPResourceSchema} class. All \texttt{GPResourceSchema} instances loaded from the server are cached in \texttt{GPResourceSchemaCache} to avoid reloading the same resource schemas in subsequent project loading.

G-Portal client applet adopts a lazy approach to load resource instances from the server. When the user opens a project, only the location, id, name and owner of each resource are loaded from the server into the client. The rest of the resource content, including the customized attributes under the \texttt{Content} element, are not immediately loaded. This lazy loading strategy achieves better efficiency, as it is unlikely that the user will examine the content of every resource in the loaded project. Instead, it is always preferred to load resource content only upon the user demands, shortening the time required for loading a project. The strategy also works very well for resources that contain large amount of data. On the other hand, the core attributes such as id, name and owner are almost always required for several operations, such as display of resource and synchronization between the map-based and classification-based interfaces. Caching these attributes in the client eliminates the need to communicate with the server when performing these operations.

Annotation is implemented in almost the same way as normal resources, but with an additional boolean attribute to denote whether it is an annotation. Other digital library services provided by the system, such as visualization and classification, will simply treat annotations as resources by ignoring this boolean attribute.
Because the GeoTools package only provides support of general purpose GIS features, we design additional classes to capture the project/layer/resource relationships, as shown in Figure 8. The GPProject class contains project attributes and a set of instances of GPLayer class. GPLayer in turn has a set of GPResourceGroup instances.

4.3 Database Design

At the G-Portal server end, persistent storage and efficient queries are the key design issues. As shown in Figure 5, the G-Portal server deploys both a Tamino XML Server and an Informix Relational Database Server. G-Portal resources formatted in XML are stored in the Tamino XML server. The XML server supports efficient storage and queries on XML data. As the resource attributes are semi-structured in nature, the XML server can support XPath and XQuery style queries on these resources, which can not be done using
a relational DBMS. In addition to resource contents, resource schemas are also stored in the Tamino server, since they are represented in XML Schemas, which are themselves XML documents.

While the Tamino XML Server can handle the XML style queries on resources, it lacks the support of spatial data types and spatial queries. For example, it is not only cumbersome for XML server to store shape objects, but also highly inefficient to process spatial queries. To address this deficiency, the Informix Database Server is used. Location attributes are stored in the Informix RDBMS, which provides efficient support of storage and query processing of spatial objects. Since location attributes are always retrieved when loading a project, the Informix database is always accessed. To avoid accessing the other server (the Tamino XML Server), the rest of the core attributes are also stored in the Informix server so that they can be fetched together with the location attributes. In this way, the speed of project loading is much faster.

Figure 9 depicts the database schema design of the Informix RDBMS. In the resource DB, each type of resources takes up one table. For example, all country resources are stored in the country table and all exam questions in the examquestion table. The location attribute is stored in the geometry column of each table. The spatial data types of this column depend on the types of resources. For example, we use mulit_polygon for countries and point for cities. The location attributes are indexed using R-Tree, which provides efficient support for queries involving spatial predicates, such as coveredby and intersect. Separating resources into individual tables avoids a single large table containing all resources and allows the R-Tree index to adapt itself to more efficiently support the different spatial data types of different resources.

The resource_info table records all the resource types. It also contains another column to indicate whether this type of resource is annotation. There is no distinction between resource and annotation in the way that they are stored in their corresponding tables. The relationship between annotations and the annotated resources are captured in the table res_ann. This enables direct queries involving annotation and the resources that they annotate.

In the project & layer DB, the tables layer_info and project_info store information about layers and project. The project/layer/resource relationships are determined by the layer_resource and project_layer tables. All classification schemas reside as individual files on disk and the path to each file is recorded in the classschema column of the class_info table, along with other attributes about the classification schema. The project_class table decides what classification schemas are associated with which project. Similarly, XSLT templates are also stored as files on disk with their location recorded in the location column in the xslt_info table. Each type of resource can be associated with an XSLT template in the resource_xslt table. The project_xslt table determines
the resource/XSLT relationship to be used in a project and the default XSLT
template for that type of resources.

5 Resource Content Visualization and Editing

In this section, we discuss issues related to the visualization and editing of
resource content. The use of XML based resource representation poses chal-
lenges in visualization and editing. In particular, we will highlight the issues
pertinent to XML resource content transformation and XML content editing.

5.1 Transformation of XML Content

Directly displaying raw XML contents sometimes may not be the preferred way
to visualizing resource contents because users who are not familiar with XML,
especially elementary or high school students, may find it difficult to interpret
the nested content of XML resources. To provide users with alternate views
of resource contents, we have implemented a mechanism that allows XML
contents to be transformed into other formats that are more appealing to users.
XSLT has been chosen as the underlying template language for specifying
the transformation rules. In the current implementation, we have included
templates that transform XML into HTML format, which users are much
more familiar with.

XSLT templates are configured by project administrators when defining projects.
Each type of resource may have one or more XSLT templates associated
with it, representing different ways of transforming the resource content. In a
project, for each type of resource, one of the templates is configured (by the
project administrator) as the default. Users can choose to view the content in
its raw form or after applying the default or selected template. Figure 2(b)
shows the transformed version of the resource in Figure 2(a) using one of the
XSLT template.

We use the XSLT engine comes with JDK1.4 to perform the transformation.
Since we have to use Java 1.1 in the client, XSLT transformation is performed
on the server side and only the transformed content is transferred to the client
applet. If the transformed format is HTML, the client applet opens a new Web
browser window and writes the HTML content to the window using Javascript
codes generated by the client applet. Communication from Java to Javascript
is done through LiveConnect 4 with the netscape.javascript package, which

4 http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/livecon.htm
is implemented in both Netscape and Internet Explorer.

5.2 XML Editing

To allow users to create and update resources (including annotation resources), it is necessary to have an user friendly XML editor. Existing commercial or public XML tools do not provide their editing components as independent reusable modules. Our decision with regard to this issue is to implement our own module, called the XML Editor. The XML Editor is designed to be a component that can either be used within the G-Portal or as an independent tool for other XML editing tasks.

As shown in Figure 10, the XML Editor adopts the familiar Windows Explorer metaphor where the left panel is the XML tree with element names as node names and the right panel displays the content of the selected element. Attribute values and text values of the currently selected element can be entered in the textboxes in the right panel.

The XML editor is schema guided, meaning that it requires a resource schema to guide the editing of an XML resource in terms of its element structure. A schema guided editor has the advantage of being able to create and modify appropriate element structure during the editing process. The editor first parses the XML Schema and generates a base skeleton of the XML content with empty value for each element. The user can then fill in the values of the elements. Insertion and deletion of elements is only allowed when it conforms to the given schema. Before saving the edited XML content, the editor verifies that the filled values satisfy all the constraints in the schema. This procedure not only ensures that the resource content created is valid (conform to its
Figure 11 shows the class design of the XML Editor. The design of the Generic-Tree and related classes is a simplified version of that of the JTree in the Swing package. The XMLModel extends the TreeModel base class and overrides the important methods to implement functionalities for parsing XML Schema into a base skeleton and generating XML element instances given the parsed XML Schema. Each node in the skeleton tree is represented by the XMLModelNode class, which contains one XML Schema element and one XML element. The classes for dealing with XML Schema and XML documents are borrowed from the Oracle XML Development Kit (XDK) for Java. The Oracle package is chosen because it provides Java classes for representing XML Schema, which most existing Java libraries for XML do not support. The EditorFrame instantiates a GenericTree object with XMLModel and XMLModelNode instances. EditorFrame also contains a ElementEditPanel which is the right hand side panel for editing the content of the current element. The ElementEditPanel class receives an XMLModelNode instance from the EditorFrame whenever the selection in the skeleton tree changes. The XML Schema element in this XMLModelNode instance is then used to create labels and textboxes for users to fill in values and attributes of the corresponding XML element instance being edited. The filled values are stored back to the XML element instance in the XMLModelNode instance.

Fig. 11. Class Diagram of the XML Editor

schema) but also makes the editing process easier for users.

6 Resource Classification

As described in Section 2, G-Portal provides a flexible scheme for classifying the resources according to user-defined classification rules [11]. This section discusses how the classification scheme is implemented in the system.

Resources are classified according to classification schemas that dictate how resources are assigned into categories and how the categories are organized into some taxonomy. Each classification schema can be expressed in a specially designed classification language, as illustrated in Figure 12. In the classification schema shown in Figure 12, we group the exam question resources by the value of the /Resource/Content/Subject/Topic element of each resource and assigning them into one of the categories “Natural vegetation”, “Coasts”, “Rivers” and “OtherTopics”. This classification rule will only be applied to resources that have a value of “Physical Geography” in its /Resource/Content/Subject/Part element. The taxonomy definition portion of this schema is not shown here. Interested readers can refer to [11] for the complete definition of the classification language.

Given a classification schema, the classification engine automatically performs the category assignment of resources and constructs the taxonomy accordingly.\footnote{In our implementation, G-Portal uses another classification schema compiler to convert a classification schema into its XML equivalent.}

The class design of the classification module is shown in Figure 13. A set of classes (in the top half of Figure 13, originated from the ClassificationSchema class) have been designed to represent the classification schema and its various components. As there could be different actions (e.g., language syntax validation, classification of resources, etc.) to be applied on these classes, the Visitor pattern [6] is used to keep the set of classes remain stable while allowing new actions to be added. The ClassificationSchema and related classes all have one single method accept() that takes in an instance of the Visitor interface as parameter. Actions such as syntax validation (ValidationVisitor), schema con-
version (FromLanguageVisitor), and resource classification (ClassificationVisitor) are implemented as visitor classes inherited from the Visitor interface. By passing a different Visitor instance to the accept() method, different action can be performed. This eliminates the need for each of the ClassificationSchema and related classes to have one method for each action (e.g., validate(), fromlanguage(), and classification()) and makes adding new actions as easy as creating new subclasses implementing the Visitor without altering the entire set of classes.

The I/O design allows different sources of input and targets of output repository to be specified. The CSEngine class accepts the generic base I/O interfaces. Concrete classes implementing these interfaces can be supplied to the classification engine component. For example, the resources to be classified by the classification engine may be DOM Element objects instantiated from the data given by the user using the client applet (CSElementInput) or resources directly fetched from the Tamino server (CSTaminoInput). In fact, the XML
editor introduced in Section 5.2 adopts a similar I/O structure, which allows it to be used within the G-Portal client applet as well as a standalone tool.

7 Resource Query Processing

7.1 Resource Query Language

As shown in Figure 3, the G-Portal query frontend allows users to specify search criteria on resource attributes and bounding boxes for spatial query conditions. To support these unique features, a new query language, RQL (Resource Query Language) has been developed for G-Portal resources. RQL is a XQuery-like language with spatial extensions. All queries in G-Portal are first expressed in RQL before they are given to the server program for evaluation. For example, the query specified in Figure 3 will be expressed in the following RQL statement:

```rql
select GCEExamProj
for $i schema(ExamQuestion)/*/Resource
where $i/Content/Description ~= "*USA*"
return $i
```

The RQL statement consists of four clauses. The `select` clause specifies the G-Portal project to be queried. The `for` clause identifies the resources to be queried (those with the resource schema `ExamQuestion` in this example) and iterates through these resources, binding them to the variable `$i` one at a time. The `where` clause filters unwanted resources by specifying the filtering expression on the binding variable. In the above statement, we only keep resources whose `Description` element contains the string “USA”, as specified by the user in the frontend. Spatial predicates such as `coveredby` and `intersect` can be used as part of the filtering expression, enabling us to query based on spatial attributes of resources. For example, a sub-expression `coveredby($i,Polygon((-95,30),(-95,50),(-70,50),(-70,30)))` can be added to the `where` clause with a conjunction to retrieve only resources located within the given polygon. Finally, the `return` clause generates the resources for each of these bindings fulfilling the filtering expression.

7.2 RQL Query Processing

The G-Portal query processor resides in the server program. It is responsible for parsing a given RQL query statement, determining a query evaluation plan and executing the plan. As the G-Portal resources are stored in both the
Tamino XML Server and Informix Database Server, and the former does not provide good spatial query support, the RQL query processor must derive a plan that decompose a query into two subqueries, one for XML Server and another for Informix Server. The results returned from these two subqueries will later be merged into the final query result before being returned to the query frontend.

The Java classes required for implementing RQL queries and query processing steps are shown in Figure 14. The RQLQuery class contains an RQL query statement in its raw query string format and in the form of a parsed query tree. The query tree is represented by the RQLQueryTree class together with the RQLQueryTreeNode class. It is produced by the RQLParser class by parsing the RQL query statement. The RQLParser class is a parser generated by the CUP ⁷ library using a yacc equivalent specification written in CUP syntax. The RQLParser class also requires the RQLScanner class to perform lexical analysis. The RQLScanner class, although implements the Scanner interface from the CUP package, is generated by another package called JFlex ⁸ using a lex equivalent specification written in JFlex syntax. With the parser (RQLParser) and the lexical analyser (RQLScanner), a given RQL statement can be validated and parsed into a query tree. As a RQL query has to be decomposed into two subqueries, two subclasses of the RQLQuery class are created, namely TaminoQuery and InformixQuery. They are responsible for transforming the query tree into query strings to be processed by the Tamino XML Server and Informix Relational Database Server, respectively. They also retrieve the query results from the two database servers and pass them back to the RQLQuery class, which then merges the results and sends them back to the client.

Fig. 14. Class Diagram for RQL Query Processing

\[\text{server.query}\]

 pared() \[\text{RQLQueryTreeNode}\]

 \[\text{TominoUtil}\]

 

\[\text{JCUP package}\]

 \[\text{lr_parser}\]

 \[\text{Scanner}\]

 \[\text{RQLquery : string}\]

 \[\text{RQLQuery}\]

 \[\text{RQLParser}\]

 \[\text{RQLScanner}\]

 \[\text{RQLQueryTree}\]

 \[\text{InformixQuery}\]

 \[\text{TaminoQuery}\]

 \[\text{DBUtil}\]

 \[\text{InformixUtil}\]

 \[\text{RQLQueryTreeNode}\]

 7 http://www.cs.princeton.edu/appel/modern/java/CUP/

 8 http://www.jflex.de/
8 Conclusions and Future Work

In this paper, we give an overview of a unique Java-based Digital Library Portal known as G-Portal. As a geography resource portal, G-Portal offers both a map-based interface and a classification-based interface to organize and display metadata resources that describe public domain Web resources in the geography education domain. The metadata resource representation format adopted by G-Portal is based on XML which is ideal for representing information with flexible structures and for information exchange. We also introduce the concepts of project and layer to organize G-Portal resources for supporting different learning needs.

G-Portal consists of several important functional modules including the visualization, user management, project management, resource management, query engine and resource annotation modules. To realize these modules, Java has been used to implement both the client applet and server program of G-Portal. This paper describes some essential Java packages and classes that have been developed, and the design decisions made during the portal development. The system is accessible from the G-Portal website [5].

At present, the G-Portal project has constructed several geography-related resource collections including high school geography examination questions, resources harvested from the DLESE project, country resources and others. To establish a more systematic approach towards selecting, editing and sharing quality resources, a G-Portal collection building team consisting of educators and researchers in the geography domain will be formed in the near future.

As part of our future work, G-Portal will be further extended with more functional modules to support e-learning and community building. Pedagogical studies will be conducted to ensure that G-Portal can support the learning models used in the classroom environment. The notion of personalized project space will also be introduced to support individual students’ learning needs. Within a personalized project space, a student can configure a project’s content and user interface tools to suit his/her current interests allowing him/her to use the project as a reference in the learning process.

Acknowledgements

This work is funded by the SingAREN Project M48020004.
References


