2008-03-12

Extending Web Application Development to the User-Editable Space

Brian S. Goodrich
Brigham Young University - Provo

Extending Web Application Development to the User-Editable Space

by

Brian Goodrich

A thesis submitted to the faculty of

Brigham Young University

In partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Brian Goodrich

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory

Date	Dr. Phillip J. Windley, Chair
Date	Dr. Daniel Zappala
Date	Dr. David Embley

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Brian Goodrich in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Dr. Phillip J. Windley Chair, Graduate Committee

Accepted for the Department

Date	Dr. Parris Egbert Graduate Coordinator

Accepted for the College

Date	Dr. Tom Sederberg, Associate Dean College of Physical and Mathematical Sciences

ABSTRACT

Extending Web Application Development to the User-Editable Space

Brian Goodrich

Department of Computer Science

Master of Science

The growth of the web increased dramatically when users were provided with applications that let them use just their browser to post and edit content on the World Wide Web. Offering users the ability to use their browser to create their own web applications, instead of just posting text and images, would cause another Internet evolution. This thesis describes the EXPPO system (Extensible Page Productions and Operations), a web application development environment for both end-users and technical-users. EXPPO leverages the end-user's previous experience with internet browsers by using a page based development experience or a Page Oriented Architecture. Because applications are structured in this architecture, components used in one application can be re-used by another, providing end-users with functionality that was created by more technical users. This thesis demonstrates how the EXPPO development environment can be used to create functionally rich web applications.

ACKNOWLEDGMENTS

This work would not have been possible without the support of many people.

First and foremost, I would like to thank my wonderful wife, Ashalon, for her support and patience during the development of this work.

I want to thank my advisor, Dr. Phil Windley for the inspiration and support to pursue this project.

I want to thank Dr. Daniel Zappala for assistance and instruction in writing this document and expressing the concepts involved with it.

I would like to thank my entire graduate committee for their patience while the implementation of this thesis and its supporting documents were produced.

TABLE OF CONTENTS

List of Figures

List of Tables

1. Introduction

The ability to use an internet browser to create and edit web pages has revolutionized the web by allowing ordinary people to create massive amounts of content on the World Wide Web. Previous to the creation of user-editable sites, a much more advanced set of technical skills and expensive hardware was required to disseminate information to the users of the World Wide Web. Now, everyone with access to the World Wide Web can post or contribute text and images to wikis, blogs, home pages, forums, and other user-editable areas.

Another revolution is possible by allowing users to create web applications, rather than static content. In general, user-editable web sites only allow users to edit presentation content. While this is acceptable to provide text or images for a reader that is the extent of presentation content's inherent capabilities. The problem is that users can only create content that is meaningful within the domain of a particular application. Providing users with a standardized method by which they could create their own web applications would allow users to customize the Web to fulfill their specific needs without limitation and a more functional web could scale the same way that the current user-editable sites have with simple web pages, blogs, forums and wikis.

With a standardized application creation framework, users could borrow and reference pre-existing work to create their own web applications. To make this possible, users require a browser-based application development framework that supports structured data and a way to organize user-extensible functionality. Using a structured data

format means that information can be easily queried by internal and external systems for re-use, sharing, aggregation, analysis and display. This provides users with facilities for using large amounts of data without necessarily having to store and maintain the entire bulk of it themselves. With a modular organization, applications can execute composable logic that can be can be exchanged with other applications, and web application authors will no longer be required to create specific computer logic themselves. Thus, a user will no longer need years of technical training to create a web application. For these reasons, this framework requires the ability to allow users to harness functionality written by themselves or others, to have the capacity to exchange structured data and to provide metadata support for third party applications.

To make the creation of these web page applications understandable to end-users, this framework should use an architecture that is conceptually familiar to these users. The user's internet browser experience is based on the concept of pages, therefore intuitive construction of a web page application should also use a page based architecture [Quint][Gardner]. To meet this requirement, this thesis introduces a framework that uses a Page Oriented Architecture. A Page Oriented Architecture is a web focused application design methodology that uses *pages* as the fundamental unit of organization. These units can be used in combination to create web applications. A page contains functional handlers that dictate the behavior of the web page that it represents, and the metadata, content and data that comprise that particular piece of a web application.

In this framework, pages of a web application must be easily machine readable to allow other application authors and logic modules to interact with the data that belongs to a page unit. To provide this automated access to logic, metadata, data and content, the page units need to be stored in some kind of structured format [Di Iorio]. For the required structure, this thesis uses existing semi-structured W3C standards to allow easy integration with the browser interface. The Extensible Hyper-Text Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax. XHTML enables the use of the HTML browser language for user presentation and the use of XML to structure the logic, metadata, content and data of a given page unit. This structure enables this framework to present data with context, to insert that data into presentation content, and to support a simple data query interface that can serve both logic modules and third party developed applications.

This thesis describes the EXPPO system (Extensible Page Productions and Operations), a development environment for building web applications that are structured, modular, extensible, and still support the presentational strength expected of the World Wide Web. The EXPPO system hosts a browser development environment, allowing users easy access to the ability to create and edit custom web pages. By using a modular logic system and a simplified data query system, EXPPO can support custom and packaged application logic in both local and remote environments. Using the current format standards of XML and XHTML and a Page Oriented Architecture, EXPPO can create web applications that have an inherent, accessible structure, and still retain the user expressiveness of HTML.

This thesis demonstrates how the EXPPO development environment can be used to create functionally rich web applications that inherently support XML-based interaction. To validate the framework, we created an EXPPO web application that tests each aspect of the framework and the claims made in this document. To demonstrate how ordinary users can use EXPPO to build web applications, we build an Interview Log system that can capture job requirements and maintains a set of comments made about candidates for that position. We use examples from these two applications to explain the mechanics of the EXPPO system.

2. Thesis Statement

A web-based development environment with a Page Oriented Architecture enables users to create and edit web pages to contain XML-formatted data, a user-extensible collection of metadata, and a page handling set of instruction descriptors, in addition to the traditional textual content. The system designed as proof of concept of this work, called EXPPO (Extensible Page Productions and Operations), substantially increases the power of the traditional Wiki metaphor by allowing useful web applications to be built in a user-editable space.

3. Related Work

There has been considerable interest in helping users and developers more easily design and build web applications. A number of companies have developed web application frameworks whose main goal is to allow ordinary users to create web applications without requiring the technical knowledge of programming. Discussed in this section are a spectrum of related technologies which have been grouped into the following three sections: User-editable Applications, Structured Web Architectures, and Code Frameworks.

3.1 User-editable Applications

The origin of the user-editable web begins with the wiki. A wiki is an online application that gives users the ability to create and edit web pages through a browser interface. The first wiki, called the WikiWikiWeb, was an application created by Ward Cunningham, thereby giving him credit as the creator of the Wiki (or WikiWiki) software concept and arguably, the inventor of the user-editable web site. Cunningham named the WikiWiki after the Hawaiian term he learned on his first visit to Hawaii. The “wiki wiki” or “quick” shuttle buses at the Honolulu Airport provided an alliterative replacement for the word quick and allowed Cunningham to name his software concept WikiWikiWeb instead of quick-web [Cunningham].

Wikis have had many uses. The WikiWikiWeb application was created as a forum for the design pattern community by adding an automated supplement to the Portland Pattern Repository [Cunningham]. The wiki soon became increasingly recognized as a way to develop and express a knowledge base. Wikis began to be used as

collaborative forums for software development. Although originally used for project management, workflow charting, and code discussion, other non-technical users soon joined the wiki movement as well. The concept has continued to grow in popularity and although it is difficult to catalogue the number of wikis operating on the web, it is obvious from a casual search that wikis are very much alive and active on the Internet today [Schwall]. As the wiki metaphor has continued to develop, further steps have been made towards the concept of a World Wide Web, which has not only open read, but also open access writing ability. Sometimes referred to as the "Writeable Web," this open write web is a vision of a medium where free expression is more than just text, but also a sharing of data and functionality.

A series of extensions to the original wiki concept have been implemented in a number of wiki variants. The WikiMedia foundation was one of the earliest contributors to the wiki concept. MediaWiki is a large open source project for wiki meta-applications such as Wikipedia, Wiktionary, Wikinews and Wikimedia Home. The WikiMedia organization provides wiki users with presentation templates, document version control and a formatting syntax that allows users to manipulate their text to be more presentable [WikiMedia]. WikiCalc, from Dan Bricklin, is a wiki program that works from a foundation of adding structure to data [Bricklin]. Bricklin describes WikiCalc as "a Web authoring tool for pages that include data that is more than just unformatted prose." WikiCalc's current version is a web-based spreadsheet application that supports only basic mathematical functionality for the web community.

Amaya is a client-side implementation of a user-editable application [Quint]. Amaya was originally created in 1997 as a test-bed and demonstration environment for new technologies developed by the W3C organization. Since that time, it has developed into a stand alone application that serves as a web browser that gives users the ability to create a customized representation of web pages. There are several ways of viewing web pages in Amaya, including a table of contents view, a structure view, a source view, a links view and an alternate view, that assist with adjusting web pages to a user's desires. The customization process involves a series of annotations and Amaya created web pages are stored using XML documents and style sheets associated with those documents. Amaya can publish these customized web pages to share them. Amaya could be considered a fat-client desktop application that extracts documents from the web. Amaya does not have support for web applications that have a persistence layer or to provide customized functionality for web applications.

3.2 Structured Web Architectures

One example of a structured web technology is WebML (or Web Modeling Language), a graphical notation for representing the content, composition, and navigational features of a web application [Ceri]. WebML borrows from the principles of Entity-Relationship diagrams and the Universal Modeling Language. There are three different models in a full WebML specification: the Data Model, the HyperText Model and the Presentation Model. The Data Model most closely resembles an ER diagram, a common database notation, and also is compatible with the UML class diagrams common to Object Oriented Programming. The Data Model is a specification for showing how a given

web site relates to its underlying persistence model. The Hypertext Model is a graphical representation of a web application's page flow and the composition of those pages. Composition or content in WebML is broken into Units of which there are seven predefined types: data, multi-data, index, multi-choice index and hierarchical index, entry, and scroller. These units reside in a page "container" and related to other page "containers" by a series of links, which can point back to other units on the same page or other units on other pages. The Presentation Model is a style specification of a hypertext application, used to determine the look and feel of a web page. Although this three-fold model bears resemblance to the classification of data used by this thesis, it is important to note that WebML is not a development environment, but rather a modeling language used to create specifications for web applications.

IsaWiki was developed to follow the paradigm of the Xanadu project [Di Iorio][Nelson]. IsaWiki is based on a conceptual term that the author's coin as "global-editability." IsaWiki is a wiki development environment for customizing new and existing web pages that has its own proprietary language called IML. IML is a conversion language used as a generic representation of the different document types used on the Web. Its grammar is composed of a few elements: paragraph blocks which containing text and inline elements, tables containing cells, collections of vector-based graphics, and lists. IsaWiki provides a web page editor in the form of an Internet Exporer and a Firefox browser toolbar plugin that is available for download. IsaWiki claims to provide a separation of data and presentation layer in IML, but uses this separation mostly to prevent data loss during document conversion into IML. IsaWiki maintains a version control sys-

tem of original documents and personalized modifications for each user in that has access to the wiki. Essentially, IsaWiki provides a custom layer of presentation on top of what a user would normally conceive of as the web, and provides the ability to share these customizations via a repository with other users. While IsaWiki focuses on providing shareable and customizable views of the web that can introduce new content, IsaWiki is weak in its use of data and does not provide users with persistence for customized content or the ability to introduce new functionality.

Dabble DB is a powerful web-based system that lets you store and manage just about any kind of information and share it over the web. Most of the DabbleDB focus is based in spreadsheet technology (such as automatically generated calendars and data hyperlinks) and they expose data collaboration technologies. The DabbleDB application is very data focused, allowing users to select different methods for exposing data, including charts, graphs, geographic map representation, pivot tables and grouping [Dabble]. DabbleDB is more concerned with exposing these representations to users, and less on custom functionality than the subject of this thesis.

Of the recent entries to the native web development field, JotSpot is the most interesting and is the closest parallel to this thesis [Gonzalez-Reinhart]. JotSpot provides features for users to integrate web applications into a wiki. JotSpot has three fundamental principles about storing data: everything is a page, pages can hold any number of typed properties, and pages are XML. The types of applications that have been hosted by JotSpot include company intranets, document collaboration systems, bug loggers and

trackers, internet video on demand (i.e. Strmz.jot.com), a web-based RSS reader (i.e. Rojo.jot.com) and also out-of-the-box applications such as call loggers, polling applications, help desk wikis and email clients that write directly to a JotSpot user's wiki [JotSpot]. However, while JotSpot does support a logic framework with its selectable and modifiable application templates, these templates have the limitation of still needing to originate from JotSpot developers. Additionally, JotSpot does not strongly emphasize the ability of a user to "own" and reuse structured data or the ability to share it. Since being acquired by Google, there has been little forthcoming about the current state of this application, and the original JotSpot.com website has been shut down.

3.3 Code Frameworks

Other works related to this thesis are code frameworks, systems concerned with the exchange of fragments of code. The goal of these frameworks is not only to provide developers with code libraries to use as tool sets to finish applications more rapidly, but also to educate programmers, and provide safer, more secure logic that has been reviewed by a body of peers. These systems are not concerned with data, providing an execution environment for that code or providing their services to non-technical users.

The CodeSwap service from Microsoft is a desktop application that allows Visual Studio .NET developers to easily connect to other developers and search for code. The author of the code provides a brief summary describing their code and then adds it to their list of shared snippets, which are uploaded to a repository. The search will scan the file names, descriptions and the content of the files to find matches in the centralized

CodeSwap index. When users find the code that they are looking for, they simply download it. Additionally, Microsoft's Visual Studio development studio now supports a search and download client for snippets of code called the Code Snippets Manager [CodeSwap].

The SNIPPETS project, started in the late 1980's, is a repository of useful snippets of C and C++. Over time, it has grown to more than 94,000 lines of code in over 700 separate files. The goal of SNIPPETS was to collect and disseminate the best C/C++ answers to "How do I...?" programming questions. Although the focus of the SNIPPETS project continues to be its archive of C++ source code, it now includes Java, D, Python, Perl, and assembly for embedded systems work. Similar systems are supported in the Zend and Eclipse development environments. Although a very useful forum for exchanging logic between developers, SNIPPETS itself is not concerned with data or exposing an execution environment for users to reference code from its repository.

The CPAN or the Comprehensive Perl Archive Network is a large collection of Perl software and documentation. It supports a large search base for Perl modules and a downloader for that software. The source material for CPAN comes from developers all over the world and "has thrived because of the willingness of programmers to share the fruits of their labor with the community" [Wall et. al.]. The network has been online since 1995 and as of the time of writing, CPAN supports over ten thousand modules, five thousand authors and almost three hundred mirrors [CPAN].

Support for systems that can automatically look-up and download applications, libraries and install them often comes embedded in a user's operating system. An example of this is the Advanced Packaging Tool (APT) that comes packaged with Debian. APT is a front-end for the package management system used by Debian GNU/Linux and its derivatives. APT simplifies the process of managing software by automating the acquisition, installation and update maintenance of software packages. APT relies on repositories to discover software using the apt-search command, while the apt-get command uses repositories to resolve dependencies during installation. The Debian project maintains a repository services containing over 19,000 software packages ready for installation. Additional repositories can be added to the APT sources.list configuration file and those URI repositories will also be queried by APT.

4. The EXPPO System

The EXPPO System is based on a Page Oriented Architecture, in which the basic unit of organization for a web application is a *page* unit. A page unit contains a representation of a web page in three sections: each page has persistent data storage, an interface specification and a set of metadata. The data storage of a page contains the local data being used by the page. The interface specification or content section holds web page formatting, and references to data and functional handlers. Metadata contains a page's identifier, a set of functional handlers that a web page can use, and other information about the page unit. Handler references allow EXPPO users to not only apply pieces of logic to their web applications, but also to call and execute code in other EXPPO sites and to customize the results to their own application. Hence, a page unit possesses accessible data and functions for users to interact with. A cohesive set of page units creates a web application.

In EXPPO, there are three primary methods for using a page unit: Edit, Query and Render. As can be seen in *Table 1*, each of these methods requires a different set of parameters and produces a different output. We discuss these methods in this section, while the implementation choices for each of the components involved are discussed in Section 5.

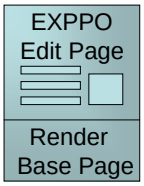
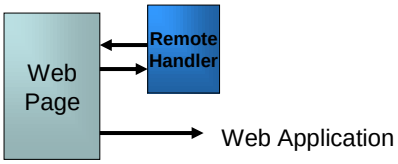
Page Unit Method and Parameters	EXPPO Page Response
Query (string page_id, string QueryStatement)	Query Result
Edit (string page_id, 'EDIT')	
Render (string page_id)	

Table 1 Chart of EXPPO Page Operations and Responses

4.1 Page Method: Edit

The Edit method provides the authoring interface for EXPPO and requires the name of a stored web application file, usually the page identifier of the root page. When EXPPO Edit is requested without a file identifier, a page creation screen is shown, asking the user the name of the web application the user would like to create, if the application does not exist, or edit, if the application requested already exists in the library file of the system. Regardless, once the desired name is submitted, the file in question is opened or created and the user is transferred to the EXPPO Edit Page, shown in *Figure 1*.

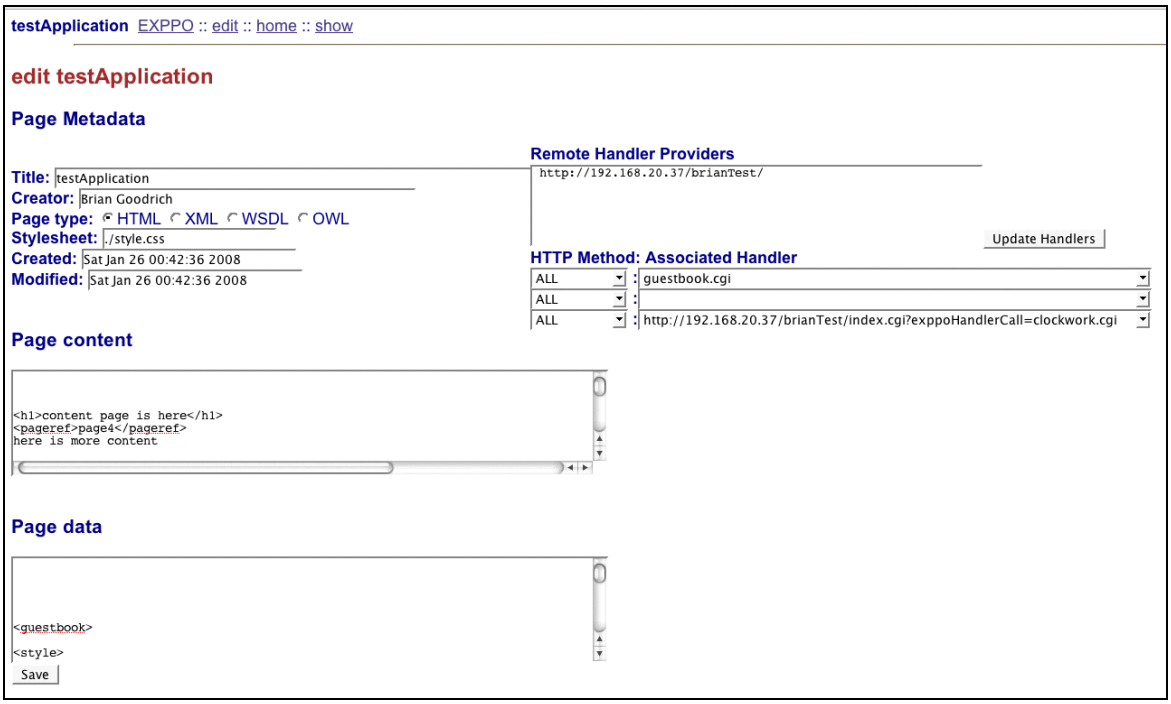


Figure 1 Screenshot of EXPPO Prototype Edit Window

The Edit Window is the web application development environment of the EXPPO prototype and provides the input fields for an EXPPO developer. As can be seen in *Figure 1,* several forms are shown on Edit Window, including the different values for a page's metadata, and text areas for defining a page's content and data. Users can select the functionality a web page uses by selecting from drop down lists of handlers that are known to the EXPPO system hosting the Edit Window.

From this interface, users may edit the values in each of the page units belonging to the file requested. If input inconsistent with the format of the EXPPO meta-language is entered into one of the parts of a page unit, the Edit window will refuse to save the page unit in that state and will display an error next to the appropriate content or data window.

Another input form is provided to enter and store the names of remote EXPPO systems. When an EXPPO developer adds the name of a remote EXPPO system to this list, Edit will add the handlers stored in the handler repository of the indicated system to the lists of known handlers. These new handlers will be added to the Edit Window's drop down lists of handlers available to add to a page unit and become part of its functional behavior. Remote handler specifications in the drop down list include the URL required to access them, thus differentiating them from their local counterparts.

4.2 Page Method: Query

The EXPPO system provides a query system to enable data sharing, exchange of modular components and data access for remote interaction. This query system is a key component of EXPPO that enables selection of *page* units and their components. Because data is owned by *page* units, and a web page is accessed using a URL, the root of any query into an EXPPO system begins with a URL. With *page* units having a structured architecture, a query statement can easily be created that indicates a location and its accompanying data result inside that structure. This is accomplished by adding an EXPPO query instruction and the location statement to the URL. The exact format of an EXPPO query is discussed in Section 5.4.

4.3 Page Method: Render

Rendering an EXPPO page involves all of the components of a *page* unit. The Render method is responsible for accumulating all of the instructions an EXPPO web application has for a given page, interpreting it, and delivering browser compatible instructions. This process involves several steps. The content section of a page can contain embedded page units, references to items in the data section of a page and references to functional handlers. References to handlers and data and embedded page units must be replaced in the set of instructions being sent back to the requesting client.

Page units can contain other page units, and these sections of a page unit must be parsed and converted into page links that a user can click on to navigate to that page. The result of this is that page flow through a web application is built through the hierarchical structure that comprises a page unit. This provides a standardized way to use the structure of an EXPPO web application to navigate from page to page.

Data can also be referenced in the content section of a page unit. These references point to a specific location in the document, indicating that the data that is stored at this location should be inserted where the reference is made. This feature lets data references act like data structures in a programming language, while the data section that the references point to acts almost like a symbol table in many programming languages.

Function handlers are only executed when a page is Rendered. At Render time, the content of a page must be checked for references to handler logic. Handlers referenced in the content section of a page unit point to their execution information in the metadata section of a page. At render time, these handlers will be executed and their result inserted into the location of the handler reference. Therefore, handler processing can actually add references to data, whole embedded page units or even references to other handlers. These handlers may be using data from the originating system, or a different set of data, depending on how the application logic is written. To assist with acquisition of data, EXPPO provides a number of built-in functions for handler logic to retrieve data from EXPPO systems.

Once these steps are completed, the different divisions and forms are assembled in XHTML. The resulting output is returned to the original requesting browser client. Section 6 gives a detailed example of how a Web Application can be built in EXPPO.

5. Implementation

The EXPPO system is constructed as set of Perl modules that are supported by the ATOM API to interface with a DBXML database [Obermeier]. The application has the ability to create structured web applications using XML page units that conform to the EXPPO meta-language. EXPPO supports an XPath based query interface and the ability to interact with local and remote code referred to as functional handlers.

The technology behind EXPPO leverages pre-existing standards and technologies to create a web application development environment that is revolutionary, and compatible with current technologies. EXPPO uses XML to store pages, XPath to query pages and XHTML when rendering pages. First, we describe how XML is used to structure and to provide the meta-language for pages. Our use of XML also makes it easy to design a system of transitioning between page units. Next we explain how we use XPath statements to implement the query system so that EXPPO can reference data and support handlers. Functional handlers integrate with page design using support functions, message passing standards and transaction request processing to interact with a *page's* data. We finish by showing two supporting technologies. EXPPO includes the ability to rapidly generate implicit pages to support rapid end-user development. EXPPO also uses, a simplified database interface to manage transactions using the native format of the database and the specific format of EXPPO documents.

5.1 Page Oriented Programming: XML Page Units

The format of an EXPPO page unit conforms to the XML 1.0 specification. Using XML as a format enables EXPPO to use existing XML tools, including XPath, a language for selecting and computing data from XML documents [Chamberlin]. XPath is based on XML hierarchy trees, using a variety of criteria to navigate around XML documents and select particular XML elements.

By using the XML format for its meta-language, EXPPO provides structure that is machine readable via XPath and flexible enough for user defined application develop-

ment. EXPPO also uses the XML format to store page descriptions and to dictate the page flow of the web application being created.

In the EXPPO prototype, authors are required to do a large percentage of XML tag writing manually. The EXPPO prototype responds with rudimentary warnings if invalid XML is submitted to be saved to a page in either its content or data sections. Possible improvements to this manual process are discussed in section 9.

5.2 EXPPO Meta-Language

The EXPPO XML meta-language contains three main elements that describe each page unit. These three components are a functional description of a web page and, when combined with a cohesive set of other pages, comprise a web application. As can be seen in *Figure 2*, the three sections of a page unit - metadata, content and data - are expressed in XML format. A more verbose explanation of their contents follows the diagram.

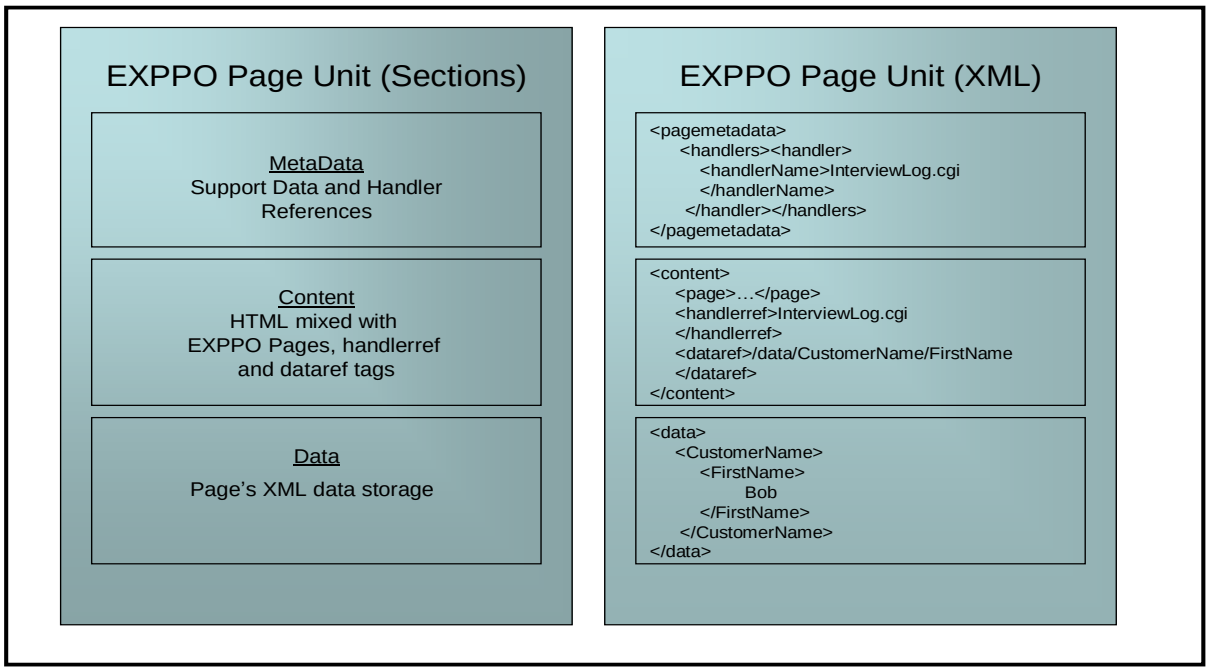


Figure 2 Divisions of a Page Unit and their XML Representation

- Metadata – This section of the document contains information on who created a document, when it was created and last modified, the page's title, and a page type. Handlers used by a page unit will be specified in this section of the XML meta-language document. The metadata section must contain a page identifier tag, which is the unique identifier for a web page document at a given level of the containing document's XML hierarchy.
- Content – This is the interface specification for a web page, which must conform to the XHTML standard to be viable with the rest of the system. References to data items in content will be rendered with the result of the XPath statement that the reference points to. Render will also cause references to handlers in this section to execute the referenced handler and the result to be inserted at the location of the reference. The content section will replace pages embedded inside its structure with page links, using the XML hierarchy to create page flow.

- Data – This is the data that is owned by a page unit and can be referenced by tags in the content section. Examples of elements this section might contain include information on products to be advertised for sale, image references, blog entries, news stories, job openings, candidates for those openings and any other data that should be stored about a given page's tables, pictures, graphs, windows, and text areas. An EXPPO data query will by default reference data in the page's data section.

```
<!DOCTYPE Feed [
  <!ELEMENT Feed (Entry*)[
    <!ELEMENT Title (#CDATA)>
    <!ELEMENT Subtitle (#CDATA)>
    <!ELEMENT Updated (#PCDATA)>
    <!ELEMENT ID (#CDATA)>
    <!ELEMENT Link (#CDATA)>
    <!ELEMENT Rights (#CDATA)>
    <!ELEMENT Entry (Page*)[
      <!ELEMENT Page [
        <!ELEMENT PageMetaData (Contributor*) [
          <!ELEMENT Page_Id (CDATA)>
          <!ELEMENT Author [
            <!ELEMENT Name (#CDATA)>
            <!ELEMENT Uri (#PCDATA)>
            <!ELEMENT Email (#PCDATA)>
          ]>
          <!ELEMENT Contributor [
            <!ELEMENT Name (#CDATA)>
            <!ELEMENT Email (#PCDATA)>
          ]>
          <!ELEMENT PageType (#PCDATA)>
          <!ELEMENT Display [
            <!ELEMENT Content_Type (#PCDATA)>
            <!ELEMENT Style_Sheet (#PCDATA)>
          ]>
          <!ELEMENT Handlers (Handler*)[
            <!ELEMENT Handler [
              <!ELEMENT HandlerMethod (#PCDATA)>
              <!ELEMENT HandlerName (#PCDATA)>
              <!ELEMENT HandlerRef (#PCDATA)>
            ]>
          ]>
        <!ELEMENT Content (Page_ref*,Handler_ref*,Data_ref*, HTML*, Page*)[
          <!ELEMENT Page_ref (#PCDATA)>
          <!ELEMENT Handler_ref (#PCDATA)>
          <!ELEMENT Data_ref (#PCDATA)>
          <!ELEMENT HTML (XHTML)>
          <!ELEMENT Page (EXPPOPage)>
        <!ELEMENT Data (XML)>
      ]>
    ]>
  ]>
]>

<!ENTITY XHTML "http://www.w3.org/TR/xhtml1/">
<!ENTITY XML "http://www.w3.org/2002/xmlspec/dtd/2.10/xmlspec.dtd">
<!ENTITY EXPPOPage "a <!ELEMENT Page> cannot be included into the DTD without causing recur-
sion">
```

Figure 3 EXPPO Meta-Language DTD

Figure 3 is a formalized Document Type Definition or DTD for the EXPPO language [Lu]. Each of the elements specified conform to standard DTD 1.0 syntax, with the exception of embedded EXPPO pages, an element which would cause recursion and violates the DTD rule set. Where no other specification is given, an element contains exactly one of each of its children. Where an element specification contains a note about one or more of its children with a star symbol, it is an indication that the element may contain any number of the children element preceding the star. PCData is a pseudo acronym for parsed character data and CData indicates character data.

The DTD begins with a feed element because EXPPO uses the ATOM format standard to store its documents. The root page entries of web applications are stored as entries inside the ATOM feed. While the ATOM format serves no purpose in the current implementation of EXPPO, it was chosen to ease integration with ATOM publishing and storage applications in the future.

5.3 Page Transitions

EXPPO uses the hierarchical nature of XML as a method of dictating the page flow of a website application. This means that the specification for a page can be placed inside another page. When the site in question is generated, only a link to the embedded page is visible on the parent page. *Figure 4* demonstrates how the page flow of EXPPO web applications can be designed:

In order to keep EXPPO page flow structured and yet unrestricted, standard HTML link tags can be used in the EXPPO specification to allow user-developers to keep page flow as embedded XML *page* units, as standard link tags, or both. In *Figure 4* the XML for *page bob* is contained within the XML for *page alice*, thereby instructing EXPPO to create a link on page alice for a user to navigate to page bob. When a user navigates to page bob, the page id in the url would change from "alice" to "alice/bob." EXPPO uses this path based page id to location page bob inside page alice.

```
<page alice>
   <page bob>
   </page bob>
   <a href://[ip-address]/index.cgi?page_id="george"/>
</page alice>
…
<page george>
</page george>
```

Figure 4 XML Page Flow example

As shown here, page alice contains a reference link tag to page george. Page george's XML is contained outside page alice, yet the reference link is an instruction to create a link to page george on page alice. By using the structure of XML for page flow instructions, EXPPO has the ability to describe an entire website in a single XML document or to divide that document into multiple separate components.

5.4 EXPPO Query System

Because data in the EXPPO system adheres to the XML standard, a query takes the form of an XPath statement. The design of a query to the system, a URL concatenated with an XPath statement, was designed to be conducive to automated construction.

As shown in *Figure 5*, concatenating a URL with a query ("?XPath=") and an XPath location will return a data selection from an EXPPO file. While accessible through a browser window, the query system is mainly intended for automated systems to access data. Using these query URLs is how data is retrieved by other EXPPO systems, handler logic and third party-applications.

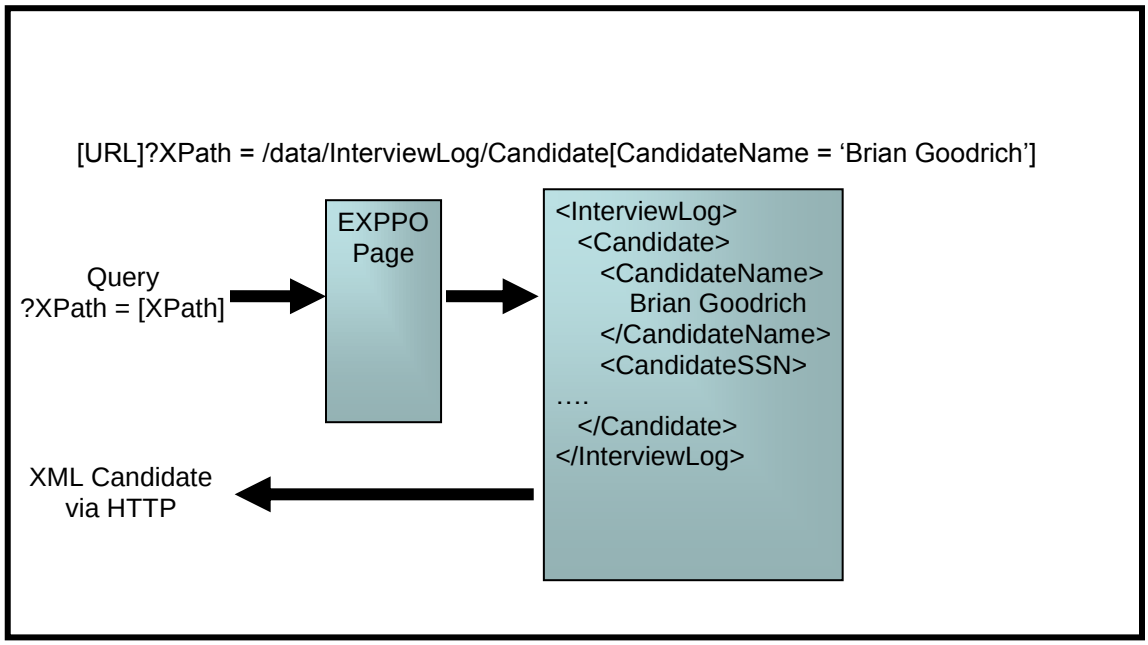


Figure 5 Functional Diagram of the EXPPO Page Query System

A full query to the EXPPO system is the simple concatenation of a URL, a parameter indicating a query and an XPath statement. For example, a user may want to query a website with data on car parts, having a page flow design as shown in *Figure 6* for data items with attribute *parent-part* that have a value equal to two. The desired query might appear in an XPath statement like this:

http://[server.domain.tld]/index.cgi?page_id = parts_home/parts_directory/parts_list;

XPath=data/part_list[part/parent-part=2]

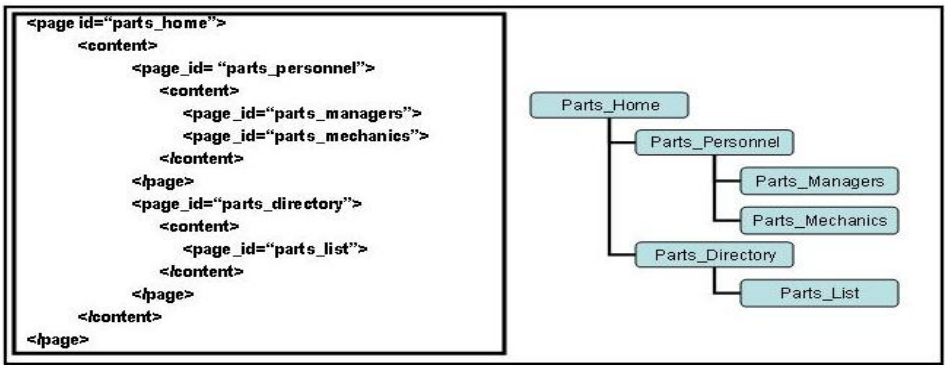


Figure 6 Example of Embedded Page Flow for Sample Query URL.

The return value would appear in a browser window as shown in *Figure 7.*

```
<part_list>
   <part number="1070">
      <description>Head gasket for 1995 Geo Prizm</description>
      <parent-part>2</parent-part>
   </part>
   <part number="1071">
      <description>Head gasket for 1993 Toyota Corolla</description>
      <parent-part>2</parent-part>
   </part>
</part_list>
```

Figure 7 Data that is the Result Set of the Query in the Part_list Example

5.5 Functional Handlers

Handlers are functional modules that dictate page logic and behavior. The current prototype supports Perl scripts, although extensions for additional languages should be fairly easy to implement. One or more handler tags may be contained inside the metadata section of a page tag, and references to web application code are referenced by a file path or URL inside these handler tags. EXPPO is able to call handlers from the file system of the hosting machine using a file path, or remotely from other EXPPO application systems

by using a URL. Multiple handler tags can be placed together in chains to benefit from functionality from multiple handler modules.

Placing the handler XML tags inside the metadata section of page documents accomplishes two things. First, EXPPO gains functionality for web applications while still holding the original premise that the foundational unit of the EXPPO system is a page unit. Secondly, adding handler specifications to the metadata section gives handlers the power to be easily "searchable" in the system. A handler can be retrieved using a simple extension to EXPPO and an XPath statement that points to the handler specification.

Using this system, programmers can provide the public with rich functionality for web applications because handlers are modular enough to be exchanged and tailored to an end-user's needs. Increases in the amount of software that is used in EXPPO databases causes growth that benefits not only EXPPO users, but also other developers that query these EXPPO systems for handlers. Additionally, because handlers are modular, less code-savvy users can also build useful web content by combining different handlers in use by other EXPPO systems.

Figure 8 shows a typical communication between a handler and the EXPPO system that requested its execution. At render time, a handler reference is evaluated and the URL of the handler is requested. Often, a handler will need to use some data from the requesting page and so it sends a request using the EXPPO Query System discussed in Section 5.4. EXPPO provides a small library of built in functions to assist developers with this process. The handler parses the response, produces its output and returns the

result to the requesting system. The result is then inserted into the requesting part of the EXPPO page, as discussed in Section 4.3.

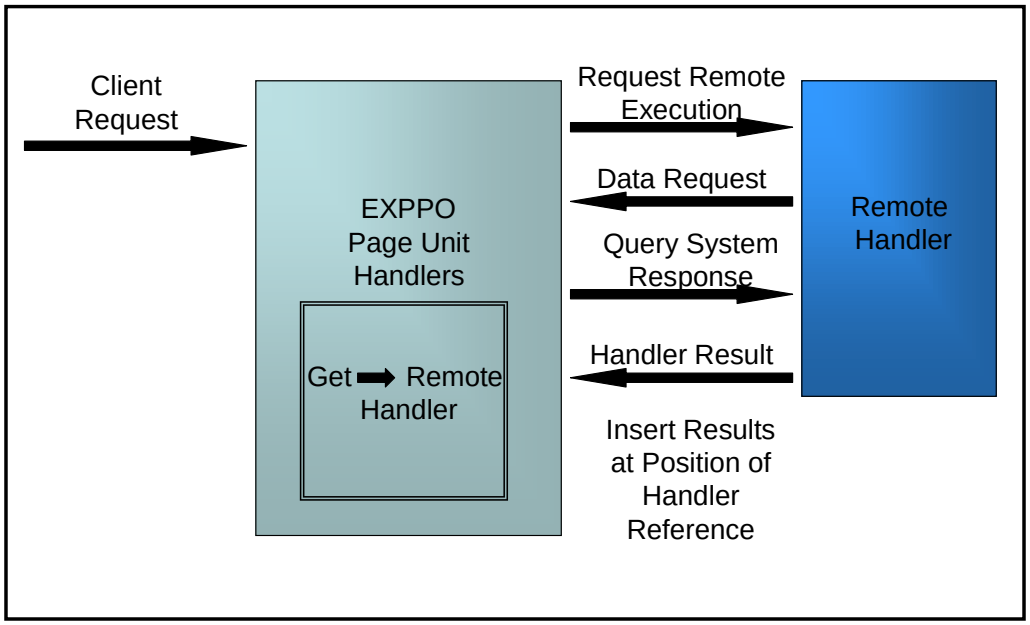


Figure 8 Diagram of EXPPO Remote Handler Execution

To facilitate interaction with remote handlers, EXPPO implements a handler query system. This system involves a simple meta-instruction for a handler query to a list of user-specified URIs that point to other EXPPO systems. When an EXPPO URL is requested with a key-value pair of "exppoHandlerCall=exppoHandlerDisc", the EXPPO system will respond by returning a list of the handlers stored at that EXPPO system. Future implementation could easily include some kind of description of the services that those handlers provide, whether contained in a WSDL document or another format.

5.5.1 Handler Message Passing and Handler Data Transactions

The EXPPO system uses an XPath based message passing system to handle data transactions with a dynamic set of functional handlers. This message passing system is

based on key-value pairs with the key being the name of an XHTML input variable and the value being equivalent to the input variable's value. When creating an EXPPO handler, a developer selects how data should be used by naming the input variables in the handler with an XPath statement appended with a meta-tag instruction. The meta-tags in question fall at the end of the XPath statement, and indicate either a write ("ExppoWrite") or delete ("ExppoDelete") statement. A lack of a meta-tag in the input variables name indicates a read transaction. For example, if a user wanted to have his data about an interview session stored in the company EXPPO site, a portion of the handler's XHTML output could be:

```
<input type="textbox"
name="/data/InterviewLog/InterviewEntry/InterviewCandidate/CandidateName/ExppoWrite"> </input>
```

This particular input variable indicates that the value that the user inputs into this textbox should be written at the location of the XPath statement preceding the "ExppoWrite" instruction. This XHTML, when submitted via HTTP POST to the EXPPO system with a user entered value of "Bob Seger", will create an entry in the data section of the page in question that looks like the XML in *Figure 9.* EXPPO processes the XPath statement and creates parts of the hierarchy to make the XPath valid if those parts do not already exist and then writes the value of the key-value pair (in this case "Bob Seger"), to the appropriate location.

When designing an ExppoWrite statement in a handler there are several conditions to consider. First, if the data section of the page unit in *Figure 10* had been completely blank, the result would have been the same. When evaluating an XPath

statement, EXPPO follows the existing XML element hierarchy in a step-by-step procedure until the end of the path is reached or the next element required in the path does not exist. Then EXPPO creates any elements required to fulfill the path and writes the value at the indicated location.

```
<page>
  <data>
    <InterviewLog>
      <InterviewEntry>
        <InterviewCandidate>
          <CandidateName>
            Bob Seger
          </CandidateName>
        </InterviewCandidate>
      </InterviewEntry>
    <InterviewLog>
  </data>
</page>
```

Figure 9 Example Interview Log Data Entry

If an XML element already exists, and another statement uses the same XPath, EXPPO will create another entry at the most specific level available. To use the previous example in *Figure 9*, if the same XHTML input is used a second time with a value of "Harshwardhan Nagaonkar," a second CandidateName element will be created under the same InterviewCandidate tag with that value. While this may seem like an erroneous behavior in the example, it exemplifies why careful selection of XPath names is important as they are responsible for the automated construction of the XML in the EXPPO *page* units.

The example in *Figure 10* shows how to use an EXPPO handler input to write a social security number to a specific part of an XML document using values that already exist in the document. With the exception of the "/ExppoWrite" part of the statement, the

input name in *Figure 10* follows standard XPath syntax for the location of a CandidateSSN inside an InterviewCandidate element where there also exists a CandidateName element with a value equal to ‘Bill Seger’. Since ‘Bob Seger’s’ InterviewCandidate element does not meet this criteria it would not be written there, and EXPPO would create a single CandidateSSN element with the value ‘999-99-9999’ after the CandidateName element with the value of ‘Bill Seger’. The automated construction of these statements is facilitated by the EXPPO query system, which can serve handlers with *page* data, and EXPPO provides built-in functions to facilitate this.

```
<input type="textbox" name="page/data/InterviewLog/InterviewEntry/InterviewCandidate[CandidateName
= ‘Bill Seger’]/CandidateSSN/ExppoWrite">999-99-9999</input>

<page>
  <data>
    <InterviewLog>
      <InterviewEntry>
        <InterviewCandidate>
          <CandidateName>
            Bob Seger
          </CandidateName>
          <CandidateSSN>
            555-55-5555
          </CandidateSSN>
        </InterviewCandidate>
        <InterviewCandidate>
          <CandidateName>
            Bill Seger
          </CandidateName>
                          <Candi-  ◄──  dateSSN>999-99-9999</CandidateSSN>
        </InterviewCandidate>                    (will be inserted here!)
      </InterviewEntry>
    <InterviewLog>
  </data>
</page>
```

Figure 10 Example of doing a value specific location write in the EXPPO system

Delete statements follow much of the same behavior but are even simpler to evaluate as they are not concerned with an accompanying value to be written. Exppo-Delete simply evaluates the XPath statement associated with the control to which it is attached and then removes any of the resulting data from the page unit. Consider the following example, on the same page:

```
<input type="submit”
name="page/data/InterviewLog/InterviewEntry/InterviewCandidate[CandidateName = ‘Bill
Seger’]/ExppoDelete">
```

Using this statement, any InterviewCandidate element containing a CandidateName with the value of ‘Bill Seger’ will be deleted. The amount of data

encompassed by the delete can be increased by changing the position of the '[' character in the XPath statement to include more parts of the path. As with deletes in most database systems, ExppoDeletes are very powerful statements and should be used very carefully.

5.5.2 XML Message Cluster Creation

To manage more complex transactions with remote systems, the implementation of EXPPO includes a system for combining multiple messages from the same handler that have similar XPath key values. The problem arose from an inability to place multiple incoming XPath key-value pairs in the intended parts of an XML document. *Figure 11* shows a set of example messages to be inserted into the XML data below it.

```
 <input type="textbox"
name="page/data/InterviewLog/InterviewEntry/InterviewCandidate/CandidateName/ExppoWrite">
Harshwardhan Nagaonkar </input>

 <input type="textbox"
name="page/data/InterviewLog/InterviewEntry/InterviewCandidate/CandidateSSN/ExppoWrite">999-99-
9999</input>

<input type="textbox"
name="page/data/InterviewLog/InterviewEntry/InterviewCandidate/CandidateAge/ExppoWrite">
23</input>

<page>
   <data>
      <InterviewLog>
         <InterviewEntry>
            <InterviewCandidate>
               <CandidateName>
                  Bob Seger
               </CandidateName>
               <CandidateSSN>
                  555-55-5555
               </CandidateSSN>
            </InterviewCandidate>
            <InterviewCandidate>
               <CandidateName>
                  Bill Seger
               </CandidateName>
            </InterviewCandidate>
         </InterviewEntry>
      <InterviewLog>
   </data>
</page>
```

Figure 11 Example Interview Log Entry with Multiple Entries

These instructions have been submitted from a single handler that is referenced by the current page unit. The outcome of this handler is to add a new interview candidate with a name, age and social security number to an interview entry. The next example, *Figure 12*, shows how the messages will be executed without the message clustering system.

```
<page>
  <data>
    <InterviewLog>
      <InterviewEntry>
        <InterviewCandidate>
          <CandidateName>
            Bob Seger
          </CandidateName>
          <CandidateSSN>
            555-55-5555
          </CandidateSSN>
          <CandidateName>
            Harshwardhan Nagaonkar
          </CandidateName>
          <CandidateSSN>
            999-99-9999
          </CandidateSSN>
          <CandidateAge>
            23
          </CandidateAge>
        </InterviewCandidate>
        <InterviewCandidate>
          <CandidateName>
            Bill Seger
          </CandidateName>
          <CandidateName>
            Harshwardhan Nagaonkar
          </CandidateName>
          <CandidateSSN>
            999-99-9999
          </CandidateSSN>
          <CandidateAge>
            23
          </CandidateAge>
        </InterviewCandidate>
      </InterviewEntry>
    <InterviewLog>
  </data>
</page>
```

Figure 12 Bad Behavior Result on Figure 11

Figure 12 shows the disastrous behavior associated with submitting multiple simultaneous statements through the message passing interface alone. No new InterviewCandidate entry has been created, but the new information elements for Harshwardhan have been added to the InterviewCandidate entries for both Bob Seger and Bill Seger. The cause for this is apparent; the XPath statement indicated those locations.

Although the user and the handler developer were unlikely to wish the result in *Figure 12*, the three XPath statements, when handled individually, do in fact point to the locations where the data was written.

In order for the intended data to be delivered appropriately, these three separate messages need to act together. The EXPPO system implements a system of clustering XPath messages from handlers and evaluating the clusters. First, the messages from a handler are analyzed for similarities within a major section of a page document, a major section meaning: metadata, content or data. Wherever the paths are identical up to a point, then those messages are selected to be a part of an XML cluster. The XPaths of the different statements are truncated to the point to where they stop being identical.

Using the example in *Figure 11*, three inputs will be combined into a new single key-value set. The new combined XPath statement will be:

page/data/InterviewLog/InterviewEntry/InterviewCandidate/ExppoWrite

The new XPath statement's value will include the values of all three statements. The values are clustered in XML, constructed by using the truncated parts of the original XPaths and so will contain a CandidateName element, a CandidateSSN element and a CandidateAge element, each with their accompanying values. This new statement is now evaluated as a write to the system and produces the desired result, as shown in *Figure 13.*

```
<page>
  <data>
    <InterviewLog>
      <InterviewEntry>
        <InterviewCandidate>
          <CandidateName>
            Bob Seger
          </CandidateName>
          <CandidateSSN>
            555-55-5555
          </CandidateSSN>
        </InterviewCandidate>
        <InterviewCandidate>
          <CandidateName>
            Bill Seger
          </CandidateName>
        </InterviewCandidate>
        <InterviewCandidate>
          <CandidateName>
            Harshwardhan Nagaonkar
          </CandidateName>
          <CandidateSSN>
            999-99-9999
          </CandidateSSN>
            <CandidateAge>
            23
          </CandidateAge>
        </InterviewCandidate>
      </InterviewEntry>
    <InterviewLog>
  </data>
</page>
```

Figure 13 Good Behavior Result on Figure 11

5.6 Implicit Page Generation

An additional feature EXPPO offers users is the ability to create pages implicitly by using the current page as a template [Takao]. This means that a page can exist with content and data while not actually ever having entered those elements in a page in the database. By simply adding some rudimentary tags and instructions to a root page's content field, a user can specify a series of implicitly created pages that are generated on the fly [Volker]. The ability to automatically generate pages makes the EXPPO system more convenient for web development.

For instance, say a user implements a review web site for digital cameras. The explicit page would list separate divisions of these cameras by model, manufacturer, price, or some other piece of information. When a user clicks an EXPPO hyperlink, the program will process the URL of that link. If the URL stored is for an implied page and a reference to the data items selected, EXPPO will generate a page with data stored about the selections.

For example, on an EXPPO shopping page for Olympus cameras, a user will click on a link and EXPPO will display a page with the data that corresponds to the cameras that fit the description of the user-selected link. If the user clicks on one of the cameras, another page will be generated implicitly showing more verbose data specific to the camera selected stored on the explicit parent page. This example is using two levels of implicitly generated pages from the root page of the digital camera shopping application.

5.7 ATOM API

The ATOM API is actually a separate software application, written in conjunction with EXPPO. The API serves as a wrapper system for the DBXML 2.0 database, simplifying the database API and providing the capacity to wrap transactions in an ATOM entry format [Obermeier]. The ATOM format is a publishing protocol similar to RSS 2.0. EXPPO uses ATOM entries to contain the specification documents that hold page units, making page unit documents more accessible to third party publishing applications.

Though simplifying to the interface to DBXML, the ATOM API does not possess significant portions of the functionality supported by DBXML. Although the API does not currently support a method to actually modify entries or append new entries to an already existing document, the API still makes this possible through loading a document, appending the new entry to a document and overwriting the data back to the same file, achieving the same result but suffering a performance reduction.

6. Results

To verify these claims, we built a Unit Test Application in the EXPPO system that tests each of the abilities mentioned in this thesis. Each of these tests completes successfully. Implied pages are generated, page flow is dictated by embedding page units inside other pages, the query system delivers data to handlers and browsers and handler logic can update the database by using XPath entries for input field names. *Figure 14* shows the main page of the Unit Test Application with the following features:

1. Embedded pages are rendered as page links.
2. Output from local and remote handlers use locally stored data.
3. There is a link to an implicitly generated page (the “genPage” link).
4. The page makes use of a CSS specification stored in the data section to show a data reference in use.

The Unit Test application proved valuable for development of the EXPPO system. Validation of working components indicated when bugs were introduced to the system.

Challenges not considered in the initial design were revealed during the test application's implementation.

Figure 14 Image of the Main Window of the Unit Test Application

To show how EXPPO can be used to create a real application, we designed an Interview Log Application. This system can take a job description, record it and begin taking records for applicants for that position by creating page units for each applicant for the position. Interviewers can select a particular applicant's page unit and make additional comments about a particular applicant.

Using EXPPO, it is very easy to create an Interview Log application. At the EXPPO start page, enter the identifier of the job opening, and click 'Create'. The Edit Window will appear and a user should enter a page id, select the interview_log.cgi for a

handler and click ‘Save’. The Interview Log application is now ready for use. *Figure 15* shows a page flow diagram a user might experience while using the Interview Log Application.

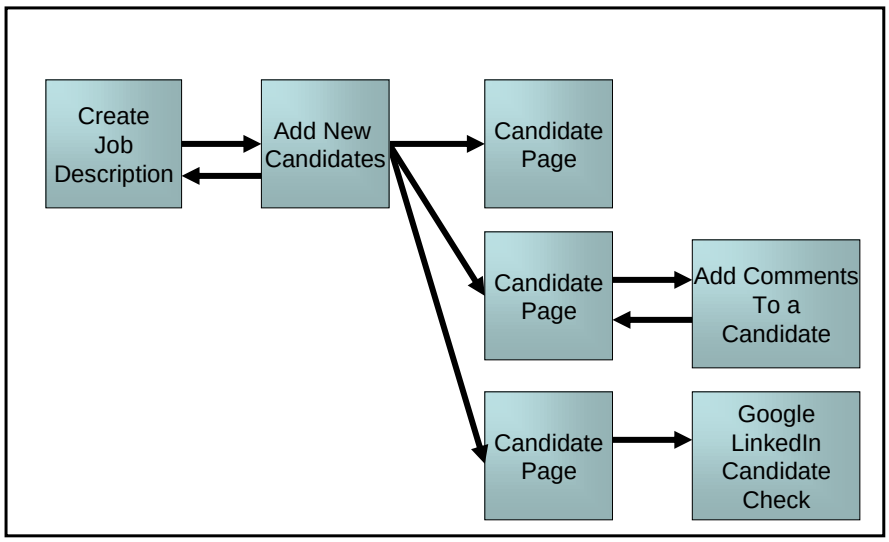


Figure 15 Page flow diagram for Interview Log Application

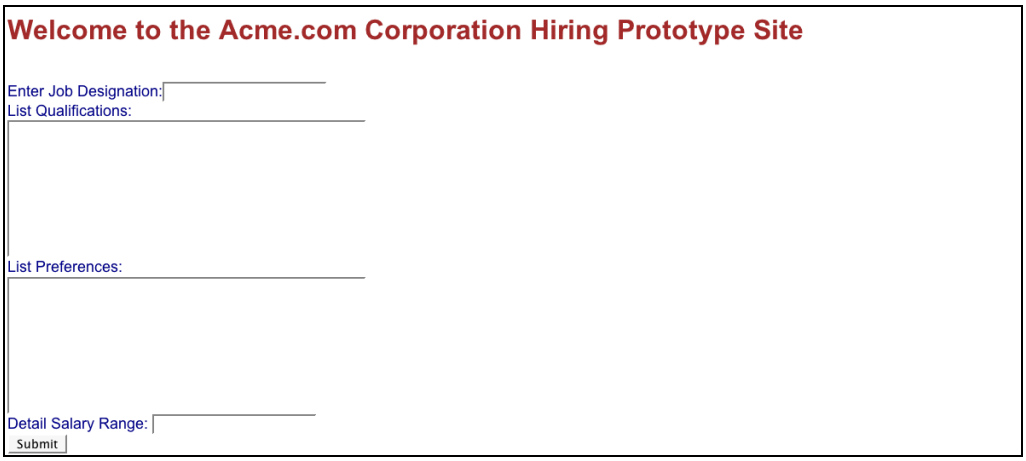


Figure 16 Image of the Create Job Description Page

Upon entering a brand new instance of the Interview Log application, the user is prompted with the screen shown in *Figure 16*, the Create Job Description Page. After entering the values into the page and clicking enter, the page will change state to present

the main page of the application, an example of which is shown in *Figure 17.* From this page, interviewers can change information about the job posting, add information about new candidates for the job position, and navigate to an existing candidate's information.

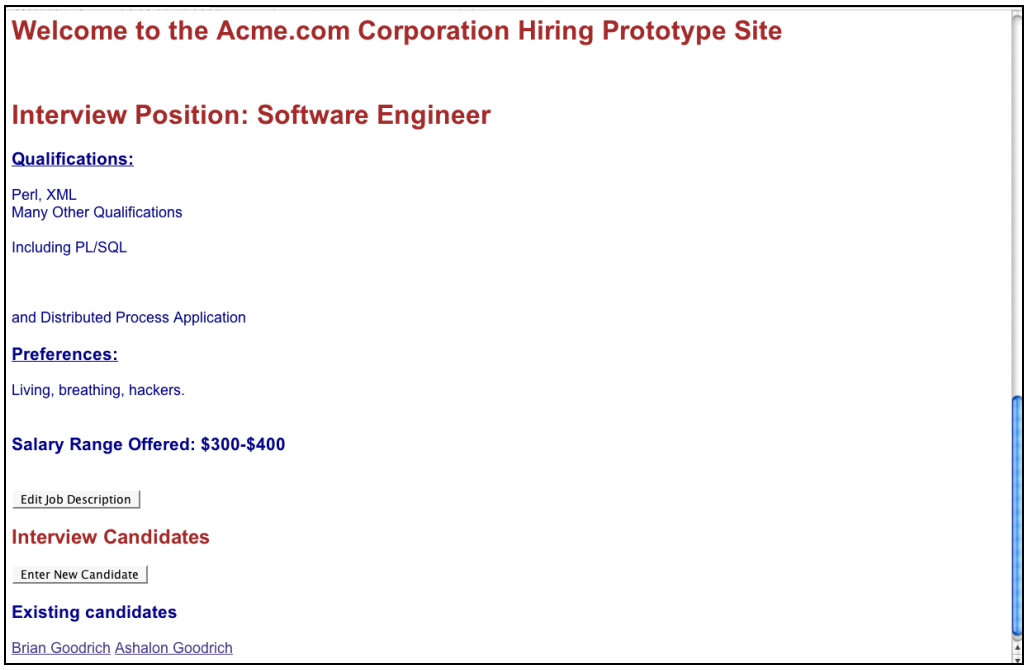


Figure 17 Image of the Interview Application Main Page for adding Candidates for a Software Engineer Job Position

Information about a particular candidate is stored in a page unit generated by the Interview Log application. An example of the Rendered version of one of these page units is show in *Figure 18.* This page holds information including a resume, a name, a social security number and a set of comments made by interviewers. The candidate page uses another handler that provides the display of this information and an interface for interviewers to add new comments about the page candidate. Not shown in *Figure 18* are a pair of generated content items at the bottom of the page that are links to search for a candidates information using the Google Search engine and the LinkedIn Network search feature.

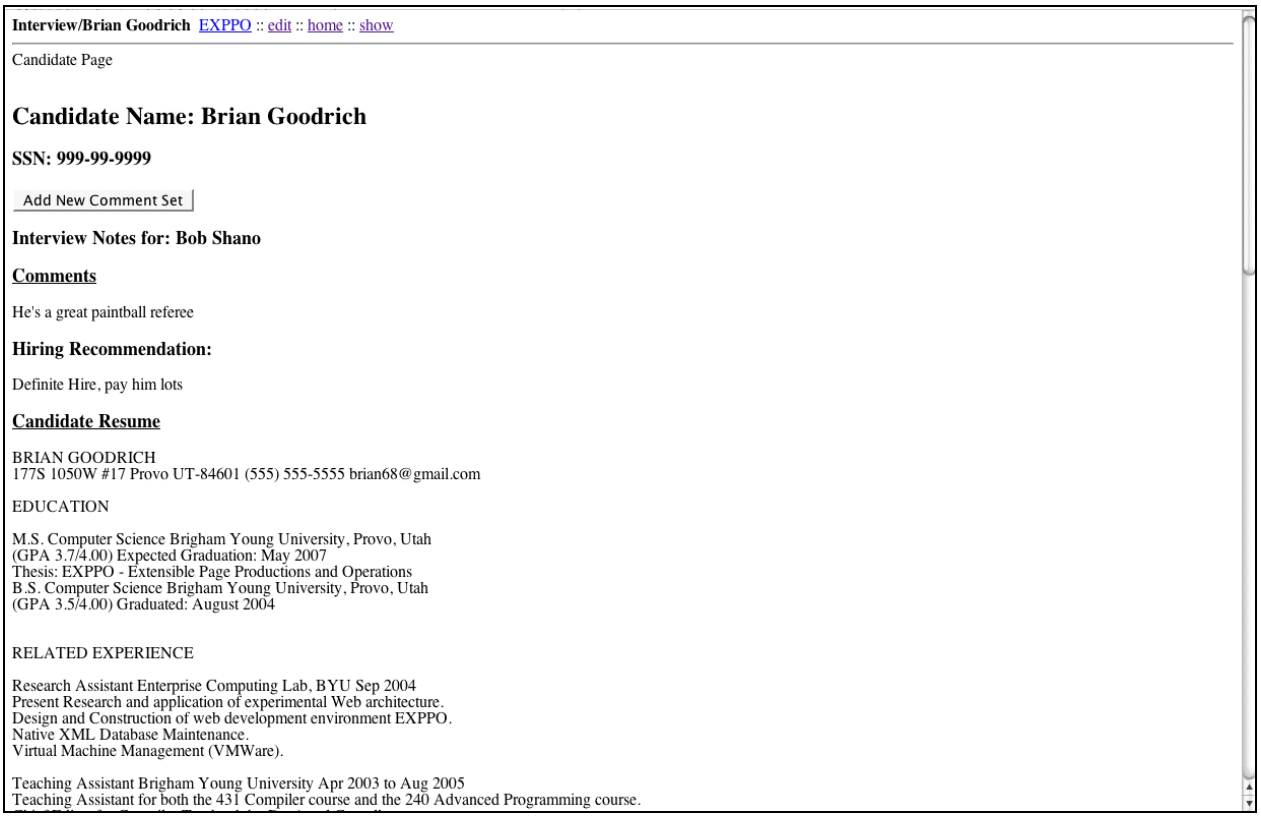


Figure 18 Log of Candidate Information on a Candidate Page

7. Future Work

Given that EXPPO is a prototype, it is not surprising that the work begun in this thesis has a large potential for refinement and expansion. An additional project for EXPPO would be the creation of a much more graphical interface for managing page units. Since EXPPO introduces strict XML structure to web application creation, sets and subsets of content and data can be arbitrarily grouped and then moved or copied to other EXPPO document locations. A GUI could be built on this principle, making it much easier to organize data and to add and rearrange parts of a web application. Such an implementation, while not functionally required by the EXPPO system, would open EXPPO application development to a much wider range of users.

Akin to the browser-based GUI improvement would be the creation of a fat-client or desktop application built specifically for EXPPO application development. While remote handler interaction could be problematic in a desktop environment, creation and organization of content, page flow and page data would still be very reasonable to construct off-line. When a document would be ready for submission into the EXPPO system, a user could publish from a desktop or fat-client environment to a set of EXPPO library repositories, to make the new EXPPO application available for use. Additionally, the fat client system could be used to create a more powerful interface for generating XPath queries by freeing the interface from the restrictions of a browser window and the inherent delay in network communications.

A centralized repository of code and addresses of EXPPO systems, (similar to the

APT or YUM RPM repositories) would greatly enhance the ability to discover handler code. A description of handler functionality (similar to a WSDL document) could communicate to end users the purpose of a particular handler, especially if those descriptions included a set of keyword tags to help users search for the type of code they desired. Search results would be ordered by an index based on the similarities to search terms and the popularity of requests for a given handler. As a particular handler becomes more and more popular, its position in the search index would be moved to a more prioritized position. This would continue up to a threshold point where the handler logic itself would be uploaded to the repository to ease the bandwidth usage of a particular EXPPO system. If usage continued to increase past yet another threshold, the repository itself could upload the actual handler code to requesting EXPPO systems, distributing code as a form of load balancing. Updates to the code base at the repository and then to systems that have downloaded the code could be handled by an extension to the subversion system already supporting the EXPPO code base. As EXPPO handlers cannot maintain state without the information from the EXPPO system, this would allow individual EXPPO systems to maintain a handler set with the same functionality as the remote handler while also having better performance throughout the entire system. This same repository system could theoretically also be used to communicate what data a given EXPPO system has available for open data exchange and the URI-XPath combination required to retrieve that data.

While handler sharing is an impressive paradigm to code exchange and speeding web application development, security has not been addressed in this system. EXPPO requires a method of first establishing a set of trusted network nodes with which

data can be exchanged. If handler discovery is to be automated with or without a repository, the method of establishing trust must also be automated. Second, a security negotiation method between individual nodes in the system and a possible repository based upon the automated trust negotiation would need to establish a secure communication line. While not functionally required by the EXPPO prototype, communication of any data but the most insensitive would require a security system meeting these two requirements.

While the current EXPPO system is concerned with only read, write and delete as meta-tags in its message passing system, a richer vocabulary of EXPPO meta-tags would allow a richer functionality set to offer to handler developers. These could include instructions to specify the position of an XML write in a given document, a re-ordering of existing data, a tag for modifying a set of XML data without requiring a delete and insert, tags indicating changes to other EXPPO files and a various other database transaction instructions. Additionally, another meta-tag type could indicate input type checking for HTTP POSTs. These tags would need to indicate what type of data should be submitted from a given input element and what exception handling instructions should be associated with a violation of the indicated constraints. Optionally, type checking could be implemented as a separate set of EXPPO tags that would generate JavaScript that would limit the keystrokes to a given input based on the field validation rules and could emit an error message and lock submission upon detecting bad input into a given field.

8. Conclusions

We have used several example applications to demonstrate the validity of the EXPPO system. Allowing web pages to exist in a Page Oriented Architecture produces semi-structured web content that is still flexible enough to create functionally rich-applications. We have shown that the EXPPO system can express a website with the XML meta-language by building a website with the EXPPO development environment. We have used the EXPPO system to create web sites that incorporate important design principles and features:

- using XML hierarchy to dictate page flow
- using code that has its logic described in a handler tag
- logic that will generate implicitly generated pages.

The fact that EXPPO requires users to manually enter XML and XHTML means that the prototype does not yet provide an improved experience for end-users. However, because the implementation of EXPPO maintains the syntax of XPath and XHTML throughout the system, further work for the interface could easily use the prototype as a backbone for a much richer experience for end-users.

References

[Bricklin] Dan Bricklin. “About Wikicalc 0.1” 9 November 2005 <http://www.bricklin.com/log/aboutwikicalc01.htm>

[Ceri] Stefano Ceri, Piero Fraternali and Aldo Bongio, “Web Modeling Language (WebML): a modeling language for designing Web sites.” Computer Networks, Volume 33, Issues 1-6, June 2000, Pages 137-157.

[Chamberlin] Don Chamberlin, Daniela Florescu, Jonathan Robie, Jerome Siḿeon, and Mugur Stefanescu (eds.). “XQuery: A query language for XML.” W3C Working Draft <http://www.w3.org/TR/xquery, February 2001.>

[Cunningham] Ward Cunningham. “Correspondence on the Etymology of Wiki.” <http://c2.com/cgi/wiki?WikiWikiWeb >

[Dabble] “Dabble DB” <http://www.dabbledb.com/about/>

[Di Iorio] Angelo Di Iorio, Fabio Vitali. “From the Writeable Web to Global Editability.” HT 2005, September 6-9, 2005 Salzburg, Austria, 2005.

[Eikvil] Line Eikvil. “Information extraction from the world wide web - a survey.” In Technical Report 945. Norwegian Computing Center, Oslo, Norway, 1999.

[Gardner] Philippe Gardner and Sergio Maffeis. “Dynamic Modeling Web Data.” In DBPL '03 LNCS Springer, 2003.

[Gonzalez-Reinhart] Jennifer Gonzalez-Reinhart. “Wiki and the Wiki Way: Beyond a Knowledge Management Solution.” Information Systems Research Center, pp. 1-22/February 2005

[JotSpot] JotSpot Inc., “JotSpot Beta: the application wiki.” <http://www.jotspot.com/>

[Lu] Shiyong Lu, Yezhou Sun, Mustafa Atay, Farshad Fotouhi. “On Consistency of XML DTDs.” In Data and Knowledge Engineering 2005, vol. 25, Issue 2, pages 231- 248

[McArthur] Gregory McArthur, John Mylopoulos, and Siu Kee Keith Ng. “An extensible tool for source code-representation using xml.” pages 199–208. Proceedings of the Ninth Working Conference on Reverse Engineering, IEEE Computer Society, Richmond, Virginia, USA, October 2002.

[Nelson] Theodor Nelson. “Literary Machines.” Sausalito, CA, Mindful Press, 1987.

[Obermeier] Sebastian Obermeier. “Embedded XML-Databases.” Seminar Workout, November 2004 <gauge.upb.de/pgmanet/seminar/workout/embeddeddatabases.pdf >

[Quint] Vincent Quint, Irene Vatton. “Towards Active Web Clients.” DocEng 2005. <http://wam.inrialpes.fr/publications/2005/DocEng05-Quint.html>

[Schwall] Johannes Schwall. “The Wiki Phenomenon.” August 2003

[WikiMedia] “Home-From the Wikimedia Foundation.” version 17th March 2006 <http://wikimediafoundation.org/wiki/Home>