LYNX: AN OPEN ARCHITECTURE FOR CATALYZING THE
DEPLOYMENT OF INTERACTIVE DIGITAL GOVERNMENT
WORKFLOW-BASED SYSTEMS

By

Iván P. Vélez-Ramírez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

University of Puerto Rico
Mayagüez Campus
2006

Approved by:

Manuel Rodríguez, Ph.D.
Member, Graduate Committee

Jaime Seguel, Ph.D.
Member, Graduate Committee

Bienvenido Vélez, Ph.D.
President, Graduate Committee

Name LastName, Ph.D.
Representative of Graduate Studies

Isidoro Couvertier, Ph.D.
Chairperson of the Department
ABSTRACT

LYNX: AN OPEN ARCHITECTURE FOR CATALYZING
THE DEPLOYMENT OF INTERACTIVE DIGITAL
GOVERNMENT WORKFLOW-BASED SYSTEMS

By

Iván P. Vélez-Ramírez

Write abstract here ...
RESUMEN

LYNX: UNA ARQUITECTURA ABIERTA PARA CATALIZAR EL DESPLIEGUE DE SISTEMAS DE FLUJOS DE TRABAJO DE GOBIERNO DIGITAL

Por

Iván P. Vélez-Ramírez

Escriba el resumen aquí...
To ...
ACKNOWLEDGMENTS

Write acknowledgments here ...
TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

1 Introduction 1
  1.1 Overview ......................................................... 1
  1.2 Problem Statement ............................................... 4
  1.3 Proposed Solution .............................................. 5
  1.4 Objectives of this Thesis ..................................... 6
  1.5 Summary and Contributions .................................... 7
  1.6 Thesis Structure ................................................ 8

2 Related Work 9
  2.1 Workflows and Web Services ................................... 9
    2.1.1 Web Services ............................................. 9
    2.1.2 Business Processes Using Web Services .................. 10
  2.2 Web Forms ................................................................ 13
    2.2.1 Traditional Approaches .................................. 13
    2.2.2 XForms ..................................................... 14
  2.3 Native XML Databases ........................................... 19

3 Lynx Architecture 21
  3.1 BPEL Execution Engine .......................................... 21
  3.2 Outgoing Email Web Service ................................... 22
  3.3 Other Partner Web Services ..................................... 23
  3.4 Incoming Email Gateway ......................................... 23
  3.5 Email Client ..................................................... 23
  3.6 XForms Player .................................................... 24
  3.7 Document Repository ............................................ 24
  3.8 User, Role and Process Repository ............................. 24
  3.9 Workflow Querying Subsystem .................................. 25
# 4 Lynx Implementation

4.1 Introduction .................................................. 26
4.2 Document Definition ........................................... 27
4.3 Process Description ............................................. 27
4.4 Workflow Querying Subsystem .................................. 35
4.5 XML Storage Subsystem ......................................... 36
4.6 Customizing Lynx ................................................. 39

# 5 Experiments and Results

5.1 Introduction .................................................. 41
5.2 Deployment and Configuration for the Experiments .............. 42
  5.2.1 Server Configuration ....................................... 42
  5.2.2 Client Configuration ....................................... 43
5.3 Experiments .................................................. 44
  5.3.1 The Experiments ........................................... 44
  5.3.2 Amount of Code Required .................................. 44
    5.3.2.1 Methodology ........................................ 44
  5.3.3 Load Evaluation ........................................... 44
    5.3.3.1 Methodology ........................................ 45
  5.3.4 XML Database Performance Evaluation ....................... 45
    5.3.4.1 Methodology ........................................ 45
5.4 Qualitative Analysis .......................................... 45
  5.4.1 Steps needed to deploy an application using Struts ........ 45
  5.4.2 Steps needed to deploy an application using JSF .......... 46
  5.4.3 Steps needed to deploy an application using Lynx .......... 47
5.5 Results .................................................. 48

# 6 Conclusions and Future Work

6.1 Research Conclusion .......................................... 50
6.2 Future Work .................................................. 50
6.3 Final Word .................................................. 51

References .................................................. 51

BIBLIOGRAPHY .................................................. 52

APPENDICES .................................................. 54

A Appendix Title1 .................................................. 55
LIST OF TABLES
# LIST OF FIGURES

1.1 Sample Registry of Deeds Scenario .......................... 3
2.1 XForms Model connecting to many possible user interfaces .... 15
2.2 XForms Summary .................................................. 17
3.1 System Architecture ............................................. 22
4.1 XML Schema for Outgoing Email Web Service ................. 28
4.2 Invoking an Analyst ............................................. 30
4.3 Email Client showing threaded Lynx messages ................. 31
4.4 XForms for a Mortgage Document ................................ 32
4.5 XForms Email Submission ....................................... 33
4.6 WSDL Message Definitions ..................................... 33
4.7 Document Status Page .......................................... 36
4.8 Log XML Schema ............................................... 37
4.9 Document Status Log View ..................................... 38
4.10 Document Status Attachments view ............................. 38
4.11 Workflow Process Detail Graph ................................. 39
4.12 Document Type Configuration ................................. 40
CHAPTER 1

Introduction

1.1 Overview

Digital Government information systems provide support for government officials to satisfy their citizen’s needs. Such systems could go from simple information displays, to systems that automatically canalize user requests throughout a government agency, maintain relevant document data and improve the overall quality of the services provided to the citizen. Thus, workflow systems are sometimes adopted to automate government processes by specifying how tasks are structured, who performs them, what their relative order is, how they are synchronized, and how information flows to support the tasks.

For example, a workflow system would be helpful for government agencies such as the Puerto Rico Registry of Deeds. The Registry of Deeds holds a public archive that contains all the documents about property transactions, wills, judicial orders and other legal documents. The Registry of Deeds has a large backlog of documents pending for processing due to the meticulous verifications and validations that are currently conducted by several specialized human analysts. Many of these processing steps can be automated and canalized
by a workflow system. However, the required technical expertise is often not easily available. A workflow system based on Lynx may significantly increase document throughput while simultaneously reducing cost and increasing reliability and quality of service at a cost more affordable by small and regional governments.

A simplified version of a process of registering the purchase of a property in Puerto Rico is shown in Figure 1.1. The notary public, a lawyer in Puerto Rico’s system, writes a property title deed document and submits it along with a summary called a presentation minute. This document is received by a receptionist that makes some initial validations. Then, the document is passed to a suitable analyst to verify that the information contained in the document is correct and matches with the documents already in the Registry pertaining to previous transactions on the same property. The document can be verified by several analysts if necessary. The analysts generate a second summary of the document, called an inscription minute, and can add annotations if there are minor errors that require clarifications or explanations. If there is a serious incongruence (registral fault) the document is returned to the notary public. After validation, the document reaches the Registrar himself. This person certifies that all the information is correct and officially adds the inscription minute to the Registry.

A process like this may take several days or weeks through which the workflow system must keep track of every step and provide a query interface to find out the status of every running process at all times. This scenario is typical of many other governmental processes that require a long-running sequence of validations and approvals involving multiple people and systems making decisions and providing information.

Experience in regional and municipal Digital Government environments has consistently demonstrated the need for familiar and broadly accessible interaction mechanisms and user interfaces for the effective adoption of information technologies (IT) given the
relatively low level of exposure of personnel to IT in these settings. Email is familiar for people, provides a simple means of communication for person to person interaction, allows easy interconnectivity between all participants, and most importantly it allows mobility and capability of working from a distant location through the Internet. Email has the potential to free participants from the constraints of space and time allowing senders and recipients to communicate at convenient times and places [19]. Studies have consistently demonstrated the striking number of different uses for email: email can support conversations, operate as a task manager, as a document delivery system, an archive, and contact manager, to name a few [19]. Also, from a technical standpoint, email operates using simple ubiqui-
tous protocols available across links of widely varying qualities, firewalls and other security membranes. For example, participants outside a firewall can easily interact with workflows thanks to email without having to give them an access to the intranet.

Furthermore, workflow applications are expensive to build and maintain. Yet, for either a business or the government, building a workflow is sometimes indispensable. However, the business process expert who generally has little programming skill must resort to a software vendor to customize the business application. This procedure is expensive and time-consuming [28].

1.2 Problem Statement

Workflow systems allow the specification and evolution of complex business processes without requiring complex programming skills. Ordinary business process workflows are oriented towards interacting with human users directly via some interface that runs at their workplace desktop [11]. This approach typically follows a pull-based model, where the user is burdened with periodically logging in and inspecting the system to verify the status of pending workflow transactions [12] requiring their attention. On the other hand, Web service based business processes provide support for aggregating Web services into new higher-level Web services by means of process composition [16]. This approach provides insufficient support for direct or synchronous interaction with persons. Collaboration tools like email or instant messaging exclusively do not provide the necessary support for structured business processes.

Moreover, building a new workflow application user interface, or modifying an existing one, is expensive since the customized programming process for the presentation, validations, and business logic needed is expensive and time-consuming. Specifically, traditional approaches such as the usage of HTML forms show their limitations as they have
limited features, need of scripting to accomplish common tasks, blend presentation and purpose, and integrate poorly with XML. In addition, traditional HTML forms introduce more complexity in programming Web applications. This complexity originates from several main sources [9]. First, dynamic web pages are often dynamically generated, which makes application code harder to understand and makes troubleshooting more difficult. Second, even when dynamic Web pages are a single source code entity, they are often composed of a mix of markup languages, client-side scripting and server-side function calls, which makes them difficult to read. In addition, the skills needed to understand such source code are continuously expanding, which makes maintenance difficult. Third, the high number of software technologies used in some Web applications makes those applications complicated to design and maintain. These technologies can include JavaScript [13], JavaServer Pages with taglibs [14], servlets [19], Struts [1], XSLT [28], DOM [28], SOAP [28], Web Services [28], Enterprise JavaBeans [18], etc., along with related protocols and configuration data. Finally, traditional form technologies do not work for interaction through Email because they need direct communication with a server that provides the data and validations required through HTTP. Trying to simplify and reduce all these complexity and overhead is of particular importance in Digital Government environments where programming skill is severely scarce and difficult to hire.

1.3 Proposed Solution

The proposed solution is Lynx, a new email extension for workflow systems based on Web Services that leverages on current standard and broadly known technologies, such as email, XML and XForms, in order to reduce the amount of code and thus the cost of developing and maintaining the workflow application. Lynx keeps the information in XML throughout the whole process. This is achieved using XForms, message-style Web services, a BPEL engine, and a native XML database. Lynx also overcomes some of the limitations
of traditional workflow systems, collaboration tools, and graphical user interfaces by providing a web service through which a Web services based workflow application can interact with human partners via an email based forms interface without requiring a specialized client. Lynx can extend many Web service based workflow engines with the ability to send transaction requests to human workflow partners using email not only as the transport mechanism, but rather as the interaction application. Using Lynx, users carry out workflow transactions by processing electronic forms transported to/from their email accounts. The email client implements part of the graphical user interface (GUI). Lynx supports the XForms [14] standard and generates each form based on the type of the document being transported and desired view for the interaction with a person. The request for the transaction is generated by a business processes typically specified using a language such as the Business Process Execution Language (BPEL) [8]. In summary, our work should provide a general purpose email messaging architecture to interact with human partners, reduce the amount of custom code required to develop and maintain the application, and use of common email systems in order to allow government employees to interact with automated workflow engine components using familiar interfaces and without requiring re-learning of new applications.

1.4 Objectives of this Thesis

The main objective of our research is the development of a new web services-based architecture for interactive workflow applications that leverages on current standard and broadly known technologies, such as email, XML and XForms, in order to reduce the amount of code and thus the cost of developing and maintaining the application. The new architecture provides an extension to Web service based workflow engines with the ability to send transaction requests to human workflow partners using email not only as the transport mechanism, but rather as the interaction application. This research also intends
to assess the advantages and disadvantages of the proposed architecture in a sample Digital Government scenario involving a Puerto Rico Registry of Deeds workflow process.

The specific objectives of this work are as follows:

• Design a general purpose email messaging architecture to interact with human partners by using the BPEL workflow language.

• Develop a general purpose outgoing email Web service.

• Develop an incoming email gateway to the BPEL engine.

• Develop a system implementing the proposed architecture applied to a Digital Government workflow.

• Define and implement BPEL processes specifying the workflow of a subset of documents most widely used at the Registry of Deeds

• Design and implement XForms user interfaces for each required document.

• Conduct a series of analysis on the system’s performance in a sample Digital Government environment.

1.5 Summary and Contributions

TODO

The design and development of a general purpose email messaging architecture to interact with human partners by using the BPEL workflow language and XForms...

The development of two prototypes for...

An evaluation of....

The publication and presentation of two papers...
We illustrate the usefulness of our approach in a Digital Government scenario...

1.6 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 discusses related work that serves to develop this thesis. Chapter 3 presents an overview of the Lynx architecture and its components. In Chapter 4, we present a detailed description of Lynx implementation and integration with a Web services based workflow. Chapter 5 presents the evaluation and analysis of this thesis. Finally, Chapter 6 presents a summary of our contributions, conclusions and future work.
CHAPTER 2

Related Work

This chapter presents background information, previous work, and relevant publications related to this research. The topics include: Workflows, Web Services, Web forms technologies, and native XML databases.

2.1 Workflows and Web Services

2.1.1 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed using SOAP messages, typically transported using HTTP with an XML serialization in conjunction with other Web-related standards[22]. eXtended Markup Language (XML) is used to provide information about the data in a document to users of varying platforms. The Simple Object Access Protocol (SOAP) [30] is used for cross-platform interapplication communication. The Web Services Description Language (WSDL) [3] is used to describe the online services.
2.1.2 Business Processes Using Web Services

Business Process Execution Language for Web Services (BPEL4WS, or BPEL) is a standard to address how Web services can be combined into higher-level business processes by describing workflows or orchestrations of Web services invocations [16].

The BPEL data model is built on top of WSDL 1.1 messages and XML Schema 1.0 [23] types. All data used within a process model is defined using WSDL message definitions and XML schema types and elements. XPath 1.0 is used for data manipulation. Expressions used for the selection of data, for conditions, and for other purposes are specified as XPath expressions.

Fundamentally, Web services can be modeled as stateless processors that accept messages, process them in some way, and formulate a response to return to the requestor [16]. Yet, in the real world a Web service is not just exposing simple and stateless services. Web services can be reused and combined into more complex services that may provide multiple interactions with a partner for a single business process.

Business processes may run for hours, days, or months, and they may invoke other long-running services. A business process can contain steps that require waiting for external events or human interaction. The aggregation of Web services to new, higher-level Web services by means of process compositions allows for more flexibility. Such compositions can adapt more quickly to changing business needs, compared to hard-coded applications [16].

However, human user interactions are currently not covered by BPEL, which is primarily designed to support automated business processes based on Web services. In practice, however, many business process scenarios require user interaction [27]. Yet, collaboration tools alone, like email or instant messaging, do not provide the necessary support for
structured business processes. Although Web services are designed for machine-to-machine interaction, a service-oriented architecture with Web services could be applicable for interactions involving humans. This could allow the creation of workflows where the service invocations could be replaced by humans providing a requested information or requiring skilled human validation. [27] discusses scenarios and outlines features that need to be supported by BPEL for interactions with people. Our research helps to overcome some of these limitations by enabling email as an alternative mechanism for interaction between a Web service based workflow process and human users.

In addition to the common HTTP transport used for Web services, the Apache Axis Mail Transport [5] allows the transmission of SOAP messages via email through SMTP. Its intention is to send Web service messages between Web services and not between a Web service and a human user via email. Yet, the receiving application must be capable of extracting the payload from the SOAP message which may require running a Web service on the client side. In architecture, Lynx acts as a gateway between a Web services based workflow process and a human partner.

Chakraborty proposes a system called PerCollab which allows convenient communication and collaboration mechanisms (such as SMS, IM and email) to support the activities of a workflow [11]. However, they had to extend IBM’s BPWS4J [13] BPEL language implementation to implement this functionality. Our goal is not to require modifications to the workflow language. We made Lynx generic so that it doesn’t even require a specific BPEL execution engine.

GreenBSN [29] is a middleware architecture for supporting mobile business service networks. Its main goal is to allow vendors to sell their software as a service, using mobile wireless devices. GreenBSN’s service output adaptor module is similar to our proposed outgoing email server component because it delivers communication through a user’s preferred
channel, such as SMS or email. However it is mostly used for asynchronous notifications that only deliver the result of a business process Web service invocation. Lynx allows complete interaction with business processes by allowing human partners to both receive and send information pertinent to the workflow through their emails.

Podgayetskaya [20] proposes an architecture and model for business process support for e-government using a workflow engine and Web services. However, it uses RMI for the workflow enactment service, and a Web-based user interface instead of Web services and email tools.

Ranganathan [2] proposes an architecture that integrates workflow into a pervasive computing environment. This architecture provides a system that generates a customized workflow that describes how various services should interact with one another. However it lacks a mechanism for human interaction through email. Human interaction is allowed via a custom Web interface generated by a user-interface Web service thus requiring significant development effort and specialized user training.

Microsoft offers BizTalk Server [21], an integration server that allows development, deployment, and management of integrated business processes, and XML-based Web services. However, BizTalk can only export its process definitions to BPEL. Also, although BizTalk has different adapters that provide different communication mechanisms, its SMTP adapter consists only of a send adapter [17] that is mainly used for notifications. Thus, it cannot receive messages through SMTP.

Similarly, many other existing applications do not include complete interaction through email. Most only provide notifications through email or instant messaging. Both, Bonita [?] and YAWL [?] feature a notification service through an instant messaging protocol. jBPM [?] specifies the business process using an ad-hoc language called jPDL, and only supports simple notifications through Java Messaging Service.
2.2 Web Forms

On the Web, forms have become very common since nearly all user interaction is done through some type of form. Although it is a very used technology, traditional Web forms are showing its age. In this section several current Web form technologies will be discussed.

2.2.1 Traditional Approaches

According to the HTML specification [?], an HTML form is a section of an HTML document that contains normal content, markup, and special elements called controls such as checkboxes, buttons, text inputs, selection menus, hidden controls and labels on those controls. Users generally "complete" a form by modifying its controls, entering text or selecting menu items, before submitting the form to an agent for processing (e.g., to a Web server, to a mail server, etc.)

Yet, traditional HTML forms predate XML by more than 5 years. Thus, they have several limitations:

- Poor integration with XML
- Limited features make even common tasks dependent on scripting
- Device dependent, running well only on desktop browsers
- Blending of purpose and presentation
- Limited accessibility features

Additionally, some frameworks like the Java Server Faces (JSF) [?] aim to bring rapid user interface development with server-side Java by allowing to create user interfaces with drag-and-drop and providing much of the plumbing that JSP developers have
to implement by hand. JSF provides a set of extensible user interface components and an
event-driven programming model. These components are transformed into different con-
crete, client-side user interfaces through the use of render kits. JSF’s event model is strongly
typed and allows developers to write server-side handlers for events generated on clients.

More recently, technologies such as Ajax [?], or Asynchronous JavaScript + XML,
have been changing the way interaction is done on the Web. Ajax incorporates presenta-
tion using XHTML and CSS, dynamic display and interaction using the Document Object
Model, data interchange and manipulation using XML and XSLT, asynchronous data re-
treival using XMLHttpRequest, and JavaScript binding everything together. However, Ajax
applications involve running complex JavaScript code on the client, thus it is difficult to
make that complex code efficient and bug-free.

2.2.2 XForms

XForms is an XML application that represents the next generation of forms for the
Web [14]. By splitting traditional XHTML forms into three parts, XForms model, instance
data, and user interface, it separates presentation from content, allows reuse and gives
strong typing, reducing the number of round-trips to the server, as well as offering device
independence and a reduced need for scripting. XForms is not a free-standing document
type, but is intended to be integrated into other markup languages, such as XHTML.

XForms have been designed to meet the limitations of HTML forms listed in Section
2.2.1:

• Excellent XML integration (including XML Schema)

• Provide commonly-requested features in a declarative way, including calculation and
  validation

• Device independent, yet still useful on desktop browsers
• Strong separation of purpose from presentation
• Universal accessibility

An XForms form consists of separate sections that describe what the form does, and how the form looks. These are:

• XForms Model - the description of the form.
• Instance Data - initial form data that will be read and written during form interaction.
• XForms User Interface - standard set of visual controls.
• XForms Submission Protocol - defines parameters for serializing and submitting instance data.

Figure 2.1 illustrates how a single device-independent XML form definition, called the XForms Model, has the capability to work with a variety of standard or proprietary user interfaces:

![Diagram](image)

Figure 2.1: XForms Model connecting to many possible user interfaces

The XForms User Interface provides a standard set of visual controls that are
targeted toward replacing today’s XHTML form controls. These form controls are directly usable inside XHTML and other XML documents, like SVG. Other user interface components for XForms may also be developed.

An important concept in XForms is that forms collect data, which is expressed as XML instance data. Among other duties, the XForms Model describes the structure of the instance data. This is important, since like XML, forms represent a structured interchange of data. Workflow, auto-fill, and pre-fill form applications are supported through the use of instance data [14].

Also, there needs to be a channel for instance data to flow to and from the XForms Processor. For this, the XForms Submit Protocol defines how XForms send and receive data. Data is usually sub in XML.

Finally, the form controls need to be connected to the instance data elements. This connection is called binding. The binding also establishes a set of conditions that are applied to the instance data, such as the data type, if it should be read only, required, relevant, or calculated.

Figure 2.2 summarizes the main aspects of XForms:

All these features of XForms have the potential of dramatically reducing the amount of custom GUI code necessary to implement client applications and improving the user experience by giving immediate feedback of what is being filled in. Our work intends to exploit these benefits and usage patterns of XForms to implement the user interfaces required by workflows using Lynx as a general purpose email messaging extension to interact with human partners by using the Business Process Execution Language for Web Services. We hypothesize that by exploiting XForms we will demonstrate the viability of implementing complex distributed interactive applications with significantly less coding. Additionally,
simple modifications to the interaction screens will often not require re-programming of GUI code, with resulting XML that can be fed directly into the workflow and database. This is important in environments where programming skill are severely scarce and difficult to hire, such as in the government.

There are technologies competing against XForms. The most relevant is InfoPath [15], an application bundled with Microsoft Office 2003. InfoPath, like XForms, converts user input into a new or modified XML, which can then be fed into a back-end system. At a high level, both seek to overcome a similar challenge: translating user interaction into XML [15]. However, the two technologies have several essential differences.

The InfoPath application is focused on providing a visual environment of similar quality to the rest of the Microsoft Office suite for creating and filling out forms. In contrast, the XForms specification is designed to encourage implementations not to focus exclusively on visual media, but rather to define only the intent behind various controls. The XForms specification gives implementations the choice of specifying how a particular form control can be implemented. Also, XForms encourages development using a defined declarative XML syntax as mentioned above. On the other hand, InfoPath continues to encourage the
deployment of scripts just like HTML forms [15].

Moreover, the recommended system requirements for InfoPath demand a fairly modern Intel-compatible computer: a Pentium III or greater as well as Microsoft Windows 2000 or greater. Furthermore, the software is bundled only in the Microsoft Office Enterprise edition. On the other hand, the XForms specification was designed to work on the broadest possible range of devices, from small PDAs to big servers. XForms software is being made available in a variety of packages, for many platforms, and both open source and commercial.

Other previous work has also used XForms to simplify Web programming. [9], describe a way to use XForms to simplify Web programming. They define a simplified programming model for form-based Web applications using XForms. However, on the server side they use a subset of J2EE, Java Beans and Struts as enabling technologies. This introduces coding that Lynx tries to avoid by using BPEL, XML standards, and email interactions, using XForms.

The Orbeon Presentation Server [?] is an open source platform that uses XForms to make form-based web applications. Unlike other web application frameworks based on Java objects or scripting languages, OPS is based on XML documents and XForms. This leads to an architecture perfectly suited for the tasks of capturing, processing, and presenting XML data, and does not require writing any Java or scripting code at all to implement a dynamic presentation layer for your web application. OPS is built around a user-friendly Ajax-based XForms engine, which brings standard W3C XForms to mainstream browsers, and the XPL engine, a mature, high-performance XML pipeline engine for processing XML data.
2.3 Native XML Databases

There are basically three approaches to store XML documents in a database. The first is to store XML documents as text in a field within a record. The second is to map the document’s schema to a database schema and transfer data according to that mapping. The third is to use a set of structures that can store any XML document.

Databases that support the second method are called XML-enabled databases. Databases that support the third method are called native XML databases. A native XML database defines a (logical) model for an XML document, as opposed to the data in that document, and stores and retrieves documents according to that model. A native XML database has XML documents as its fundamental unit of (logical) storage [?]. Furthermore, a native XML database system is built and designed for the handling of XML, and it is not just a database system for an arbitrary data model with an XML layer on top [?].

The fundamental unit of storage in a native XML database is a document, equivalent to a row in a relational database, while a collection is a set of related documents and plays a role similar to that of a table in a relational database or a directory in a file system. Thus, native XML databases store complete documents and can store any document, regardless of the schema [?].

In a performance analysis between an XML-enabled and a native XML database, [?] concludes that the native XML database has a better performance for handling large XML documents. This is due to the conversion overhead needed by XML-enabled databases. In contrast, native XML databases access the XML data directly. The only weaknesses found in this analysis were the larger database size of the native XML database due to the space required for the data and index, and the slower updates. [?] also suggests that if the schema of the data is known, a relational database is more efficient, with the reconstruction
of the complete XML documents being its weak point.

Finally, in a very recent article from 2006, Dr. M. Kay [?], explores the role of XML in workflow applications. He states that XML fits very well with workflow applications, because it’s natural to think of them in terms of documents. He states that, in fact, XML is such a good fit that one should design an application as an XML-based workflow where one might have adopted a completely different approach in the past. He describes how specific XML technologies such as XML Schema, XSLT, XQuery, XForms and XML databases fit into the picture. The use of a pipeline processor, such as Cocoon, for binding all these XML technologies together is suggested, although no concrete implementation of such a system is described. Lynx has been in development for more than a year taking these technologies into consideration, but instead of a pipeline processor we have used BPEL as the execution processor, and interact directly with persons through the use of Email messages.
CHAPTER 3

Lynx Architecture

Figure 3.1 shows the different elements that comprise a workflow architecture integrating Lynx. The server side is composed of a BPEL execution engine, an outgoing email Web service and other partner Web services, and an incoming email gateway. The workflow engine exports a Web service interface that can be used to initiate and interact with a business process. Each business process running in the BPEL execution engine may interact with multiple business partners exporting their own WSDL Web service interfaces. Essentially, Lynx consists of two modules: an outgoing email Web service and an incoming email gateway. The client side is composed of a standard email client application and an XForms player component. The following sections describe the role of each of these components.

3.1 BPEL Execution Engine

The BPEL Execution Engine provides the workflow management capabilities. BPEL processes executed by this component interact with the external world through Web services [16]. A workflow process is specified in an XML-based language. BPEL defines a model and grammar that describes the behavior of a business process based on interactions
between the process and its partners. The interaction with each partner occurs through Web service interfaces. The BPEL process defines how multiple concurrent service requests from these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. By composing services into new, more complex Web services, BPEL allows creation of an heterogeneous distributed application [18]. The BPEL engine can run business processes for hours, days or months, and may invoke other long-running services. A BPEL process may contain steps that require waiting for external events or human interaction by invoking a Web service that handles this type of interaction. In this case, the BPEL process invokes the Lynx Web service as described in the following section.

3.2 Outgoing Email Web Service

Lynx’s outgoing email Web service provides the necessary services to interact with human partners through email. It dynamically generates the email message containing the
document sent when a process needs to interact with a human partner. The service accepts
documents to be processed by a human partner via its Web service interface. In response
the service automatically generates an electronic form for the document and send it as an
attachment to the human partner via email.

3.3 Other Partner Web Services

Other optional partner Web services may provide other services required by the
BPEL processes. These services can include document validation, external notifications,
transaction logging, document storage in an external database, and other external processes
such as transactions that need to be completed by a business process of another government
agency.

3.4 Incoming Email Gateway

An email server is periodically monitored by the Incoming Email Gateway that
listens for incoming email messages generated by interactions with human partners. It
forwards any received processed documents to the appropriate step within a running BPEL
process thus allowing it to continue its workflow.

3.5 Email Client

Any standard email client can be used to receive emails. The emails received by the
users have an attached document that are viewed with the XForms player component. The
MIME type of the attached document is defined as a custom application/type registered in
the client to be able to view it with the corresponding XForms player.
3.6 XForms Player

This component acts as a plug-in that renders the document received through email as an electronic form with controls that allow more sophisticated interactions than HTML forms. XForms allow data to be validated by the browser, such as types of fields being filled in, that a particular field is required, or that one date is later than another. XForms are device independent, meaning that the same form can be delivered without change to a traditional browser, a PDA, a mobile phone, a voice browser, and even an email client. Also, XForms are themselves XML documents that will be filled from other XML documents called instance data.

3.7 Document Repository

The documents are stored and maintained in a native XML database. Thus, documents can be inserted as XML, retrieved as XML using XQuery [25], and updated through XUpdate [26].

3.8 User, Role and Process Repository

Any standard relational database enables persistent management of process information for the BPEL execution engine. Authentication information for each user is also kept in this database. Also the role and email address for each user allows the process to send emails to any specific person that belongs to a certain role.
3.9 Workflow Querying Subsystem

The Workflow Querying Subsystem is a web-based interface that allows monitoring the status of documents pending processing, awaiting action, and shows documents successfully processed. It also serves as a redundant interaction mechanism in addition to email by providing web-based access to the XForms sent to the persons through email. Only users with appropriate access level depending on the role can submit, modify or view the documents awaiting for a human reply or validation in a workflow.
CHAPTER 4

Lynx Implementation

4.1 Introduction

This section describes implementation and integration details of Lynx in the specification of the sample business process from Figure 1.1. The integration of Lynx in a workflow process involves a process reengineering. This means that the processes by which the organization, the Registry of Deeds in this case, creates value and does work should be thought again and redesigned, ridding it of operations that have become antiquated. The process reengineering we propose involves the receipt of the documents in electronic format instead of in paper. In the Registry of Deeds actual computer-based system employees have to manually fill in the information from manuscript documents they receive at their offices, or from document submissions they receive via fax. Instead of increasing the throughput of transactions of the process, this current automation places more responsibilities on the reception and analyst employees. Conversely, if the documents were received in an electronic format, the complete process could be automated beginning in the notary public offices where the documents originate. This would make all the required information available from the start, resulting in a more agile business processes.
4.2 Document Definition

To be able to incorporate Lynx to a Web services based workflow, the document type XML schemas must first be defined. Document schemas consist of several pre-designed schema templates based on actual Registry of Deeds inscription minutes. These templates include common information that is used among different types of documents. For example, many Registry of Deeds inscription minute documents have the following features that are common with the over 160 different documents used in the Registry of Deeds:

- **Header.** Property number, town, inscription number, description, obligations, and title holders.
- **Entities.** The person or corporation that sells or buys involved in a transaction.
- **Presentation information.** Date and time presented, seat number, journal number, town.
- **Annotations, Log and Attachments.**

These definitions common to all the documents are placed in a master XML schema file that can be imported into any of the specialized XML schemas that implement the definition of the Registry of Deeds documents. The imported schema is then assigned a namespace \texttt{http://ece.uprm.edu/RegPropCommon} that identifies the set of elements and attributes of the data types to be used in another document. Accordingly, these schemas must be also imported into the BPEL process definition.

4.3 Process Description

The current Lynx prototype executes the workflows using the free, open-source, ActiveBPEL [4] implementation of BPEL. The BPEL process is deployed as a message-style Web service within an Apache Axis server used by the ActiveBPEL engine. Message-style
web services are used because they allow the creation of more document-centric interactions and allow the data to be expressed more naturally, when compared to XML-RPC. Also, message-style directly uses industry standard schemas, provides maximum power and extensibility, avoids building upon assumptions about implementation platform, and allows flexible mapping of platform data structures to XML. This lets the web services work without the need to use any Java-XML binding that would require SOAP encoding, and JavaBeans for each type of document. This approach enables us to have a single web service with a generic operation instead of many web services or a web service with many overloaded operations, one for each different type of emailed document. The outgoing email web service is made generic by having a single Java method processing any XML that arrives at the service by using a message-style provider. Therefore, the Web service would not need to be recompiled nor redeployed in case that a new type of document is sent to the outgoing email web service. Furthermore, our system is made extensible by allowing the addition of new document types to be included in the workflow and be accepted by the Web services. The only changes required are the BPEL description, including the processing of a new data type, and adding importing of the new data types XML schemas [23].

```xml
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://registro.egov.ece.uprm.edu">
  <xsd:complexType name="EmailInfoType">
    <xsd:all>
      <xsd:element name="to" type="xsd:string"/>
      <xsd:element name="from" type="xsd:string"/>
      <xsd:element name="subject" type="xsd:string"/>
      <xsd:element name="callback" type="xsd:string"/>
      <xsd:element name="class" type="xsd:string"/>
      <xsd:element name="body" type="xsd:anyType"/>
    </xsd:all>
  </xsd:complexType>
  <element name="EmailInfo" type="tns:EmailInfoType"/>
</xsd:schema>
```

Figure 4.1: XML Schema for Outgoing Email Web Service
The BPEL process invokes the Lynx web service by encapsulating the information necessary to communicate with a human partner inside a message that follows the EmailInfo schema depicted in Figure 4.1. The schema contains the destination human partner role that refers to the specific email addresses, subject, callback information, the name of the Java class that implements the appropriate XForms, and the specific document XML instance data payload. To achieve a generic web service the document payload is defined as a schema element of standard XML type anyType, thus accepting any document type. We show in Figure 4.2 an excerpt of the BPEL specification preparing and performing the web service invocation that will ask the human analyst to validate the documents and return it with any annotations, recommendations or attachments. The invoke operation only has one input variable, and has no output because the interaction with the Analyst was specified asynchronous. This is a design choice for the business process that is not required by Lynx. Nevertheless, asynchronous invocation is favored since an excessive wait time could cause a timeout in the BPEL execution engines Axis Web service operation.

Lynx’s outgoing Web service accepts arbitrary XML documents of type EmailInfo inside the body of received SOAP messages. This web service then extracts the encapsulated information from the body element, and uses it to construct an email message. The web service also needs to generate an XForms document specific to the type of document received and the type of interaction desired from the human partner. The XForms document can provide multiple views of the same instance data. Lynx supports this flexibility by implementing a Java interface (XFormsInterface) that includes a method that generates the desired form. The class that implements this interface is specified in the callback element of the EmailInfo message. This approach has the advantage of allowing the business process to choose the appropriate view for a particular transaction within a process. Lynx dynamically loads the class that implements the interface, instantiates it, and then invokes the method to create the specific XForm using the body part of the EmailInfo message.
<assign name="AnalystAssign">
  <copy>
    <!-- send to all persons of this 'role' -->
    <from expression="'analysts'" />
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/to" />
  </copy>
  . . .
  <copy>
    <from expression="'CancelacionHipotecaDirectaAnalystCompleted'"/>
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/subject" />
  </copy>
  <copy>
    <from expression="'CancelacionHipotecaDirectaAnalyst'"/>
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/callback" />
  </copy>
  <copy>
    <from variable="inputDocument"
      part="CancelacionHipotecaDirecta"/>
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/body" />
  </copy>
  . . .
</assign>

<invoke name="SendMail" partnerLink="email" portType="eml:SendMail"
  operation="sendEmail" inputVariable="emailInfo" />

Figure 4.2: Invoking an Analyst

as instance data. This newly generated XForms is sent as an attachment, with a custom
MIME type, to the email address specified in the To part of the EmailInfo message.

The outgoing email web service also keeps track of the specific correlation informa-
tion of a document. It specifies this information in the subject of the email message sent to
the human partners. This feature helps the email clients organize received email messages
by thread so the human partners can easily locate all the documents referring to a specific
case and/or document type as illustrated in 4.3.
Figure 4.3: Email Client showing threaded Lynx messages

At the client end, the XForms player component launches when an email message containing an XForms document attachment is received and the user opens the attached document. The email client knows it is a Lynx document that needs to be viewed in the XForms player due to the attachments custom MIME application/type.

In our current prototype, we use Chiba [10], an Open Source Java implementation of the W3C XForms standard. Chiba implements XForms by rendering them as standard HTML. The Chiba servlet is a component that runs on a servlet container such as Jakarta Tomcat [6]. Therefore, Tomcat must be running locally on the client computer to support the XForms rendering. Although Chiba is a servlet-based implementation, it implements the whole XForms standard unlike most current web browsers, and works with every browser unlike other client-side XForms implementations. In the future, when mainstream email and browser applications adopt and implement the complete XForms standard, the XForms player component using Chiba could be replaced with the email or web browser client, or a native plug-in that supports the whole standard. This will reduce the applications required at the client to a web browser or an email client alone.
The location of the XForms attachment is subsequently passed to a browser that submits it to the Chiba servlet for rendering. Figure 4.4 shows a sample XForms implemented in the prototype for validating and submitting an inscription minute that the Analyst must validate.

The Analyst returns the document back to the server side after completing, validating, annotating or adding any necessary attachments to the document received by pressing a submit button. This submission element defined in the XForms has an action pointing to a mailto: URL (see Figure 4.5). In response, Chiba submits via email the updated XML instance data of the corresponding document together will all attachments and annotations back to the incoming email gateway on the server side.

![XForms for a Mortgage Document](image)

Figure 4.4: XForms for a Mortgage Document

The Incoming Email Gateway will retrieve the revised document submitted by the Analyst from a standard POP3 email server. All the information necessary to access the
incoming email account can be found in a configuration file since the all returned email messages will be addressed to the BPEL engine itself. The incoming email gateway then invokes a callback web service exported by the business process in order to allow it to continue. This invocation requires producing a SOAP message of the correct operation within the BPEL process. However, what arrives in the email is only the XML instance data. Furthermore, there is no place to specify which operation to invoke in the SOAP message itself since the BPEL process is a message-style service. We solve this problem by dispatching the correct operation based on the type of message. We defined different SOAP message part names in the process’ WSDL interface definition using the same schema type. This is also done for every variable that is accepted for each different operation in the BPEL process definition. It can be seen in 4.6 that both messages have the same XML schema type, but they have different names, depending on which operation they are associated with.

```xml
<wSDL:message name="AnalystMessageName">  
    <wsdl:part name="CancelacionHipotecaDirectaAnalyst"  
        type="CHD:CancelacionHipotecaDirectaType"/>
</wSDL:message>  

<wSDL:message name="RegistradorMessageName">  
    <wsdl:part name="CancelacionHipotecaDirectaRegistrador"  
        type="CHD:CancelacionHipotecaDirectaType"/>
</wSDL:message>
```

Figure 4.6: WSDL Message Definitions

The type of message is specified by the *callback* part of the *EmailInfo* message that
was extracted when the XForms document was created. The message type is used as the root element in the body of the SOAP message returned to the BPEL process. Messages sent to a processes need to be delivered not only to the correct destination web service port, but also to the correct instance of the business process. The process dispatches the message to the appropriate operation within the correct process instance by using the BPEL correlations mechanism. The correlation information serves as an ID for a specific instance of a business process. For example, a social security number might be used to identify an individual taxpayer in a long-running multiparty business process regarding a tax matter. A social security number can appear in many different message types, but in the context of a tax-related process it has a specific significance as a taxpayer ID [8].

In the current prototype we use email-based submission and process operation invocation to start a workflow. The initial electronic version of the document produced by the notary public, or a receptionist at the Registry of Deeds that receives a manuscript, is created with empty XForms templates of the documents that are filled with the required information, and then submitted using exactly the same process used by an intermediary such as an analyst. Thus, in Lynx it there is no difference between the initial submission of a document and intermediate submissions made by any other human partner. In the case of an initial document submission, the Incoming Email Gateway will automatically recognize a new document in the inbox. The document type specifies that the first operation in the BPEL process must be invoked and the BPEL execution engine starts a new instance of the process. If a process with the same correlation information already exists, the submission is not accepted and a response is given by email notifying that a new process could not be instantiated because a document transaction with the same ID is already in progress.

The BPEL operation that waits for a response of the specific type specified in the callback will get invoked automatically. In this way the process resumes at the right point and continues along the process specification. Afterwards, it prepares the next human part-
ner interaction, and sends a request via the outgoing email web service as demonstrated previously in Figure 4.2. One minor disadvantage of this approach is that it requires any callback web service responding to email-based transactions to use document-style invocations, and declare multiple variables of the same type.

### 4.4 Workflow Querying Subsystem

In addition to the main architectural components of Lynx, described in Chapter 3, our current prototype contemplates additional functionality necessary to build a complete system that can enable the use of a web service based workflow in a government scenario. We are augmenting the basic Lynx Web service based workflows that can be constructed with BPEL to support authentication and transaction logging. A web service based workflow has no inherent concept of users. Thus, we store user information to provide for authentication in a database. Figure 4.7 shows the initial prototype of a web-based interface to monitor the status of documents pending processing, and shows documents successfully processed and registered in the Registry of Deeds.

Properly authenticated users, such as a supervisor, the registrar, a notary public or an analyst, can inspect the status of a transaction and monitor the progress of a document throughout the business process from within this interface. This transaction logging capability is implemented by having every XML document instance carry its own embedded log that records which steps the specific document instance has gone through. A `LogType` XML schema type imported into every documents schema is showed in Figure 4.8. The `LogEntryType` complex type includes the date, the process step finished, and who did it. A similar pattern is used for the attachments and annotations in a document. The attachments and a view of the document are accessible as well.

Figure 4.9 and Figure 4.10 show the log and attachments view, respectively, from
the document status web-based interface. Attachments are valuable since they can be used to include important documents needed for the completion of a specific case regarding a document, such as an image file of the property title deed itself, or any other relevant complementary documents.

Furthermore, ActiveBPEL allows the monitoring of the deployed BPEL processes. It provides a web-based administration tool that shows a graphical view of the current state of the workflow, including the process variables and step in the workflow. This is depicted in Figure 4.11.

4.5 XML Storage Subsystem

A persistent copy of all documents is maintained in both a database that is used by ActiveBPEL for process persistence, and in an eXist [?] Native XML database that is
used by Lynx to store the current version of each document. The benefit of a native XML Database is that we don’t have to worry about mapping our XML documents to some other data structure. While XML documents are organized as tree structures, relational databases organize data in a tabular or grid-like fashion, and use relational linking in order to expose hierarchical constraints on the data. Thus, a lot of flexibility is gained through the semistructured nature of XML and the schema independent model used by a native XML database such as eXist.

Data is just inserted as XML and retrieved as XML using XPath [24] and XQuery [25] as the query languages and the XUpdate [26] language to insert and update XML documents. This is particularly helpful since some complex XML document structures may be very difficult to map to a more structured relational database.

Finally, Lynx is made adaptable to any application requiring human interaction with a web service based workflow by giving the option to customize and configure many aspects of its functionality. Any BPEL process can call Lynx's web services to accomplish communication with human partners. The configuration options are kept in XML files so
that they can be easily modified with the parameters that suit the specific application. The different types of documents in use are specified along with their correlation information and relevant features (see Figure 4.12). The correlations are specified as XPath queries of the desired values in the document types XML Schema. The POP3 and SMTP server addresses to be used by the incoming email gateway and outgoing email web service, respectively, are also specified in other XML configuration files.
4.6 Customizing Lynx

A similar approach as the one presented for the Registry of Deeds scenario using Lynx could be easily applied to other government applications. The steps needed to integrate Lynx into a web services based workflow application are fairly straightforward. First, the XML schemas for the documents that will be handled by the process must be defined. Then, the WSDL interface for each of the partner web services the BPEL process will invoke, including Lynx’s outgoing email web service must be specified. Afterwards, the different operations that the BPEL process will expose must be defined in the WSDL. These operations are specified by having a different variable for the message that will accept each of the operations, since we use message-style web service operations. Then, the BPEL
process that defined the workflow desired should be specified. Also, the XML configuration files specifying the different types of documents, correlation queries, and email servers will need to be customized for the specific application. In addition, the implementation of the XForms for each view for each document must be done, including a Java class that implements the interface to create the XForms for each document. Finally, the partner web services, and the BPEL process itself are deployed. In summary, developing a completely different application requires very little code to be programmed and in particular will dramatically reduce the amount of custom GUI code required.
CHAPTER 5

Experiments and Results

5.1 Introduction

As proof of concept, two prototype systems were implemented, using the proposed Lynx architecture framework, for different government environments where a document-based workflow would be useful:


2. A workflow for the Puerto Rico Department of Justice Courts for lawsuit cases between two private parts.

This section presents some analyses and experiments that were conducted to assess the system’s performance, and advantages and disadvantages of the proposed architecture using the implemented prototypes for the sample Digital Government scenarios.
5.2 Deployment and Configuration for the Experiments

In order to implement the...

TODO

5.2.1 Server Configuration

The server configuration used to develop and carry out the experiments comprises the following hardware and software:

- **Pentium IV 2.4 Ghz CPU, 1 GB RAM**
- **Windows XP Professional SP2**
- **Java SDK 1.4.2.** Most of the third party software components used require at least Java 1.4.2.
- **Tomcat 5.0.28.** Tomcat is the official reference for the Java Servlet and JavaServer Pages technologies. Both Chiba and ActiveBPEL require a Tomcat 5.0 or greater version. Tomcat 5.5 may be used, but it introduces some XML library conflicts that must be solved by replacing some of the default jars, and forces the use of Java 5.
- **Chiba-web-1.0.0.** The Chiba-web-1.0.0 distribution was patched by modifying the source code of the `org.chiba.adapter.servlet.ChibaServlet` class and recompiling it to allow the loading of XForms documents from locations other than Chiba’s own application context inside Tomcat.
- **ActiveBPEL 1.1.6 server.** The ActiveBPEL engine is an Open Source implementation of a BPEL engine, written in Java. The ActiveBPEL engine runs in any standard servlet container such as Tomcat.
- **Email server (SMTP and POP3).** Currently using Apache Java Enterprise Mail Server (James) 2.2.0.
• **eXist 1.0-dev-20060316 XML Database server.** Exist XML DB was installed as a stand-alone server. It can also be installed as a webapp inside Tomcat, or as an embedded DB within the Lynx webapp.

• **MySQL 4.1 Database.** ActiveBPEL also supports, without customization, Oracle, DB2 and SQL Server databases to support process persistence.

### 5.2.2 Client Configuration

The clients must have installed any standard web browser, such as Internet Explorer or Mozilla Firefox, to be able to access Lynx’s web-based interface. However, in the current prototype, to be able of receiving Lynx emails with attached XForms the clients should have installed at least the Java Runtime Environment 1.4.2, an Email client and the Chiba XForms processor.

The current Lynx prototype has been tested with Mozilla Thunderbird, but also works with any email client. The custom Lynx attachment MIME type must be associated with the XForms Processor invoker.

Also, as explained in Section 4.3, the Chiba XForms processor implements XForms by rendering them as standard HTML through a servlet. Therefore, Tomcat 5.0 or later must also be running locally on the client to support the XForms rendering. Although Chiba is a servlet-based implementation, it implements the whole XForms standard unlike most current web browsers, and works with every browser unlike other client-side XForms implementations. In the future, when mainstream email and browser applications adopt and implement the complete XForms standard, the XForms player component using Chiba should be replaced with the email or web browser client, or a native plug-in that supports the whole standard for all browsers.
5.3 Experiments

As defined in chapter 2,..

5.3.1 The Experiments

A series of experiments were developed to assess the performance of the system in order to verify that the architecture achieves acceptable performance for the type of applications supported.

5.3.2 Amount of Code Required

Analysis of estimates of the amount of code required for a sample government application will be compared with estimates of that required for alternative architectures in order to test the hypothesis that the Lynx architecture can reduce the amount of custom code required for each new application.

5.3.2.1 Methodology

TODO

Comparison of amount of code required implementing a form using XForms vs JSF, Struts, etc?

5.3.3 Load Evaluation

Time required to process many simultaneous documents, for documents of different sizes??
5.3.3.1 Methodology

TODO

5.3.4 XML Database Performance Evaluation

Comparison of storing/extracting documents using native XML and RDBMS?

5.3.4.1 Methodology

TODO

5.4 Qualitative Analysis

We hypothesize that by exploiting XForms we will demonstrate the viability of implementing complex distributed interactive applications with significantly less coding. Additionally, simple modifications to the interaction screens will often not require re-programming of GUI code. The following sections present a comparison of deployment details between Struts, JSF and Lynx.

5.4.1 Steps needed to deploy an application using Struts

1. Create development directory structure
2. Write web.xml
3. Write struts-config.xml
   - Identify required input forms and then define them as form-bean elements
   - Identify required Action’s and then define them as action elements within actionmappings element
4. Write ActionForm classes
   - Extend org.apache.struts.action.ActionForm class
   - Decide set of properties that reflect the input form o Write getter and setter methods for each property
   - Write validate() method if input validation is desired

5. Write Action classes o Extend org.apache.struts.action.Action class
   - Handle the request
   - Decide what kind of server-side Model objects (EJB, etc.) can be invoked
   - Based on the outcome, select the next view

6. Create resource file

7. Write JSP pages

8. Build, deploy, and test the application

5.4.2 Steps needed to deploy an application using JSF

1. Create development directory structure

2. Write web.xml

3. Create the Pages using the UI component and core tags
   - Lay out UI components on the pages
   - Map the components to backing beans (model object data)
   - Add other JSF features (either as tags or attributes)

4. Define Page Navigation in the application configuration file

5. Develop the backing beans Model objects
   - Model objects hold the data (JavaBeans, etc.)
   - Validator, convertor, event handler, navigation logic

6. Add managed bean declarations to the application configuration file
7. Build, deploy, and test the application

The problem with these frameworks is that all the validation and logic needs to be programmed in Java: the Actions, the validations, the Beans, etc. Essentially, both Struts and JSF are just Model-View-Controller frameworks for building HTML forms, validating their values, invoking business logic, and displaying results in HTML.

5.4.3 Steps needed to deploy an application using Lynx

1. Define XML schemas for the documents that will be handled by the process.

2. Specify WSDL interface for each of the partner web services the BPEL process will invoke, including Lynx’s outgoing email web service.

3. Define the different operations that the BPEL process will expose in the WSDL of the process.

4. Specify and implement the BPEL of the desired workflow process.

5. Customize the XML configuration files specifying the different types of documents, correlation queries, and email servers. Specify users in the Lynx administration page. If necessary, customize the web-based document status and task list interface to suit the specific application to be supported.

6. Implement the XForms for each view for each document. Create a Java class that implements the interface to create the XForms for each document.

7. Deploy partner web services and the BPEL process itself.

In summary, Lynx provides an open architecture that extends web service based workflows with human interaction through XForms. Developing an application using Lynx will require very little code to be programmed. In particular, we have demonstrated Lynx may dramatically reduce the amount of custom GUI code required since many common tasks such as marking controls as required, performing validations and calculations, displaying
error messages, and managing dynamic layout can be done without the need of scripting, in a declarative way. Also, Java-XML mapping overhead is not required since the documents are kept in XML throughout the whole application since the instance data used in the XForms is XML, the web services are message-style (not XML-RPC style), and document data is stored as XML in a native XML database. Almost everything in Lynx is limited to languages based in XML such as BPEL, XForms, and XML Schema and WSDL.

5.5 Results

TODO

Our analysis and experiments suggest that ...

Lynx would be more useful when it is used for a document-based application where the documents must be routed from desk to desk, and a go through a number of steps as different people validate or add content to the document. Lynx is also appropriate when work is directed to specific users, when the application desired involves a long-running and complex business processes, rules, steps, forms. Lynx also allows complete interaction through email may be desired, and not only email-based notifications. Finally, An architecture like Lynx would be useful when the application also involves the invocation of different web services to complete a process.

However, currently Lynx has several limitations. Although no coding is necessary for validations, coding may be necessary for other logic that may be needed through web services. Response may not be instantaneous if working through email. Also, may not replace a traditional servlet for processing any incoming request if working through the web-based interface given that it can only process a request for the specific message types that the BPEL process is waiting for. Also, Java classes that return the corresponding
XForms are needed. They could be replaced by automatic XForms generation from XML Schemas but this approach does not provide for the level of customization needed to deliver a specific view of a document to certain person at certain time. Finally, there is also a learning curve for BPEL, XForms, and other XML technologies.
CHAPTER 6

Conclusions and Future Work

We have presented the design and implementation of Lynx, an open architecture that extends web service based workflow engines with human interaction via email. Lynx uses a general purpose email messaging architecture to interact with human partners by using the BPEL language for specifying business process workflow behavior based on web services. Lynx uses XForms to minimize the amount of custom code required to implement the user interfaces.

6.1 Research Conclusion

We have presented the...

6.2 Future Work

Dynamic Generation of XForms?

Implement e-mail client plugins, or use Firefox’s XForms implementation, when complete, to eliminate server-side Chiba?
Extend the BPEL language to support human endpoints natively (see bpel4people)?

Security

Use more XQuery scripts instead of servlets with XQuery queries to get documents from DB as XML?

etc

6.3 Final Word

As final word...
BIBLIOGRAPHY


APPENDICES
APPENDIX A

Appendix Title1