The Web-Interfacing Repository Manager: A Framework for Developing Web-Based Multimedia Applications for Medical Research

Rex Jakobovits, PhD[1,2,4] Stephen G. Soderland, PhD[1,4] Ricky K. Taira, PhD[3,4]
James F. Brinkley, MD, PhD[5]

[1]Department of Radiological Sciences
University of California, Los Angeles

[2]Structured Reporting Systems
Seattle, WA, U.S.A.

[3]Department of Radiology
University of Washington, Seattle, WA

[4]Department of Radiology
Children's Hospital and Regional Medical Center, Seattle WA

[5]Structural Informatics Group
Department of Biological Structure
University of Washington, Seattle, WA

Abstract- There is a growing demand among medical researchers for tools that simplify the process of managing large collections of multimedia data. In response to this demand, we have developed the Web Interfacing Repository Manager (WIRM), a framework for modeling, acquiring, managing, and navigating through multimedia information. The system allows a programmer to model domain knowledge as context-sensitive views, from which web pages are dynamically generated in response to user queries.

WIRM is a Perl-based application server that provides a high-level programming environment for building web-based medical information systems. WIRM consists of an object-relational database and a suite of Perl interfaces for visualizing and integrating heterogeneous multimedia data. The system provides facilities for creating context-sensitive views over a multimedia database, allowing developers to rapidly build dynamic web sites that adapt their content and presentation to multiple classes of end-users.

1. Introduction

Medical research is becoming harder to manage due to four basic trends: exponential increases in volume of data, proliferation of data formats, migration towards distributed heterogeneous systems, and increased collaboration [1]. These trends result in experiments that require a far more sophisticated data management process than their predecessors. There is a growing demand among medical researchers for tools that simplify the process of managing large collections of multimedia data [2]. In response to this demand, we have developed the Web Interfacing Repository Manager (WIRM), a framework for modeling, acquiring, managing, and navigating through multimedia information. The system allows a programmer to model domain knowledge as context-sensitive views, from which web pages are dynamically generated in response to user queries.

WIRM is a Perl-based application server that provides a high-level programming environment for building web-based medical information systems. WIRM consists of an object-relational database and a suite of Perl interfaces for visualizing and integrating heterogeneous multimedia data. WIRM provides facilities for creating context-sensitive views over a multimedia database, allowing developers to rapidly build dynamic web sites that adapt their content and presentation to multiple classes of end-users.

This paper describes a new type of middleware, which enables the rapid development of web-based systems for managing research data and workflow. A design methodology for modeling experiment data as hierarchical views is

described, which provides a framework for the rapid development of drill-down navigation systems that adapt themselves to the context of the user.

We have used WIRM to build experiment management systems for a neuroscience laboratory [3], which manages the workflow of large groups of collaborating scientists. Patient demographics, MRI exams, surgeries, intra-operative photographs, behavioral experiments, and 3D brain models are all hierarchically modeled as WIRM schemas and views. Multiple privacy contexts are supported, allowing public access to published data, while protecting unpublished data and enforcing patient privacy.

WIRM is also being used to manage the clinical data generated by a natural language processor of radiology reports [4], and to support the creation of web-based teaching file repositories.

2. Architecture

The components of the WIRM architecture are shown in Figure 1. *Wirmlets* are executable CGI scripts that encapsulate chunks of logic, either supplied by the system, or defined by the user. Wirmlets generate *Web Views,* which are HTML documents passed back to the user's browser. The WIRM *Server* handles requests from the Wirmlet.

Class Definitions are pieces of structured code that specify how repository objects should be viewed and manipulated. The database server handles SQL queries and accesses a standard relational database. File data can be stored in a protected *Files Storage Area*, either protected by the repository or any files on the server machine. Files are copied on demand to the *Visualization Cache*, a Web-viewable area for staging files.

When interacting with the system, the end-user submits a request through the client browser, transmitting the URL of the target Wirmlet, the Session State (including form state, the user ID, and active object identifiers that are participating in the transaction).

The Wirmlet makes requests to the WIRM server, which translates the request into SQL and passes it to the Database Server. The results are returned and wrapped in HTML, so they may be inserted into the Web View document. Other requests may retrieve Files from the File Data area, which may cause them to be converted into Web-viewable objects in the visualization cache, which are then included in the Web View. Further requests may access methods of the Class Definitions, which generate pieces of the Web View document. Finally, the completed Web View is returned to the Web server, which in turn passes it back to the waiting Client Browser.

3. Server Components

The WIRM Server is built from a layered architecture of components, each of which handles a specific function, as shown in Figure

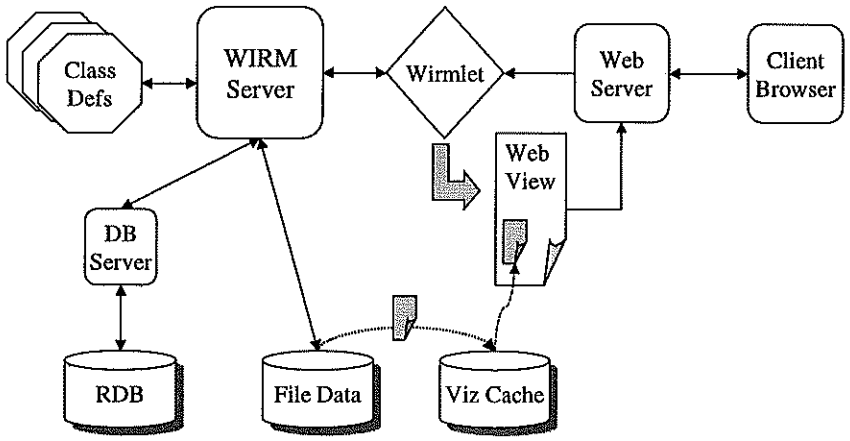

Figure 1: WIRM Architecture

2. There are six component libraries divided into two categories: the *Developer Interfaces,* which includes the Gateway, Repository Object Interface, and HTML Generator, and the *Internal Interfaces,* which include the Table Manipulator, the FSA Controller, and the Visualizer. The Developer Interfaces are used by the designer to define Wirmlets and the methods for the domain-specific Class Definitions, whereas the Internal Interfaces are used by the WIRM developers to encapsulate access to the various repository resources.

3.1 FSA Controller

The FSA Controller regulates access to the File Storage Area (FSA), which is an internal repository of files managed by WIRM. There are three classes of file handled by WIRM:

- FSA Files, which are physically managed by WIRM in the FSA
- Local Files, which exist on a file system directly accessible by WIRM's server but not in the FSA.
- Remote Files, which have been registered with WIRM but are maintained on a remote machine.

Using the FSA Controller, files may be copied into the storage area from a file handle, and file locations may be looked up based on file id. Other than file location, the interface does not maintain any file metadata. That task is managed by the Repository Object Interface, which builds an additional level of abstraction on top of the FSA Controller, allowing the Wirmlet developer to treat files as objects complete with metadata rather than just blobs in the file system.

3.2 Visualization Cache Manager

Files managed by WIRM can reside in the File Storage Area or in other locations accessible by the WIRM server. These locations are not necessarily accessible from the Web, however. The Visualization Cache Manager (VCM) provides Web access to multimedia files managed by WIRM by maintaining a Web-accessible cache (called the ``Viz Cache'') on the server machine. When a user requests a file through a WIRM-created Web page, the system uses the VCM to convert the file into a Web-viewable type and then copy the file into the Viz Cache. If this process has already occurred for

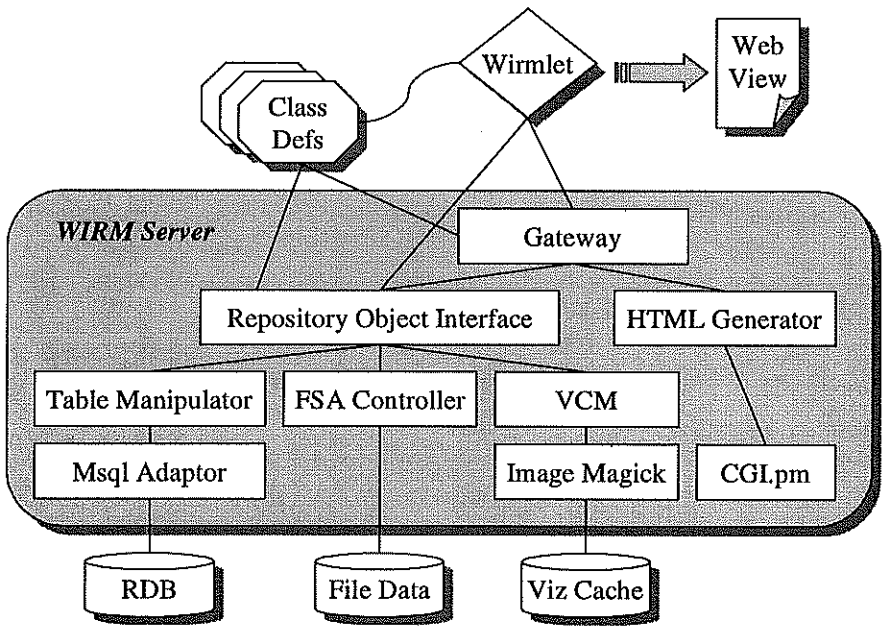

Figure 2: WIRM Server Components

the desired file, the existing copy is used.

The conversion process is automatic: when a file is requested for Web viewing, the VCM determines a file's type and then converts it to a Web-favorable format using a built-in conversion routine or a user-supplied module. The ability to supply user-defined routines is analogous to the use of datablade modules by Illustra [5] for extending their type system. For example, the Brain Mapper extends the VCM to include a module for converting MR images to JPEG's.

If a file is of an unrecognized type or is already in a format acceptable by the browser, the VCM copies it from its source location into the Viz Cache and provides a hyperlink to the unchanged copy, which the user may retrieve through their browser.

3.3 Table Manipulator

The Table Manipulator Interface provides functions for creating and deleting tables, inserting and removing records, accessing rows of object attributes by ID, formulating SQL queries, and retrieving the results of a query into a tabular data structure accessible by the Wirmlet manipulation language.

The Table Manipulator connects to the relational database server and exports a table-level interface to repository data. It handles SQL-like queries from the calling environment, which return a *statement handle* from which results can be retrieved one at a time using a standard cursor traversal mechanism.

3.4 Repository Object Interface

The Repository Object Application Programmer's Interface (REPO API) provides an object-relational data model to the Wirmlet developer by abstracting away the relational statement handles of the Table Manipulator and allowing the data to be viewed as collections of objects that conform to the Repository Object model as defined in the previous chapter. Each class is represented by a table in the relational database whose columns match the schema attributes, and whose rows comprise the instances of that class. By using the REPO API, the Wirmlet developer operates on object-oriented data structures rather than tables, navigating through networks of objects and their attributes rather than cursing through rows of data.

The REPO API is a collection of functions that allow the programmer to define new object types, import or create instances, edit existing instances, manipulate sets of instances, and pose queries over the data. An object may have more than one physical component, but the Object API allows the Wirmlet developer to refer to the object as a single entity. For example, an MR-image object consists of a file in the FSA and descriptive information in a database table. A request to retrieve an image by name would invoke an Object API function that looks up the image ID in the File Description Table, then retrieves the appropriate file from the FSA.

In addition to providing a convenient interface for the Wirmlet developer, the REPO API enforces consistency between the FSA and the associated metadata in the DBMS.

3.5 HTML Generator

The HTML Generator is a developer's interface that encapsulates some common user-interface constructs in high-level functions, which emit HTML strings, allowing the rapid development of Web documents. It is heavily used in constructing the Web View, by the View Methods of Class Definitions, and by the Wirmlets themselves. There are no methods in the HTML Generator that depend on the repository object model, so this API can be used independently of a WIRM repository. User interface methods that deal directly with Repository Objects can be found in the Gateway Interface.

The HTML Generator provides a suite of functions for creating and parsing interactive form elements (e.g. popup menus, scrollable lists, etc.), and other shortcuts for generating HTML syntax (e.g. turning an array into a

formatted HTML table, handling document layout, displaying a thumbnail image, etc.).

3.6 Gateway Interface

The Gateway Interface provides a high-level interface for displaying repository objects as Web pages. There are generic methods for viewing objects as labels, rows, and full pages. The Gateway provides methods for generating HTML tables of objects, and for generating form elements for choosing from a list of objects. There are methods for creating JPEG versions of image files, and for creating thumbnails and clickable image maps.

The Gateway is used along with the HTML Generator for developing Wirmlets and for defining the View Classes of repository object types.

4. Methodology

A web application is defined by its domain knowledge and domain data. We have defined the Query-By-Context view model [6], in which domain knowledge is encoded in two forms: Class Definitions (Schemas and Methods) and Custom Wirmlets. The domain data is encoded as instances in the database and files in the repository. As part of the system design, domain experts articulate their domain knowledge to a system developer, who then uses WIRM to encode the domain knowledge as Class Definitions. In some cases, the domain experts may perform some of the encoding themselves, especially with regard to schema design.

The task of building a research management application is really a data integration problem, as it involves resolving schematic and semantic conflicts across knowledge representation systems [7]. A major role of the system is integrating heterogeneous systems and providing a uniform interface for accessing data from multiple sources. Each external application that interfaces with the system can be seen as an information producer or consumer. Some sources provide data, others provide knowledge, and many provide both knowledge and data. For each source, the key is to integrate the source's knowledge into a consistent framework, and map the data into the model using the knowledge framework as a guide.

The WIRM Development Methodology first requires the developer to work with the domain experts to develop a canonical modeling of real world concepts (knowledge) using the Repository Schema Model [8], and then map existing information systems to the canonical model, which facilitates the integration of data entities across systems.

In the WIRM Development Methodology, it is recognized that schema and class definition must occur as a process of stepwise refinement. An application is constructed in many interleaving stages of experiment design and execution, and the classes and Wirmlets will naturally evolve as the developer's understanding of the experiment data matures.

The methodology consists of the following steps:

- Designing Schemas in terms of the Repository Schema Model
- Modeling context-sensitive Class Definitions as defined in the Query-By-Context View Model
- Developing custom Wirmlets to interact with users and external applications
- Evolving schemas and Wirmlets as the EMS matures

The domain experts should work with the developer to identify the salient concepts in the domain, and to define attributes for each of those concepts in terms of atomic, composite, or aggregate types as allowed by the Repository Schema Model. Composite types can be system-defined or user-defined, as allowed by the object-relational data model. This process often involves observing the experiment data and classifying it according to semantic ontologies. It is often useful to pick a specific concept to be the "central concept" and have all other concepts relate to it hierarchically.

It is common for two concepts to merge after awhile, or for a concept to split into separate concepts. The schema evolution capabilities of WIRM facilitate this.

Once the concepts and their attributes are identified, they should be implemented using the built-in point-and-click Schema Definition Wirmlet. The Schema Definition Wirmlet supports the creation of attributes that conform to the Repository Schema Model. A drop-down menu enables the user to choose from the allowable atomic types (strings, integers, etc.) or to choose a reference to a composite type. Once schemas have been defined, the repository is immediately ready to accept new data objects of these types.

The process of modeling classes involves creating definitions that adhere to the form defined in the Query-By-Context View Model. Class definitions are composed of methods that can be View Functions, Make Functions, Edit Functions, or Delete Functions. These are implemented as specially formatted Perl functions in a Class Definition file. The contents of the Class Definition file make up an important part of the Repository State, which is part of the Session State that determines the contents of every Web View. By adhering to this model, the designer is given a natural framework in which to create context-sensitive interfaces. By defining the prescribed View Functions, the system automatically enables a Virtual Navigation Space of drill-down visualization interfaces, as will be demonstrated in the examples which follow. Moreover, by defining the prescribed Edit-Fns and Make-Fns, the Edit Object and Make Object Wirmlets automatically enable a hierarchical workflow management interface.

WIRM provides a template for creating domain-specific Wirmlets, which has the following basic structure:

1. Initialize the HTML form, and display banner and title.
2. If no submit button has been pressed, display a prompt.
3. Otherwise, process the results.

Some Wirmlets have more a complex structure, such a multiple interactive processing stages, but they all follow this general format.

Once the web application has domain-specific schemas, custom class definitions, and a set of custom Wirmlets to handle the Workflow, the system is ready to support real users. Inevitably, system will become out of date as users request new features in the interface, or the experiment itself evolves. At this stage, the WIRM's ability to support dynamic schema evolution and class modification is called upon.

Another common process as the system matures is to customize the interface for multiple user classes. This is accomplished by performing a test on the user context within the Wirmlet code, and designing the behavior of the Web View to be context sensitive using conditional branches.

5. Conclusion

WIRM is being used to support several large-scale research projects, and has demonstrated its effectiveness as a tool for building web-based multimedia applications for medical research. The Brain Mapper Experiment Management System [9], shown in Figure 3, is being employed on a daily basis by over a dozen researchers at the University of Washington, and is continually being adapted as the experiment evolves. The project's goal is to develop an information framework for managing cortical stimulation data obtained during neurosurgery. The experiment requires fine-grained collaboration and data sharing among radiologists, neurosurgeons, neuroscientists, statisticians, computer vision experts, database administrators, and a number of technicians, students, and assistants. Functional brain mapping data consist of thousands of ordered MRI slices grouped into exams, 3-D rendered brain images, digitized photographs, lists of identified site coordinates, and alphanumeric tables of patient demographics. A wide range of heterogeneous software applications is called upon to interact in a complex workflow process [10].

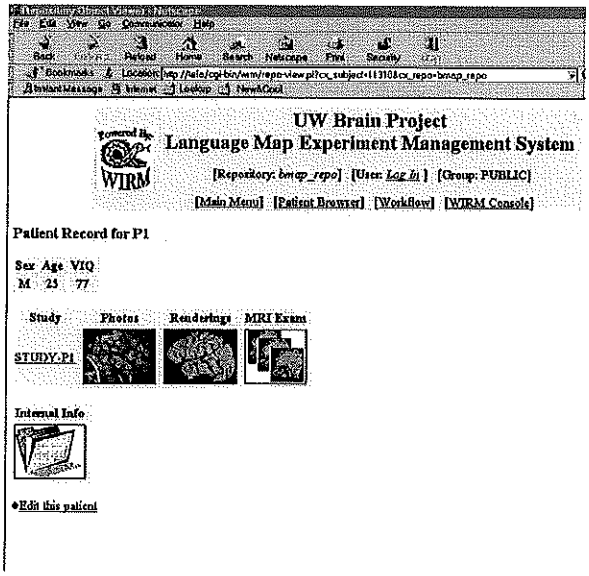

Figure 3: The Brain Mapper EMS

In addition, WIRM is being used to support a project to develop Natural Language Processing tools for structuring radiology reports. WIRM provides hierarchical access to the intermediate data structures generated by the NLP algorithms, and supports querying over the text of the reports and the structured knowledge bases.

Finally, WIRM is being used to power MyPACS.net, a service that enables radiologists to create their own web-accessible teaching file repositories by uploading images through a web browser. Authors are prompted to enter descriptive information about a case, including diagnoses, imaging modalities, radiographic and pathological findings. Once a teaching file had been created, it becomes indexed by a WIRM-based Teaching File Search Engine, and can be retrieved according to search criteria on any parameter, including structured keywords and full text matching. WIRM allows the author to regulate access to each teaching file by protecting it with a password.

WIRM is available for download at http://WIRM.org.

Acknowledgements

This work has been funded by NIH SBIR grant R43-MH61277-01 and Human Brain Project grant DC02310.

References

[1] Jakobovits R, Soderland SG, Taira RK, Brinkley, JF. Requirements of a web-based experiment management system. Submitted to *AMIA Fall Symposium, 2000.*

[2] Ioannidis Y, Livny M, Gupta S, Ponnekanti N. ZOO: a desktop experiment management environment. *VLDB*, pp. 274-285, 1996.

[3] Modayur BR, Prothero J, Ojemann G, Maravilla K, Brinkley JF. Visualization-based mapping of language function in the brain. *Neuroimage*, 1997: 6: 245-258.

[4] Taira RK, Soderland SG. A statistical natural language processor for medical reports. *AMIA Fall Symposium*, 1999.

[5] Stonebraker M. Object-relational DBMSs: the next great wave. Morgan Kaufmann, 1996.

[6] Jakobovits R, Brinkley JF. Query-By-Context: A framework for modeling and navigating multimedia research data. *AMIA Fall Symposium,* 1998.

[7] Kim W, Seo J. Classifying schematic and data heterogeneity in multidatabase systems, *IEEE Computer*, December 1991.

[8] The Repository Schema Model. Technical Report 99-103, Structured Reporting Systems.

[9] Jakobovits R, Brinkley JF. Managing medical research data with a web-interfacing repository manager. *AMIA Fall Symposium,* 454-458, 1997.

[10] MyPACS.com: a service for creating web-based teaching file repositories. Submitted to *RSNA InfoRad 2000.*