

## **Application Integration for Free Open Source Medical Software: A Case Study**

Kip Canfield, PhD

Department of Information Systems

University of Maryland, UMBC

Correspondence to:

Kip Canfield

IFSM UMBC ITE 425

1000 Hilltop Circle

Baltimore, MD 21250

[canfield@umbc.edu](mailto:canfield@umbc.edu)

410-455-2649

## **Abstract**

**Introduction:** This paper presents a specific case study of integration between Free and Open Source applications.

**Methods:** The case study does not use an existing system but one with simulated data. This approach allows emphasis on the underlying issues. This case study integrates an open source Content Management System with an open source native XML database using the Simple Object Access Protocol (SOAP).

**Results:** An architectural pattern for integrating open source medical applications is presented with a working demonstration system. This architectural pattern is evaluated against a list of goals for health care information systems taken from the health care informatics literature and is found to be consistent with those goals .

**Discussion:** Free and Open Source Software (FOSS) offers a rich development environment for health care information systems and application integration offers way to build effective systems in a modular fashion without duplication of effort.

**Keywords:** Free Open Source Software, Electronic Medical Records, XML Databases, Application Integration

## Introduction

Free Open Source Software (FOSS) has had significant growth recently using Linux and applications built on that operating system. Linux servers grew in both revenue and shipments by about 50 percent from last year [1].

Medical software is in a good position to take advantage of these developments since the penetration of software for clinical use is low compared to other business sectors [2] and investment in health care information technology is now greater than that for all other sectors [3]. Furthermore, the architecture for modern Internet applications is supported by a vigorous sector of information technology that includes web servers, application servers, databases, and protocol standards. This Extensible Markup Language (XML) and web browser-based architecture is central to most current software development for Internet applications. Software architecture [4] is defined as “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.” We define a software architecture here that integrates FOSS elements as components to create a clinical information system.

Some goals frequently mentioned in the literature for FOSS approaches to health care information systems are:

- Avoidance of vendor lock-in [5]
- Ability to integrate and interoperate with other systems [6, 7]
- Lowered costs for development, acquisition, and maintenance

The last goal for lowered costs is uncontroversial, but not yet supported in the literature in large part due to the recent appearance of FOSS. This paper presents an argument that application integration offers a reasonable and logical possibility of lowered costs, but can offer no empirical evidence. This case study will use these goals as requirements.

Many existing FOSS clinical systems are delivered over the web. A review of the prominent medical FOSS projects listed at <http://www.linuxmednews.com/LMNProjects> shows that they typically offer application services such as authentication, community services such as forums or bulletin boards, and other non-patient record

functions. This duplication of programming and development effort is not necessary with a strategy of application integration. Each project also defines its own patient record data model and API functions which prevent effective accomplishment of all the above listed system goals by reducing interoperability and increasing maintenance costs as described below.

This case study project takes a current FOSS Content Management System (CMS) as the base adds the Native XML database (NXD) using the Simple Object Access Protocol (SOAP) to integrate the two applications. CMSs are a robust sector of the FOSS world as seen in the list at <http://www.la-grange.net/cms>. They rival or exceed commercial offerings. NXDs [8] are more recent developments but with a standard Application Programming Interfaces (API) like XML:DB (<http://www.xmldb.org/>), they offer a very attractive way to manage documents that does not have the ‘impedance mismatch’ problems of XML-enabled relational database approaches that can lead to complex development where the XML documents have to be taken apart and put back together for every retrieval and update.

## **Methods**

This case study does not use existing systems as components but creates them using simulated data. This approach avoids unnecessary complexity and allows focus on the underlying goals listed above. The following steps were used for this project. Each of these steps is detailed in the following text.

- A FOSS CMS was chosen.
- A FOSS NXD was chosen.
- A XML standard schema for health care content was chosen.
- A protocol for application integration was chosen.
- 100 patient records were generated in XML format according to the standard schema.
- The input and output interfaces were created using style sheets.

### *System Components*

This section presents a description of each component of the architecture. The CMS used for the study is OpenACS

(<http://www.opencs.org>). It is a FOSS system based on the Aolserver web server (<http://aolserver.org>) with its built-in TCL (Tool Command Language - <http://www.tcl.tk>) support and the PostgreSQL (<http://www.postgresql.org/>) as the relational database back end. CMSs support many of the application functions that we require for a clinical record system infrastructure. They are also efficient, effective for high-load use, and have large support infrastructures that include free Internet support by their organizations and fee-based support by companies that specialize in FOSS. These community-oriented CMSs offer many services that we can use in building systems, but one of the most important is authentication and authorization. Most all of these FOSS CMSs offer sophisticated group-based authorization that is easily adapted to Health Insurance Portability and Accountability Act (HIPAA) requirements with the supplied configuration interfaces.

The TCL API in the Aolserver web server allows scripting any additions that are required to the CMS while the postgresql database contains the data model for every OpenACS application. For example, a forum or bulletin board system requires that all posts be stored in the database. From the OpenACS web site: “OpenACS is an advanced toolkit for developing community web applications. The Open Architecture Community System provides: A large set of applications, which can be used to deploy web sites that are strong on collaboration. Some of the powerful applications are Workflow, CMS, Messaging, Bug/Issue tracker, e-commerce, blogger, chat and forums. OpenACS has dozens more applications [and a] sophisticated application development toolkit, which provides a extensive set of APIs and services to enable quick development of new applications.” Most CMSs similarly allow applications to be added in a modular fashion.

The NXD Exist (<http://exist-db.org>) is a FOSS project authored by Wolfgang Meier [9] and was used for storing patient record documents in this study. The web site describes Exist as: “eXist is an Open Source native XML database featuring efficient, index-based XPath query processing, extensions for keyword search, XUpdate support and tight integration with existing XML development tools. The database is lightweight, completely written in Java and may be easily deployed in a number of ways, running either as a stand-alone server process, inside a servlet-engine or directly embedded into an application. Depending on how the database engine has been deployed, eXist offers several interfaces to application developers, including HTTP, XML-RPC, SOAP and WebDAV. Java developers should have a look at the XML:DB API, which provides a common interface to access XML database

services.”

The HL7 Clinical Document Architecture (CDA) [10] is an XML standard for clinical documents that can be used not only for information transfer, but also for operational storage of patient data. This case study used the CDA schema for generating the patient record documents, but any other XML standard for clinical documents could be substituted.

FOSS NXD databases that offer search and update through a standard API are interchangeable in the sense that one can transfer a document collection from one database to another without changing any code other than the referring URL. The standard schema supplied by the CDA also gives independence from the project specific data models that most patient record systems have. One only has to move their documents to another compliant NXD to change. NXDs are just beginning to be considered for health care systems [11, 12]. There is also proprietary activity in this area such as Microsoft's InfoPath 2003 Health Level Seven Clinical Document Architecture Demo (<http://www.microsoft.com/technet/itsolutions/mso/cda/default.asp>) and commercial NXD vendors such as X-Hive (<http://x-hive.com/>).

In order to populate the NXD for this study, the Toxgene program [13] (<http://www.cs.toronto.edu/tox/toxgene/documents.html>) was used to generate CDA documents according to its standard XMLSchema. Toxgene allows one to annotate an XML schema with special instructions for generating documents and save this as a template file. The Toxgene program uses this template to generate any number of XML documents based on the template patterns. These synthetic CDA documents validate to the CDA schema, but are populated with random words from the general English and medical dictionaries- i.e. they do not make any sense but are schema compliant. Very simple Extensible Stylesheet Language Transform (XSLT) style sheets were used to create the display and update interfaces with a server-side transformation.

SOAP [14] was used to connect the CMS to the NXD. This web services protocol [15] is open, standard and has support in most languages including TCL. It is an XML-based protocol for making Remote Procedure Calls (RPC) between systems over the Hypertext Transport Protocol (HTTP).

### *System Description*

This section details the development of the integrated demonstration system. A working demo of this system using the simulated patient documents is available at <http://celtic.umbc.edu:8009/>. The interfaces for both the CMS and the NXD are not intended to be realistic, but offer a technical example of the functionality required for application integration. The undeveloped entry screen to OpenACS is shown below in figure 1. It is a minimal presentation where typically would be the entry page design for the clinical system.

OpenACS uses a server-side markup scheme where each presentation page is actually a pair of pages. In this case, there is both an 'index.adp' and an 'index.tcl.' The Aolserver Dynamic Page (.adp) is the page with the markup (analogous to .php or .jsp) while the .tcl page defines the variables and procedures for that markup. OpenACS offers many default applications such as the site search and forum examples shown in figure 1. OACS2NXD is the name of SOAP-based NXD connector application. Figure 2 shows the page for patient record access from the remote NXD.

The search interface is minimal since this is not a specific issue for software architecture. One requests a file name in the top form, the software creates the SOAP request, sends it to the separate NXD process, and inserts the XSLT of the returned XML source in the second gray box. One can use the links in figures 2 and 3 to jump to the XML source and the XSLT source. To show the CMS-side authorization, note that the header bar shows the user as 'Not logged in.' This demo assumes (unrealistically, but functionally equivalent to the real situation) that unauthenticated people have read authorization for patient records and authenticated ones have read/write authorization. After login, figure 3 shows the simple update field of the encounter provider's last name. The XUpdate standard from the XML:DB API was used here. Note that the header bar now shows the user's logged in status as 'Welcome, *Username*.'

Under this architecture, the interface code is given by XSLT and is completely transferable to any other compliant system with completely different component parts. Furthermore, these XSLT style sheets are easily maintained in repositories as a library for developers. The transform is done with the xslt.tcl file and uses the FOSS TCL library

tDOM (<http://www.tdom.org/>). The XUupdate document is dynamically created with TCL code in the xup.tcl file. Both of these other TCL files are under 10 lines of code. The main procedural code used here is given in the index.tcl file shown in figure 4.

The SOAP connects were made with the FOSS project TclSOAP (<http://tclsoap.sourceforge.net/>). This occurs in figure 4, line 14 with ‘source soap.tcl.’ Figure 5 shows these simple SOAP interface calls from soap.tcl. There are more complex calls used for searching and available at the Web Services Description Language (WSDL) URL for the NXD (in this case <http://celtic.umbc.edu:8080/exist/services/Query?WSDL>). The index.tcl file dynamically defines the variables in the index.adp presentation file (shown in figure 6) marked with “@variable@.”

## Results

This section presents an analysis of the case study architecture and compares the results to the list of goals for health care FOSS given in the introduction. This analysis refers to the technical description of the demonstration system.

### *Vendor Lock-in*

When all the components are chosen, some amount of procedural code needs to be written to connect everything. A key requirement to make this architecture attractive is that this component must have only a small amount of non-transferable code of this sort. Figures 5 and 6 show a remarkably small amount of procedural code for such a rich ‘combined’ application. This is a very simplified implementation, but what is needed for a realistic implementation is clear. The code in index.tcl is what would have to be recreated in case of changing the CMS component of this architecture. The XML infrastructure remains the same and the needed libraries for XSLT, Document Object Model (DOM), and SOAP are available as open source for most all languages. Changing the NXD component is far simpler. Any XML:DB compliant database can be used and loaded with the XML patient record documents. The change cost in terms of development effort appears low. Since systems that store patient data in relational databases have differing data models, the change cost is higher.

### *System Integration and Interoperability*

This architecture requires that all service providers expose their functionality as web services under a Services Oriented Architecture (SOA). In this study, SOAP was used as the web services protocol and RPCs were easily accomplished. Since a standard XML schema was used for the patient record documents, any XML-aware application can use them. The open system approach allows one to add functionality in a modular and organic fashion through development or integration. There is no need to require that development only take place for a specific application on a specific platform. The data that any component develops can be used by other applications in the open environment.

#### *Development and Maintenance Cost*

This case study makes an argument that development and maintenance costs are lowered due to:

- Standard schema offered with CDA or other health care standard
- Declarative character of the XML development infrastructure
- Large body of FOSS available such as CMS

A standard schema adopted by the health care community would be a great boon to application development and deployment. A standard schema for clinical documents not only makes record transfer and integration transparent, database administration in an NXD is significantly simplified over other types of databases due to the data model residing in the XML infrastructure rather than the database software as is the case with relational databases, for example. XML development has a declarative character that simplifies and generalizes development. For example, XSLT used for the user interfaces has this character.

There is a large body of FOSS available that if used in a component fashion can reduce development efforts and make use of 'best of kind' components. A notable example is the rich body of FOSS work in CMS that offers much of the functionality such as fine-grained authorization that is needed in health care systems. When such systems can be loosely coupled using the web services infrastructure of the SOA, rich functionality can be attained in an organic way.

## Discussion

This case study has made the case that application integration of FOSS components offers an architecture that has greater interoperability and lowered development and change costs. The simple demonstration system allows close inspection of the interfaces to evaluate these issues. This architecture consists of a web browser client, a CMS used as an application server and back end database(s). The architecture is a specific example of a standard multi-tier design that uses web services to create a modular SOA. Presentation logic, application logic, and database model are separated and each of these modules communicate through standard interfaces. This argument from modularity seems reasonable, but requires further study for confirmation. One should be able to substitute software products of the same class in each tier with the goal of using the best class of product for the intended system. The use of CMS as the application server tier is most problematic for this argument. Any application logic for the integrated system resides there and depends on the particular programming language associated with the CMS. For example, Java Server Pages (.jsp) could be used instead of the .adp pages used here. Servlets could be used instead of the .tcl files and Enterprise Java Beans (EJB) could be used to implement modular application functionality which has no direct analogue in this case study but could be a library of .tcl files. This additional application object (EJB-type) tier adds considerable complexity and functionality to the proposed architecture and is currently being developed in projects like that from the HL7 Reference Information Model (RIM) [10] and openEHR Archetypes [16]. It is an open question as to whether the additional functionality of these object models will hinder modularity and application integration in a trade-off fashion.

## References

- [1] Lyman J. Linux Servers Lead Market in Worldwide Growth, November 26, 2003 (available <http://www.technewsworld.com/perl/story/32259.html>).
- [2] Goldsmith J, Blumenthal D, Rishel W. Federal Health Information Policy: A Case Of Arrested Development. *Health Affairs*, 2003 22(4): 44-55.
- [3] ITAA Report, December 2003, e-Data E-Health: Healthcare Information Technology Spending Is Growing Rapidly (available <http://www.ita.org/isec/pubs/e200312-10.pdf>).
- [4] Bass L, Clements P, Kazman R. *Software Architecture in Practice, 2/E*, Addison Wesley Professional, 2003.
- [5] Kantor G, Wilson W, Midgley A. Open-source Software and the Primary Care EMR. *J. Am. Med. Inform. Assoc.* 2003 10(6):616.
- [6] Stead W, Miller R, Musen M, Hersh W. Integration and Beyond Linking Information from Disparate Sources and into Workflow, *J. Am. Med. Inform. Assoc.* 2000 7:135-145
- [7] Ehrcollaborative. Final Report: Public Response to HL7 EHR Ballot 1. 2003 (available <http://www.ehrcollaborative.org>)
- [8] Chaudri A, Rashid A, Zicari R, editors. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison Wesley Professional 2003.
- [9] Meier W. eXist: An Open Source Native XML Database. In: Chaudri A, Jeckle M, Rahm E, Unland R, editors. *Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops, Erfurt, Germany, October 2002*. Springer LNCS Series, 2593.
- [10] Dolin RH, Alschuler L, Beebe C, Biron PV, Boyer SL, Essin D, Kimber E, Lincoln T, Mattison JE. The HL7 Clinical Document Architecture. *J. Am. Med. Inform. Assoc.* 2001 8:552-569.
- [11] Paterson G, Shepherd M, Wang X, Watters C, Zitner D. Using the XML-based Clinical Document Architecture for Exchange of Structured Discharge Summaries. *Proceedings of the 35th Hawaii International Conference on System Sciences - 2002*
- [12] Xiaou Z, Keng PH. XML-Based Virtual Patient Records System for Healthcare Enterprises. *4th International Conference on Enterprise Information Systems. Ciudad-Real, Spain, April 3-6 2002*.

- [13] Barbosa D, Mendelzon A, Keenleyside J, Lyons K. ToXgene: a template-based data generator for XML. In Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002). Madison, Wisconsin - June 6-7, 2002.
- [14] Brandes W, Sagan P, The Simple Object Access Protocol and an Online Health Information Infrastructure. The Informatics Review Apr 1 2001 (available <http://www.informatics-review.com>)
- [15] Martin J, Arsanjani A, Tarr P, Hailpern B. Web Services: Promises and Compromises Queue 1(1) March 2003.
- [16] openEHR Foundation. The GEHR Software Architecture September 23, 1999 (available <http://www.gehr.org/>).

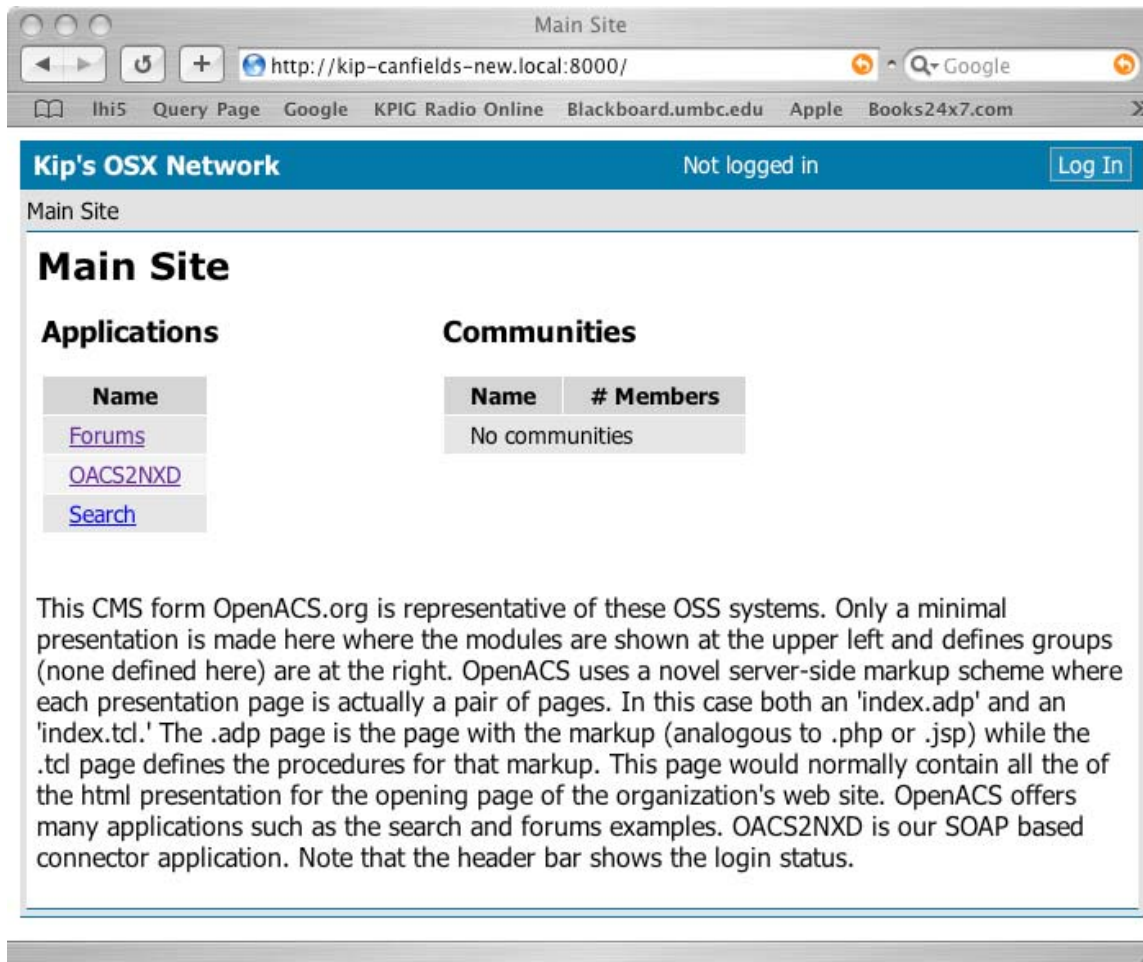


Figure 1. OpenACS Entry Screen

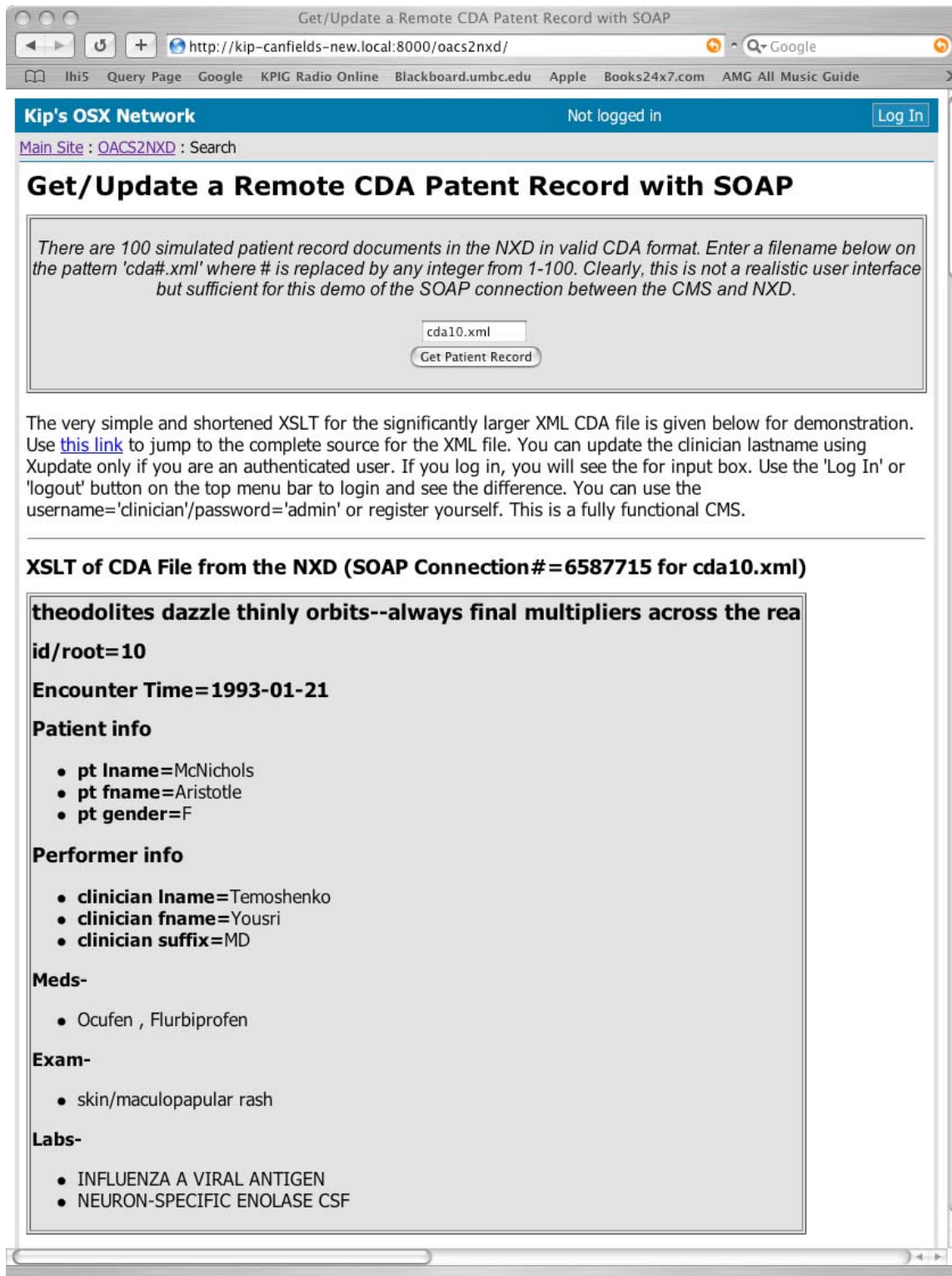


Figure 2. The Read-only Patient Record Interface.

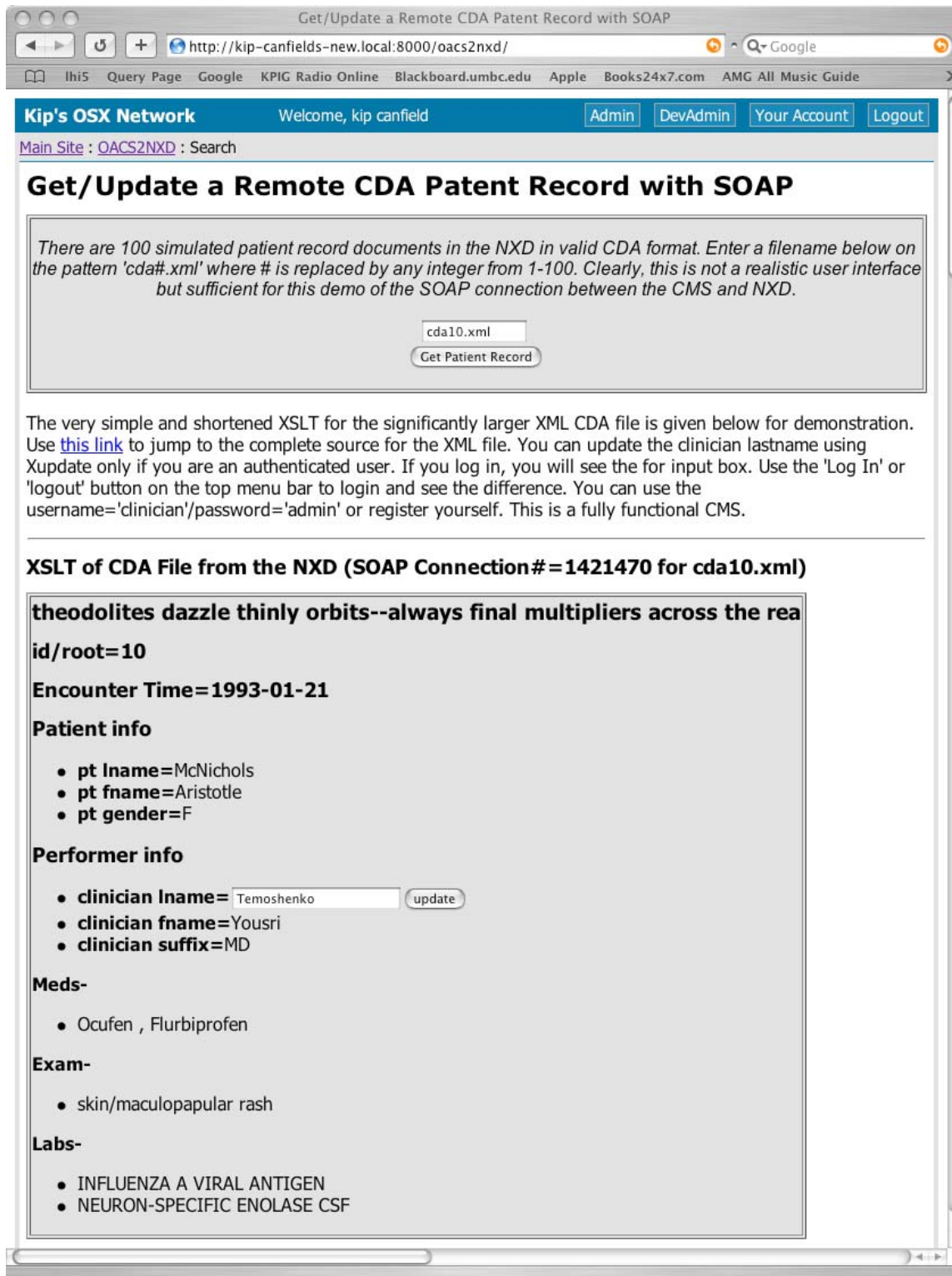


Figure 3. The Patient Record Interface with Update.

```

# packages/search/www/index.tcl
ad_page_contract {
  @author canfield@umbc.edu
  @creation-date 2003-12-03
} {
  q1:optional
  p:optional
  v:optional
} -properties {
  x:onevalue
}
if { ![info exists q1] } { set q1 "cda10.xml"}
set q $q1
package require SOAP
set user_id [ad_conn user_id]
if { $user_id != 0 } {
  set user_name [person::name -person_id $user_id]
  set con [connect admin xxx]
  set xsl [getResource $con /db/cda/cda100/cda2.xsl 1 1]
} else {
  set con [connect guest guest]
  set xsl [getResource $con /db/cda/cda100/cda.xsl 1 1]
}
source /Users/canfield/xup.tcl
source /Users/canfield/xslt2.tcl
if { [info exists v] } {
  set p "/ClinicalDocument/encounterPerformer/assignedEntity/assignedPerson/name/family"
  set xup [xupdate $p $v]
  xupdateResource $con /db/cda/cda100/$q $xup
}
set x [xslt [set xml [getResource $con /db/cda/cda100/$q 1 1]] $xsl]
disconnect $con
ad_return_template

```

Figure 4. The index.tcl File for Definition of Variables

```

SOAP::create connect \
    -proxy "http://localhost:8080/exist/services/Query" \
    -action urn:exist \
    -params {user string password string} \
    -encoding http://schemas.xmlsoap.org/soap/encoding/ \
    -uri "http://localhost:8080/exist/services/Query?WSDL"
SOAP::create getResource \
    -proxy "http://localhost:8080/exist/services/Query" \
    -action urn:exist \
    -params {sessionId string name string indent boolean xinclude boolean} \
    -encoding http://schemas.xmlsoap.org/soap/encoding/ \
    -uri "http://localhost:8080/exist/services/Query?WSDL"
SOAP::create disconnect \
    -proxy "http://localhost:8080/exist/services/Query" \
    -action urn:exist \
    -params {user string} \
    -encoding http://schemas.xmlsoap.org/soap/encoding/ \
    -uri "http://localhost:8080/exist/services/Query?WSDL"
SOAP::create xupdateResource \
    -proxy "http://localhost:8080/exist/services/Admin" \
    -action urn:exist \
    -params {sessionId string documentName string xupdate string} \
    -encoding http://schemas.xmlsoap.org/soap/encoding/ \
    -uri "http://localhost:8080/exist/services/Admin?WSDL"

```

Figure 5. The SOAP Interface in soap.tcl.

```

<master>
<property name="title">Get/Update a Remote CDA Patent Record with SOAP</property>
<property name="context">Search</property>
<center>
<table border=1 bgcolor="#dddddd"><tr><td><center>
<p><i>There are 100 simulated ...</i></p>
<form method=GET action=index.tcl>
  <small>
  <input id="cda" type=text name=q1 value=@q@ size=12 maxlength=12>
  <br>
  <input type=submit value="Get Patient Record">
  </small>
</form>
</center></td></tr></table>
</center>
<p>The very simple ...</p>
<hr />
<h3>XSLT of CDA File from the NXD (SOAP Connection#=@con@ for @q1@)</h3>
<table border=1 bgcolor="#dddddd"><tr><td>
@x;noquote@
</td></tr></table>
<hr />
<a name="xml" />
<h3>XML Source</h3>
<pre>
@xml@
</pre>

```

Figure 6. The index.adp File for Presentation

## **Legends**

Figure 1. OpenACS Entry Screen

Figure 2. The Read-only Patient Record Interface.

Figure 3. The Patient Record Interface with Update.

Figure 4. The index.tcl File for Definition of Variables

Figure 5. The SOAP Interface in soap.tcl.

Figure 6. The index.adp File for Presentation