

**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

Date

Jaime Seguel, Ph.D.
Member, Graduate Committee

Date

Bienvenido Vélez, Ph.D.
President, Graduate Committee

Date

Ana Carmen González, M.S.
Representative of Graduate Studies

Date

Isidoro Couvertier, Ph.D.
Chairperson of the Department

Date

ABSTRACT

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

By

Iván P. Vélez-Ramírez

Web service based workflows provide support for aggregating web services into new higher-level web services, but do not support direct interaction with people. On the other hand, traditional collaboration tools like email or instant messaging do not provide the necessary support for structured business processes. In addition, building or modifying new workflow application with current software tools is expensive and time consuming since it requires customized programming process for the presentation, validations, and required business logic.

We introduce Lynx, a new architecture for workflow systems based on Web Services that leverages on current standards and broadly known technologies such as XML, e-mail, BPEL and XForms, to reduce the amount of code and thus the cost and time of developing and maintaining a Web-based workflow application, and interact with human partners via e-mail based forms without requiring a specialized client. We illustrate the usefulness Lynx in a Digital Government scenario.

RESUMEN

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

Por

Iván P. Vélez-Ramírez

Flujos de trabajo basados en servicios Web proveen apoyo para agregar servicios Web en servicios de más alto nivel, pero no apoyan interacción directa con personas. Por otro lado, herramientas tradicionales de colaboración como correo electrónico o mensajes instantáneos no proveen apoyo para procesos estructurados. Además, desarrollar o modificar nuevas interfases de aplicaciones de flujos de trabajo con herramientas tradicionales es costoso y requiere mucho tiempo por la programación para la presentación y validaciones, y por la lógica del proceso requerida.

Introducimos Lynx, una arquitectura nueva para sistemas de manejo de flujo basados en servicios Web que se basa en estándares ampliamente conocidos como XML, email, BPEL y XForms, para reducir el costo y tiempo que toma desarrollar y mantener una aplicación de flujo de trabajo, y comunicarse con personas a través correo electrónico con una interfase basada en formularios sin requerir un cliente especializado. Mostramos la utilidad de nuestro método en un escenario de Gobierno Digital.

Copyright © by
Iván P. Vélez-Ramírez
2006

To my parents, for their unconditional support

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Bienvenido Vélez, for his support, and for sharing his valuable time, comments, recommendations, and advice to carry out this research. I also wish to thank the members of my committee, Dr. Manuel Rodríguez and Dr. Jaime Seguel for their support, suggestions, comments and reviewing my work. Thanks to the graduate studies representative, Prof. Ana Carmen Gonzalez, for her comments on the thesis.

A special thanks to Prof. Rafael Fernández-Seín for his advice and originally suggesting the idea of a workflow system for the Registry of Deeds that later became a scenario for developing an application using Lynx, and for his continued support during my years working at the Space Information Laboratory (SIL).

This work was supported by the UPRM Digital Government project, under the NSF Grant EIA-0306791.

TABLE OF CONTENTS

LIST OF FIGURES	x
1 Introduction	1
1.1 Overview	1
1.2 Problem Statement	4
1.3 Proposed Solution	5
1.4 Research Objectives	6
1.5 Summary of Contributions	7
1.6 Thesis Structure	7
2 Related Work	9
2.1 Web Services	9
2.2 Business Processes Using Web Services	10
2.3 Web Forms	14
2.3.1 HTML Forms and other Web Form technologies	14
2.3.2 XForms	15
2.4 Native XML Databases	21
3 Lynx Architecture	23
3.1 BPEL Execution Engine	24
3.2 Outgoing Email Web Service	25
3.3 Other Partner Web Services	25
3.4 Incoming Email Gateway	25
3.5 Email Client	26
3.6 XForms Player	26
3.7 Document Repository	27
3.8 User, Role and Process Repository	27
3.9 Workflow Querying Subsystem	27
3.10 Chapter Summary	28
4 Supporting Human Interaction through Email with Lynx	29

5	Developing an Application using Lynx	37
5.1	Introduction	37
5.2	Business Process Re-engineering	39
5.3	XML Workflow Document Definition	39
5.4	External Interface Description	43
5.5	XForms Development	46
5.6	Process Description	47
5.7	Web-based Interface to the Workflow Application	53
5.8	XML Storage Subsystem	60
5.9	Developing a New Application Using Lynx	61
6	A Workflow-based Application to Track the Flow of Legal Cases in the Puerto Rico Judiciary	63
6.1	Introduction	63
6.2	Supporting Software Substrate	64
6.3	Document Definition	67
6.4	External Interface Description	71
6.5	XForms Development	72
6.6	Process Description	79
6.7	Web-based Interface	80
6.8	XML Storage Subsystem	82
6.9	Summary	83
7	Experiments and Results	84
7.1	Introduction	84
7.2	Deployment and Configuration	84
7.2.1	Server Configuration	84
7.2.2	Email Server	85
7.2.3	Client Configuration	85
7.3	Performance Evaluation	85
7.3.1	Methodology	86
7.3.2	Results	87
7.4	Qualitative Analysis	93
8	Conclusions and Future Work	103
8.1	Research Conclusion	103
8.2	Future Work	104
	BIBLIOGRAPHY	105

APPENDICES	109
A Experiment BPEL Tasks	110

LIST OF FIGURES

1.1	Sample Registry of Deeds Scenario	2
2.1	XForms Summary	16
2.2	Sample XForms code	17
2.3	XForms Model connecting to many possible user interfaces	18
3.1	System Architecture	23
4.1	Lynx's Email Gateways	30
4.2	XML Schema Diagram for Outgoing Email Web Service	31
4.3	The <i>XFormsCreator</i> Java Interface	32
4.4	Email Client showing threaded Lynx messages	33
4.5	Sample XForms Email Submission Element	34
4.6	Sample WSDL Message Definitions	35
5.1	Sample Registry of Deeds Scenario	38
5.2	Microsoft Word Template of a PRRD Document	40
5.3	XML Schema Diagram for a Mortgage Cancellation Document	41
5.4	XML Schema Diagram Containing Common Types Used Among Documents	42
5.5	WSDL Message Definition for the PRRD Process	43
5.6	WSDL Operation Definitions for the PRRD Process	44
5.7	WSDL Correlation Definitions for the PRRD Process	45
5.8	WSDL PartnerLinkType Definitions for the Lawsuit Process	45
5.9	AnalystXForm Java Class Implementing XFormsCreator Interface	46
5.10	XForms for the Mortgage Cancellation Document	47
5.11	BPEL Components for the PRRD Process	48
5.12	Partner Links Definition in the BPEL Process	49
5.13	Partner Definition in the BPEL Process	49
5.14	Variables in the BPEL Process	50
5.15	Correlation set in the BPEL Process	50
5.16	PRRD BPEL Process main activity	51
5.17	Invoking the Outgoing Email Web Service to Interact with an Analyst	52
5.18	Receiving the Document that Results from the Interaction with an Analyst	53
5.19	E-R Diagram for Lynx Database	54
5.20	Log XML Schema	55
5.21	Document Status Page	56
5.22	Using JavaBeans with JSTL in the JSP web interface	57

5.23	Document Status Log View	58
5.24	Document Status Attachments view	58
5.25	Workflow Process Detail Graph	59
5.26	Document Type Configuration	61
6.1	Flowchart of a lawsuit case	64
6.2	XML Schema Diagram for Schema Types Common among Documents . . .	68
6.3	XML Schema Diagram for a Lawsuit Document	69
6.4	XML Schema Diagram for an Answer Document	69
6.5	XML Schema Diagram for a Summons Document	70
6.6	XML Schema Diagram for a Motion Document	70
6.7	WSDL Message Definition for the Lawsuit Process	71
6.8	XForms for a Lawsuit Document	73
6.9	XForms for a Summons Document	74
6.10	XForms for an Answer Document	75
6.11	XForms for a Motion Document	76
6.12	XForms for a Preliminary Conference Report Document	77
6.13	XForms for Validating a Lawsuit Document	78
6.14	Lawsuits status web page	80
6.15	Specific case status page	81
6.16	Lawsuit Process Document Type Configuration	82
7.1	Average documents processed per minute	87
7.2	Average time taken by each task for 2 KB documents	88
7.3	Average time taken by each task for 70 KB documents	89
7.4	Average time taken by each task for 200 KB documents	90
7.5	Average time taken by each task for random-sized documents	91
7.6	Time each task takes as percentage of total time	92

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 improving 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 through the process.

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 manually 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 such as 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.

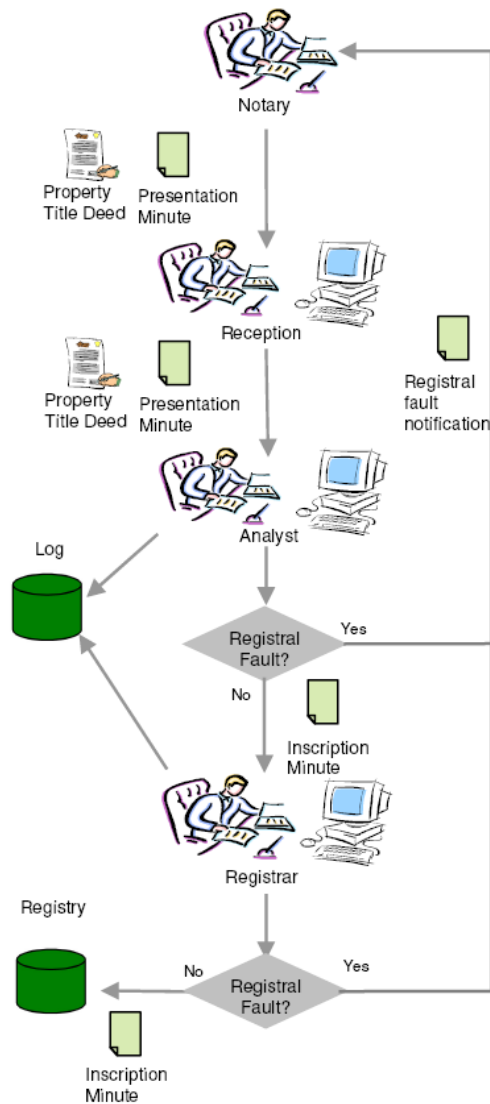


Figure 1.1: Sample Registry of Deeds Scenario

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. The document is then 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.

Our 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 and weakly connected 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 [37]. 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 [37]. Also, from a technical standpoint, email operates using simple ubiquitous 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 a government agency, building a workflow is sometimes indispensable. However, the business process expert who generally has little or no programming skills must resort to a software vendor to customize the business application. This procedure is expensive and time-consuming [43].

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 [15]. 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 [17] 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 [34]. This approach often provides insufficient support for direct or synchronous interaction with persons. Collaboration tools like email or instant messaging by themselves do not provide the necessary support for structured business processes.

Building a new workflow application user interface with current software development tools, or modifying an existing one, is expensive and time consuming since it requires

a complex customized programming process to implement the graphical user interface document validations, and to embed the business logic. Traditional approaches such as the usage of HTML forms have limited features, require scripting to accomplish common tasks, and integrate poorly with XML. In addition, traditional HTML forms introduce more complexity in programming Web applications. This complexity originates from several main sources [13]. 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, JavaServer Pages with taglibs, servlets, Struts, XSLT, DOM, SOAP, Web Services, Enterprise JavaBeans, 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 this complexity and overhead is of particular importance in Digital Government environments where programming skills are severely scarce and difficult to hire.

1.3 Proposed Solution

We propose Lynx, a new architecture for workflow systems based on Web Services that leverages on current standards and broadly known technologies, such as email, XML and XForms, in order to reduce the amount of code and thus the cost of developing, deploying and maintaining a Web-based workflow application. Lynx stores 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 (a.k.a. thick) 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 also 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 [28] standard and generates each form based on the type of document being transported and the role of the user accessing it. The request for the transaction is generated by a business processes typically specified using a language such as the Business Process Execution Language (BPEL) [11]. 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 support email-based access 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 Research Objectives

The specific objectives of this research are:

- Design a general purpose email messaging architecture to allow human partners interaction with the BPEL workflow processes.
- Develop a general purpose outgoing email Web service.
- Develop an incoming email gateway to the BPEL engine.
- Develop a prototype implementation of the Lynx architecture.

- Design and develop two Digital Government applications based on Lynx.
- Define and implement BPEL processes specifying the workflow of a subset of documents most widely used by each Digital Government application.
- Design and implement XForms user interfaces for all required documents.
- Conduct a series of analysis on the system's performance in the experimental Digital Government applications.

1.5 Summary of Contributions

The main contribution of this research is the design and development of Lynx, a general purpose email messaging architecture to interact a Web-services based workflow with human partners by leveraging on current standards and broadly known technologies such as BPEL, Email, XML standards and XForms. Lynx also leverages on Email, XForms and a Native XML Database to reduce the amount of custom code necessary to develop the user interfaces of a document-based workflow application from scratch. We illustrate the usefulness of the Lynx architecture with the design and development of two Digital Government applications. Also, a performance evaluation of the architecture was made to verify that the architecture achieves acceptable performance for the type of applications supported. Finally, as a result of this research two papers were published and presented at the IEEE 2006 International Conference of Internet and Web Services Applications (ICIW/WEBSA 2006) [23] and the ACM 7th International Conference on Digital Government Research (DG.O 2006) [22] conferences.

1.6 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 discusses related work that serves as basis for our research. Chapter 3 presents an overview of the Lynx

architecture and its components. In Chapters 4, 5 and 6 we present a detailed description of Lynx implementation and integration with a Web services based workflow. Chapter 7 presents experimental results from an evaluation and analysis of Lynx. Finally, Chapter 8 presents a summary of our contributions, conclusions and suggests directions for 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 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[42]. The 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) [46] is used for cross-platform inter-application communication. The Web Services Description Language (WSDL) [6] is used to describe the online services.

2.2 Business Processes Using Web Services

Business Process Execution Language for Web Services (BPEL4WS, or BPEL) is a standard language to specify how Web services are combined into higher-level business processes by describing workflows or orchestrations of Web services invocations [34].

The BPEL data model is built on top of WSDL 1.1 messages and XML Schema 1.0 [44] 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 [34]. 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 provides increased flexibility. Such compositions can adapt more quickly to changing business needs, compared to hard-coded applications [34]. BPEL allows modularization and decomposition of structured activities by working through well-defined interfaces.

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 [50]. 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. [50] 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 [8] allows the transmission of SOAP messages via email through SMTP. Its intention is to send Web service messages between two Web services and not between a Web service and a human user via email. 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. Lynx acts as a gateway between a Web services based workflow process and a human partner.

Chakraborty [15] 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. However, they had to extend IBM's BPWS4J [20] 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 [52] 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 [38] 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 [3] 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 2004, 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 [35] that is mainly used for notifications. Thus, it cannot receive messages through email. Recently, BizTalk Server 2006 introduced a new built-in adapter for POP3 protocol [30]. The POP3 adapter enables BizTalk applications to retrieve e-mails and their attachments from a mailbox using the POP3 protocol.

The BEA WebLogic Integration Business Process Management [4] provides a work-

flow management system to automate business processes. BEA workflow interacts with system users via a Worklist or a custom client application using the following methods: directly through the Worklist or custom client, internal XML/JMS messaging with the Worklist or a custom client application to perform additional operations, or e-mail. The e-mail capabilities only allow sending e-mail to clients outside the WebLogic Integration system.

Oracle Workflow [32] provides a complete business process management system that supports business process definition, business process automation, and business process integration. Oracle Workflow enables modeling, automation, and continuous improvement of business processes, routing information of any type according to user-defined business rules. Oracle Workflow has outgoing and incoming email notifications. However, the incoming responses that are submitted by email should either follow some specific syntax carefully when formatting the reply, or provide links to external resources.

IBM WebSphere MQ Workflow supports long-running business process workflows as they interact with systems and people. Communication is done using the MQ protocols and the workflow worklist through web portals. It also features an email adapter [21] that has both error notification and business object processing. The email adapter can receive emails with business objects but they need to be in a specific format, and a handler must be programmed for each different type of business object.

Similarly, many other existing applications do not include complete interaction through email. Most only provide notifications through email or instant messaging. Both, Bonita [5] and YAWL [51] feature a notification service through an instant messaging protocol. jBPM [27] specifies the business process using an ad-hoc language called jPDL, and only supports simple notifications through Java Messaging Service.

2.3 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.

2.3.1 HTML Forms and other Web Form technologies

According to the HTML specification [19], 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 logic and presentation
- Limited accessibility features
- Limited client side validation

Additionally, some frameworks like the Java Server Faces (JSF) [18] aim to bring rapid user interface development with server-side Java by allowing to create user inter-

faces 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 concrete, 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 [24], or Asynchronous JavaScript + XML, have been changing the way interaction is done on the Web. Ajax incorporates presentation using XHTML and CSS, dynamic display and interaction using the Document Object Model, data interchange and manipulation using XML and XSLT, asynchronous data retrieval using XMLHttpRequest, and JavaScript binding everything together. However, Ajax applications involve running complex JavaScript code on the client, thus it is technically difficult to make that complex code efficient and bug-free.

2.3.2 XForms

XForms is an XML application that represents the next generation of forms for the Web [28]. 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 2.3.1:

- 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 logic from presentation
- Universal accessibility
- Supports client-side validation

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 summarizes the main aspects of XForms.

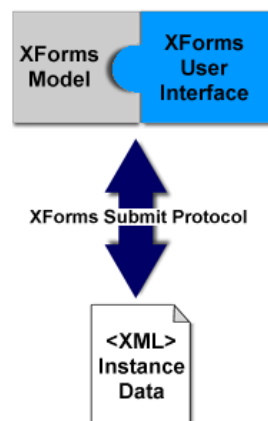


Figure 2.1: XForms Summary

Figure 2.2 shows a sample XForms showing the model, the instance, the submission

mechanism and a small part of the user interface as a body element in an XHTML container.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<html xmlns="http://www.w3.org/2002/06/xhtml12"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xf="http://www.w3.org/2002/xforms"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <head>
    <title>Demanda</title>
    <xf:model id='doc'>
      <xf:instance xmlns=''>
        <Demanda>
          . . .
        </Demanda>
      </xf:instance>
      . . .
      <xf:submission action="mailto:egov@localhost?server=
        localhost&sender=i_velez@localhost&subject=
        InitialSubmission" id="s01" replace="none" method="post" />
    </xf:model>
  </head>
  <body>
    . . .
    Demandante: <xf:input ref="Header/Demandante/Name"/>
    . . .
    <xf:trigger>
      <xf:label>Submit Document</xf:label>
      <xf:action> <xf:send submission="s01"/> </xf:action>
    </xf:trigger>
  </body>
</html>
```

Figure 2.2: Sample XForms code

Figure 2.3 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. Since XForms is just a specification for defining user interfaces in XML, one could use XForms to define the user interface of a Java Swing-based form, or even Voice Browser input form. The implementations would take care of the rendering and implementation details.

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 Scalable Vector Graphics. Other user interface components for XForms may also be developed.

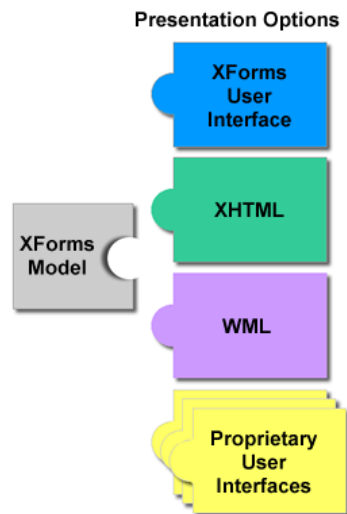


Figure 2.3: XForms Model connecting to many possible user interfaces

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 [28].

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 submitted 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.

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 the government.

There are technologies competing against XForms. The most relevant is InfoPath [31], 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 [29]. 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 [29].

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 have also used XForms to simplify Web programming. C. Richard, et. al. [13], 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 (OPS) [33] 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 suited for the tasks of capturing, processing, and presenting XML data, and does not require writing any Java or scripting code to implement a presentation layer for a web application. OPS is built around an Ajax-based XForms engine, and the XPL engine, a proprietary XML pipeline engine for processing XML data.

2.4 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 [1]. 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 [1].

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 [39].

In a performance analysis between an XML-enabled and a native XML database, [1] 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. [1] 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 [26], 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

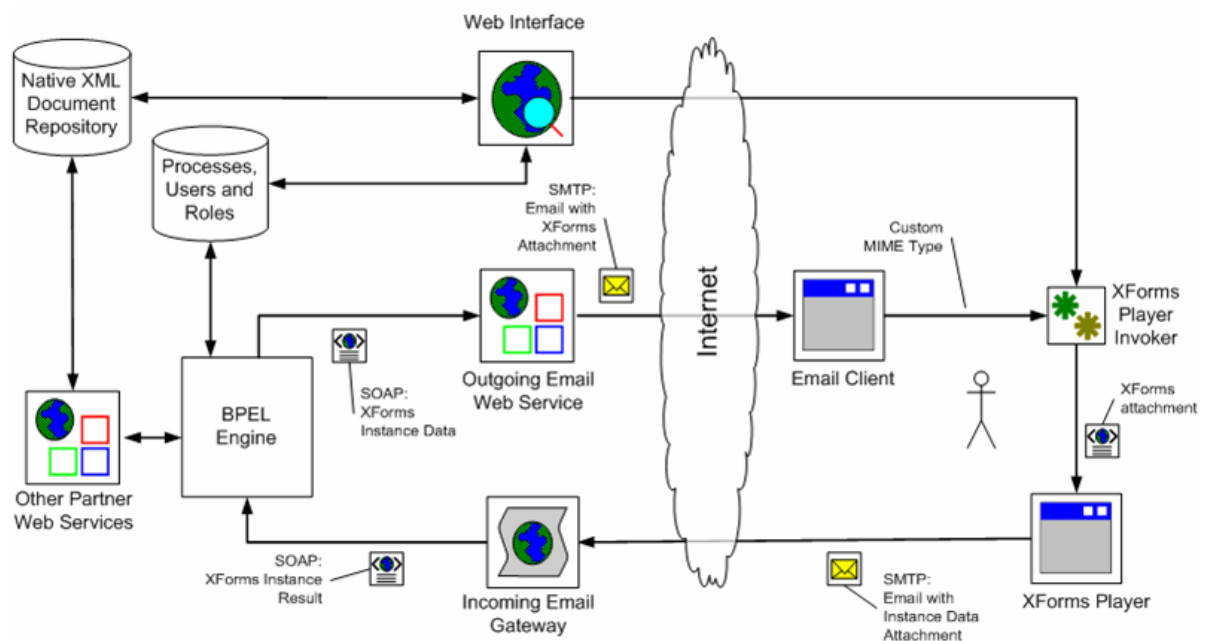


Figure 3.1: System Architecture

Figure 3.1 shows the different elements that comprise the Lynx architecture. The execution 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.

Lynx uses the outgoing email Web service to generate an email with an XForm that human partners use to interact with the process. The Incoming Email Gateway forwards the processed document resulting from the human interaction through the XForms to the appropriate step in the process. The documents are also stored in a native XML database. This allows documents to be viewed through the workflow querying subsystem's Web-based interface as an alternative path of interaction, in addition to Email.

Essentially, Lynx includes two new modules: an outgoing email Web service and an incoming email gateway. The following sections describe the role of each of these components. The server side is composed of a BPEL execution engine, an outgoing email Web service and other partner Web services, an incoming email gateway, an XML document repository, a process repository, and a workflow querying subsystem. The client side is composed of a standard email client application and an XForms player component required to render and process the XForms.

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 [34]. 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 [36]. 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 sends it as an attachment to the human partner via email. From the standpoint of the BPEL Engine, the Outgoing Email Web Service looks just like any other partner Web service.

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.

Eventually we expect all Web browsers and email clients to directly support the XForms standard. For instance, the Mozilla Firefox browser already supports part of the standard through an extension, and several plugins exist for other browsers.

3.7 Document Repository

Lynx documents are stored and maintained in a native XML database. Thus, documents can be inserted as XML, retrieved as XML using XQuery [48], and updated through XUpdate [49]. This allows the insertion and extraction of documents in XML without any conversion overhead.

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 all persons that belongs to a given role.

The use of a relational database is not necessary since the user and role information can be stored in the Native XML database. We use the relational database because the BPEL execution engine requires one to store the processes state information.

3.9 Workflow Querying Subsystem

The Workflow Querying Subsystem is a web-based interface that allows monitoring of 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.

3.10 Chapter Summary

In summary, using Lynx, workflow applications can be developed for the most part by implementing a BPEL process and XForms user interfaces and without requiring sophisticated Java code or scripting by leveraging on current standard and broadly known technologies, such as email, XML and BPEL, in order to reduce the amount of code and thus the cost of developing and maintaining a workflow application. In the following chapters, we explain in detail the process involved in supporting human interaction through email, and developing a real workflow-based application following the Lynx architecture.

CHAPTER 4

Supporting Human Interaction through Email with Lynx

This chapter discusses the details concerning Lynx's support for Web-service based workflow human interaction through email. Adding Email support to a Web-services based workflow is needed because human user interactions are currently not supported 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 [50]. A BPEL process can invoke Web services through Email. However, these Emails only serve as the transport mechanism for the SOAP messages, and not as the interaction mechanism to reach human partners. Emails used as transport are not designed to be human-readable.

Figure 4.1 shows Lynx's support for human interaction by providing an outgoing email Web service to generate an email with an XForms document that human partners use to interact with the process. After the user submits the form back, an incoming email gateway receives, decapsulates, and forwards the document resulting from the human interaction through the XForms to the appropriate step in the process.

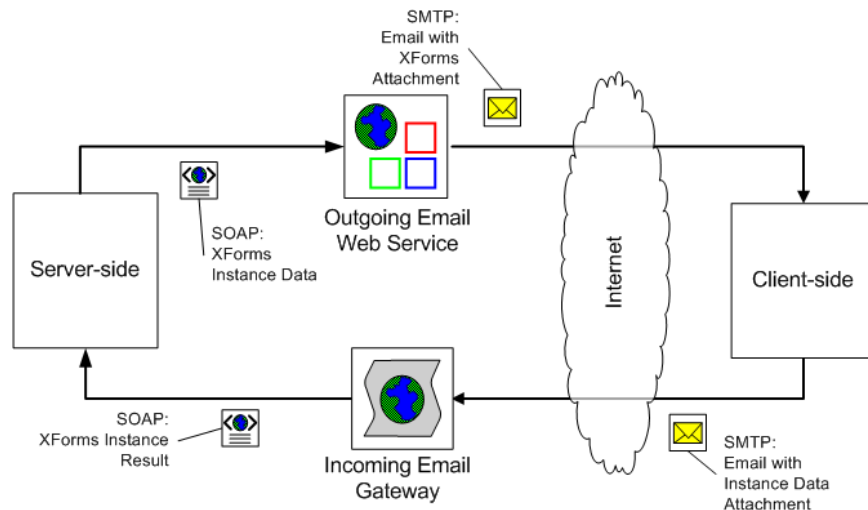


Figure 4.1: Lynx's Email Gateways

In our design we chose to use message-style [34] over the alternative RPC-style Web services to support sending email from a Web-services based workflow, such as one defined by BPEL. RPC-style uses Remote Procedure Calls SOAP invocations. RPC-style would be ideal when one is starting from Java code and is trying to expose methods as Web services. Also, if there is the need to transmit object graphs, such as circular lists or complex graphs, RPC provides an interoperable way to do it. On the other hand, message-style is preferable if there are existing XML data formats where using RPC-style and SOAP encodings would just get in the way. The problem with message-style is that it does not use the standard SOAP encodings for arrays and structures. Therefore, the benefit of referential integrity of objects referenced multiple times is lost. However, this is not the case for Lynx. Therefore, message-style Web services are used in Lynx 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,

reducing significantly the amount of Java code required.

Message-style Web services enable us to have a single outgoing email 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 would be in the BPEL description, including the processing of a new data type, and adding importing of the new data types XML Schemas[44].

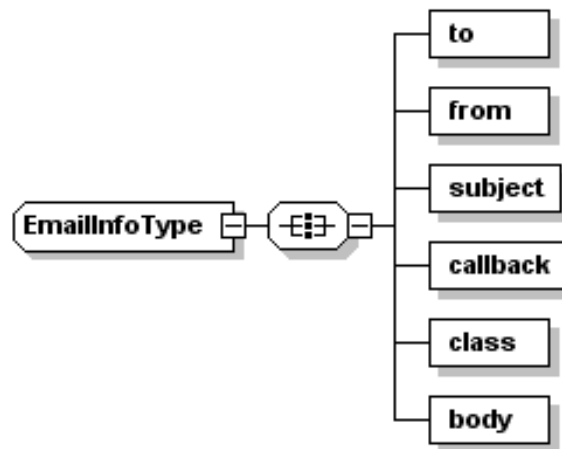


Figure 4.2: XML Schema Diagram for Outgoing Email Web Service

The Web services workflow process invokes the Lynx outgoing email Web service by encapsulating the information necessary to communicate with a human partner inside a message that follows the *EmailInfo* schema depicted in Figure 4.2. The above XML Schema diagram shows the name of the XML type *EmailInfoType* containing *all* the elements to the

right. The schema contains the destination human partner role that refers to the specific email addresses in element *to*, the *subject*, *callback* information, the name of the Java class that implements the appropriate XForms in the *class* element, and the specific document XML instance data payload in the *body* element. 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.

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 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 documents can provide multiple views of the same instance data. Lynx supports this flexibility by implementing a Java interface (*XFormsCreator*) that includes an operation named *setupXForms* that generates the desired form and returns it as a String (see Figure 4.3). The class that implements this interface is specified in the *class* element of the *EmailInfo* message. This will be explained in more detail in the next chapter.

```
public interface XFormsCreator {
    public abstract String setupXform( Document instance,
                                     String to,
                                     String from,
                                     String subject,
                                     String callback );
}
```

Figure 4.3: The *XFormsCreator* Java Interface

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 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 information of a document. 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 [11]. The outgoing email web service 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 Figure 4.4.

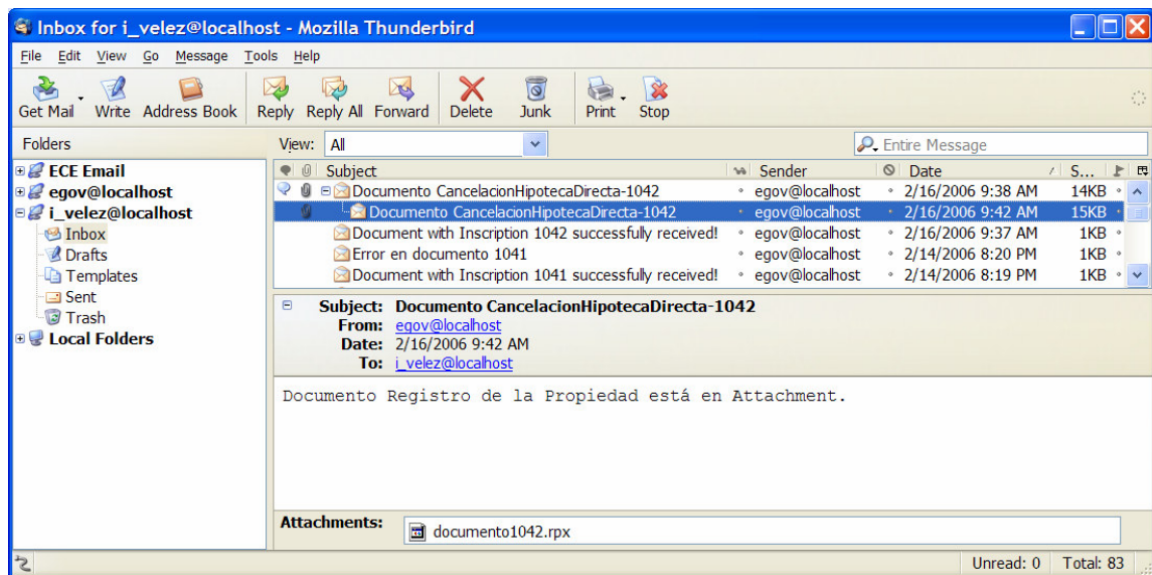


Figure 4.4: Email Client showing threaded Lynx messages

At the client end, the XForms player component launches when a Lynx 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.

The location of the XForms document is then passed to the XForms player for rendering. Then, after completing, validating, annotating or adding any necessary attachments to the received document, the the document is returned back to the server side by pressing the XForms' *Submit* button. This submission element defined in the XForms has an action pointing to a *mailto:* URL (see Figure 4.5). In response, the XForms player submits via email the updated XML instance data of the corresponding document together will all attachments.

```
<xf:submission
  action="mailto:receiver@ece.uprm.edu?server=ece.uprm.edu&
  sender=sender@ece.uprm.edu& subject=Property Registry Document
  id=s01 replace=none method=post />
```

Figure 4.5: Sample XForms Email Submission Element

The Incoming Email Gateway, that is continuously monitoring an Email inbox used by Lynx, will then retrieve the revised document submitted by the human partner. All the information necessary to access the incoming email account can be found in a configuration file since all the 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 Figure 4.6 that both messages have the same XML Schema type, but they have different *names*, depending on which operation they are associated with.

```
<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: Sample WSDL Message Definitions

The type of message is specified by the *callback* part of the *EmailInfo* message that was extracted by the outgoing email Web service 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 process 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. 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.

There is no difference in Lynx 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.

The next chapter will show how to use the Lynx's support for human interaction through email discussed in this chapter to build and deploy a Web-service based workflow including human partners.

CHAPTER 5

Developing an Application using Lynx

5.1 Introduction

This chapter describes implementation and integration details of Lynx in the specification of the sample Puerto Rico Registry of Deeds (PRRD) business process previously explained in Section 1.1. For convenience we are repeating the graphical description of the process in Figure 5.1.

A workflow application using Lynx uses BPEL to describe the process interactions and service invocations. BPEL is layered on top of several XML specifications: WSDL, XML Schema, and XPath. WSDL messages and XML Schema type definitions provide the data model used by BPEL processes. XPath provides support for data manipulation. All external resources and partners are represented as WSDL services. Therefore, the steps involved in the implementation of a BPEL workflow-based application on top of Lynx require a re-engineering of the original business process, defining the XML Schemas of the

messages exchanged, defining the WSDL interfaces of the process, using XPath inside the process to refer to data, and calling Lynx's Web services.

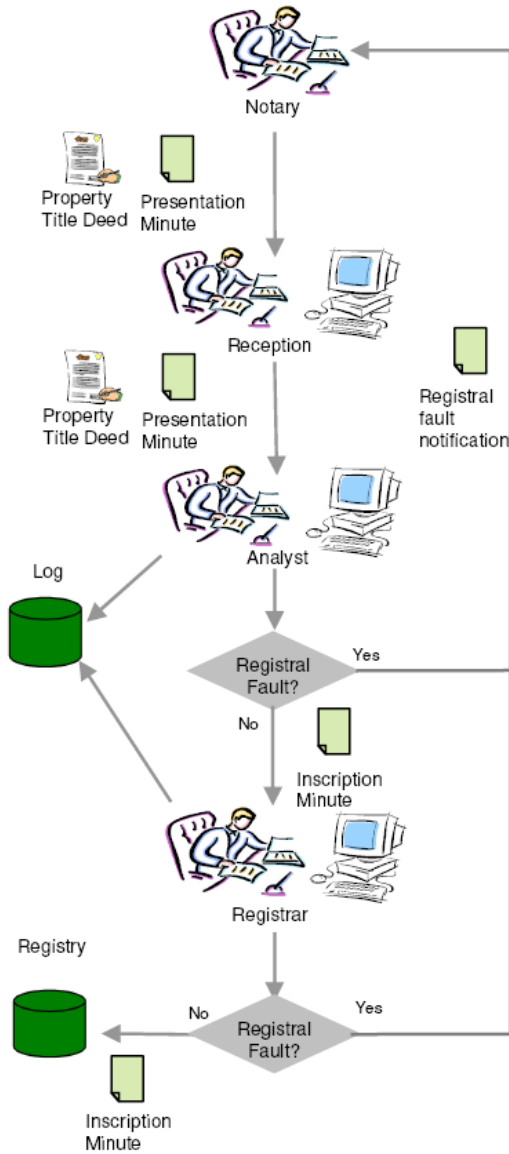


Figure 5.1: Sample Registry of Deeds Scenario

5.2 Business Process Re-engineering

The integration of Lynx in a workflow process involves a process reengineering. This means that the processes by which the organization, the PRRD 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 PRRD's 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 receptionist and the analyst. Conversely, if the documents were received in an electronic format, the complete process could be automated beginning at the notary public offices where the documents originate. This would make all the required information available in electronic form from the start, facilitating a more agile business processes and avoiding tedious and error prone manual data entry.

5.3 XML Workflow Document Definition

To develop an application on top of Lynx, we first define XML schemas for all document types used by the organization workflow. Our document schemas were based on several pre-designed Microsoft Word templates provided to us by the PRRD (see Figure 5.2).

These templates have common information used among different types of documents. For example, many PRRD inscription minute documents have the following features that are common with over 160 different documents:

- **Header.** Property number, town, inscription number, description, obligations, and

CANCELACION TOTAL DE HIPOTECA A FAVOR DE PERSONA DETERMI (1).doc - Microsoft Word

File Edit View Insert Format Tools Table Window Help

Type a question for help

Normal + 14 pt Times New Roman 14 B I U

70%

Read

Estado Libre Asociado de Puerto Rico
Departamento de Justicia
Registro de la Propiedad

Sección: Nombre de Sección

Número de Finca: Inscripción:

CANCELACIÓN TOTAL DE HIPOTECA A FAVOR DE PERSONA DETERMINADA EN GARANTÍA DE PAGARÉ NO NEGOCIABLE

DESCRIPCIÓN: Describe conforme con la inscripción Num. Inscripción. **CARGAS Y GRAVÁMENES:** Afecta a las cargas que surgen del Registro. **TITULARES:** Inscrita esta finca a favor de , según surge de la inscripción Num. Inscripción. **TÍTULO QUE SE CANCELA:** Hipoteca en garantía de pagaré a favor de , por la suma de , según surge de la inscripción Num. Inscripción. **DERECHO QUE SE INSCRIBE:** Cancelación total de la hipoteca antes relacionada.

PETICIONARIO DE LA CANCELACIÓN:
(Circunstancia Personal c/p. (o , representado por , Circunstancia Personal c/p, con facultades acreditadas en este Registro.) (o con facultades acreditadas mediante documento que se relacionará.)

CARÁCTER QUE OSTENTA EL PETICIONARIO: Acreedor y tenedor del pagaré, quien solicita la cancelación de la hipoteca por haberse saldado la deuda en su totalidad. **FE DEL NOTARIO:** El notario da fe de la cancelación del pagaré.

DOCUMENTO PRESENTADO: Escritura Número Num. Escritura, otorgada en Municipio, el día de mes de año, ante el notario Nombre Notario, donde comparece el peticionario de la cancelación. (Relacionar el documento de facultades).

DATOS DE PRESENTACIÓN: Presentada el día de mes de año, a las Hora de la (mañana o tarde), al Asiento del Diario .

En Municipio, a día de mes de año.

DERECHOS: \$ Cantidad

PRESENTACIÓN Y CÓDIGO POLÍTICO: \$10.50

Registrador

Page 1 Sec 1 1/1 At 10.3" Ln 42 Col 8 REC TRK EXT OVR Spanish (Puer)

Figure 5.2: Microsoft Word Template of a PRRD Document

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 (see Figure 5.4).

The imported schema is then assigned a namespace that identifies the set of elements and attributes of the data types to be used in another document. Figure 5.3 shows the XML Schema for the Word template shown in Figure 5.2. In the current prototype, the *http://ece.uprm.edu/RegPropCommon* namespace is used. Accordingly, these schemas will be also imported into the BPEL process definition described in the following section.

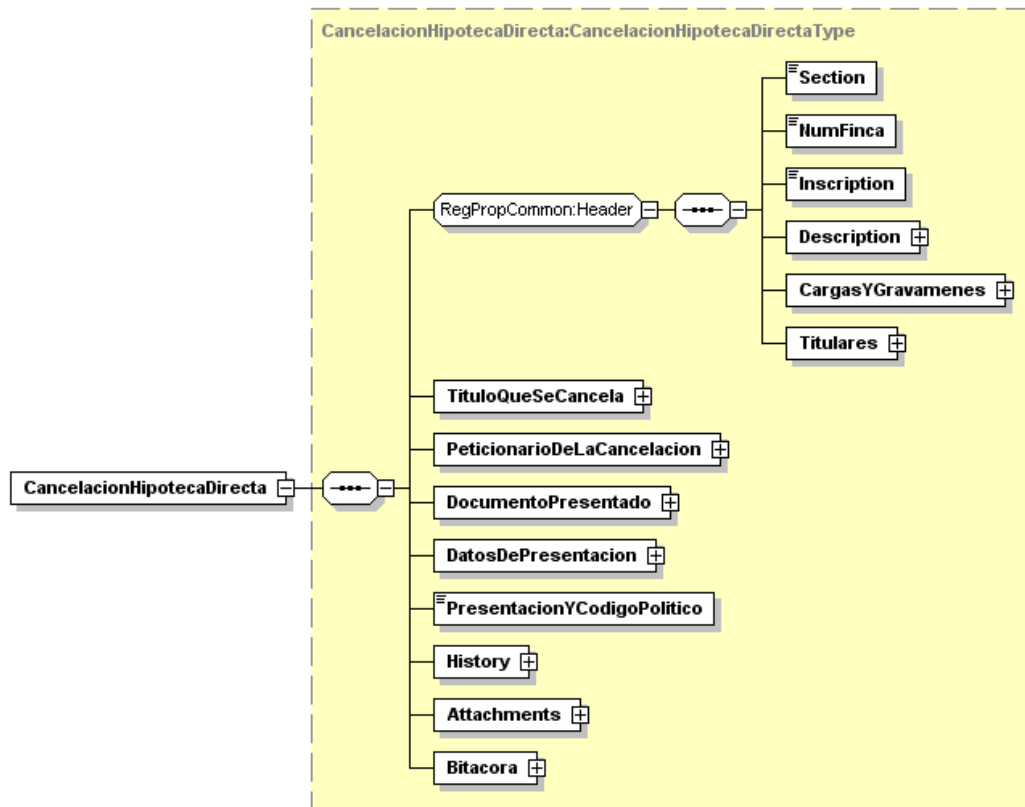


Figure 5.3: XML Schema Diagram for a Mortgage Cancellation Document

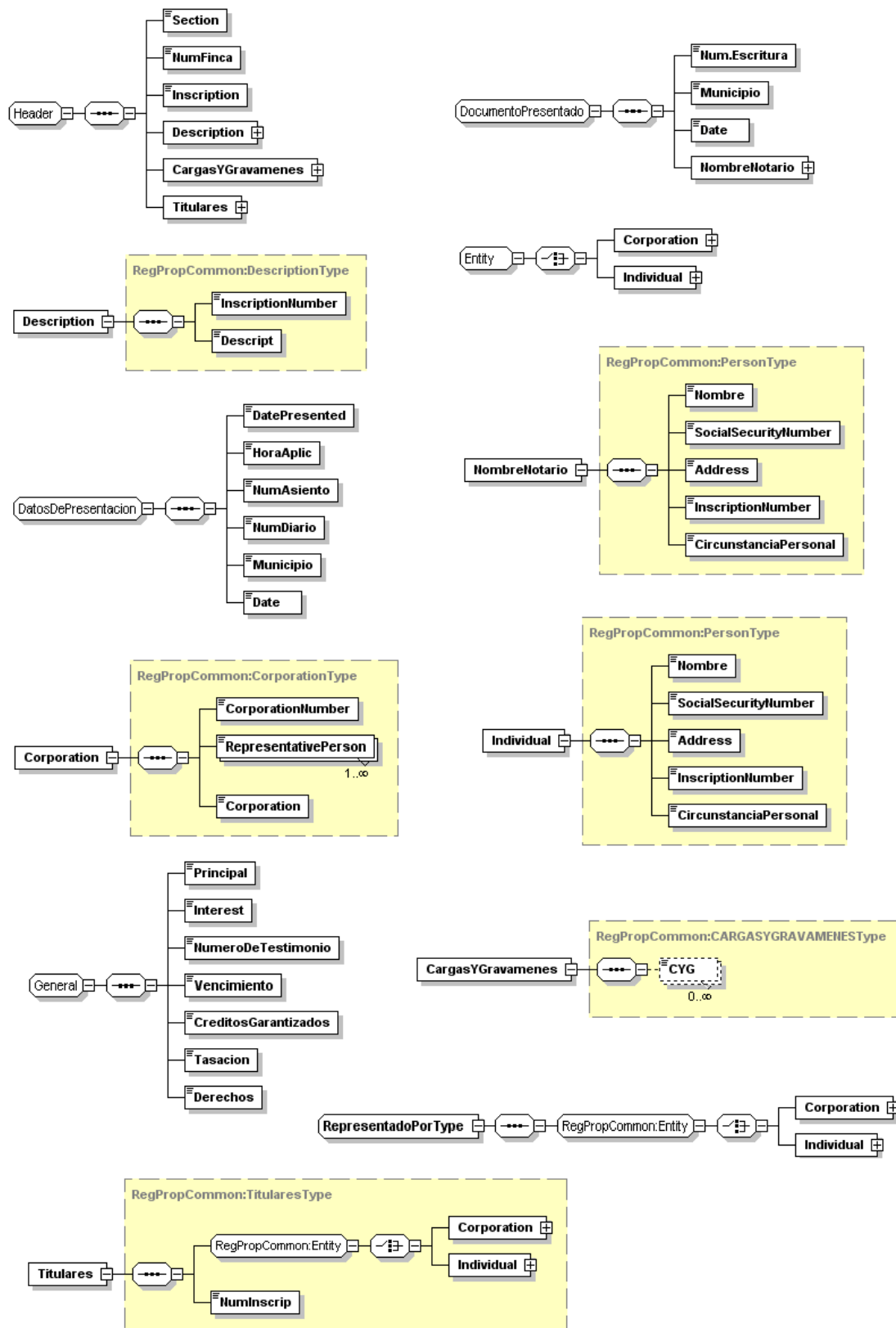


Figure 5.4: XML Schema Diagram Containing Common Types Used Among Documents

5.4 External Interface Description

The next step after defining schemas for all workflow documents is the description of the business process external interface in WSDL. This is done recalling what interactions are going to take place, specifying the web service operations, and assigning different WSDL *part* names for each corresponding operation (see Figure 5.5).

```

. . .
<wsdl:message name="cancelacionHipoteca">
  <wsdl:part name="CancelacionHipotecaDirecta"
    type="CHD:CancelacionHipotecaDirectaType"/>
</wsdl:message>
<wsdl:message name="cancelacionHipotecaAnalyst">
  <wsdl:part name="CancelacionHipotecaDirectaAnalyst"
    type="CHD:CancelacionHipotecaDirectaType"/>
</wsdl:message>
<wsdl:message name="cancelacionHipotecaRegistrador">
  <wsdl:part name="CancelacionHipotecaDirectaRegistrador"
    type="CHD:CancelacionHipotecaDirectaType"/>
</wsdl:message>
. . .

```

Figure 5.5: WSDL Message Definition for the PRRD Process

The operations exposed by the process are then defined in the WSDL as shown in Figure 5.6. In this application, *setDocument* is the operation that receives the Mortgage Cancellation document message first generated by the incoming email gateway after the notary public uses an XForms to submit the document. The other operations, *analyst-Completed* and *registradorCompleted*, receive the messages corresponding to annotated and revised versions of the document after the interaction with human Analysts and Registrar, respectively. Although the names of the *input* messages are different for each operation, what distinguishes each operation is the message's WSDL *part name* as shown previously in Figure 5.5 and explained in the previous Chapter 4.

```

. . .
<wsdl:portType name="RegistroPT">
  <wsdl:operation name="setDocument">
    <wsdl:input message="tns:cancelacionHipoteca"/>
    <wsdl:output message="tns:acknowledgement"/>
  </wsdl:operation>

  <wsdl:operation name="analystCompleted">
    <wsdl:input message="tns:cancelacionHipotecaAnalyst"/>
  </wsdl:operation>

  <wsdl:operation name="registradorCompleted">
    <wsdl:input message="tns:cancelacionHipotecaRegistrador"/>
  </wsdl:operation>
  . . .
</wsdl:portType>
. . .

```

Figure 5.6: WSDL Operation Definitions for the PRRD Process

BPEL introduces two relevant extensions to the WSDL standard: property alias definitions and partner link definitions. The correlation information is also defined in the same WSDL interface as the messages and operations. Figure 5.7 depicts the definition of a new property named *ID*, of type *string*. Also, a property alias definition is defined for each message that must be uniquely identified by the corresponding property. For this application every message is identified as pertaining to a specific PRRD document by the *ID* property that points to the PRRD document's *Inscription* number. This is specified in the *query* attribute by using an XPath expression to the Inscription location in the XML messages.

Partner link types contain the roles which define the responsibilities of each side of the conversation in a BPEL process. Figure 5.8 shows an excerpt of two of the partner links defined for the PRRD process. The *RegistroPLT* partner link type refers to the PRRD process itself, while the *Email* partner link type refers to the Outgoing Email Web Service. Each one is assigned a specific role, and the Web service's port type.

```

. . .
<wsbp:property name="ID" type="xsd:string"/>

<wsbp:propertyAlias propertyName="tns:ID"
  messageType="tns:cancelacionHipoteca"
  part="CancelacionHipotecaDirecta"
  query="/CancelacionHipotecaDirecta/Inscription" />

<wsbp:propertyAlias propertyName="tns:ID"
  messageType="tns:cancelacionHipotecaAnalyst"
  part="CancelacionHipotecaDirectaAnalyst"
  query="/CancelacionHipotecaDirectaAnalyst/Inscription" />

<wsbp:propertyAlias propertyName="tns:ID"
  messageType="tns:cancelacionHipotecaRegistrador"
  part="CancelacionHipotecaDirectaRegistrador"
  query="/CancelacionHipotecaDirectaRegistrador/Inscription" />
. . .

```

Figure 5.7: WSDL Correlation Definitions for the PRRD Process

```

. . .
<plnk:partnerLinkType name="RegistroPLT">
  <plnk:role name="service">
    <plnk:portType name="tns:RegistroPT"/>
  </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="Email">
  <plnk:role name="email">
    <plnk:portType name="eml:SendMail"/>
  </plnk:role>
</plnk:partnerLinkType>
. . .

```

Figure 5.8: WSDL PartnerLinkType Definitions for the Lawsuit Process

5.5 XForms Development

The next step consists of the design and development of the XForms used to display workflow documents and interact with human partners. The XForm can be designed using a GUI tool such as XFormation [47] or the IBM Alphaworks Visual XForms Designer [41]. A corresponding class that implements the *XFormCreator* Java interface is created for each XForm view for each document. This class dynamically generates the desired form and returns it as a String. The returned XForms can be used by both the Outgoing Email Web Service and the Web-based Interface. The *setupXform* method that is implemented in the class basically gets the instance data received and combines it with the XForms model and the XForms user interface desired, and sets the correct callback information and address of the email server to be used for submission. Figure 5.9 shows the general layout of a class implementing the *XFormsCreator* interface. An XForms for the Mortgage Cancellation document is shown in Figure 5.10.

```
public class AnalystXForm implements XFormsCreator {

    public AnalystXForm(){
        // Get SMTP host information from configuration file...
    }

    public String XFormsCreator {
        public abstract String setupXform( Document instance,
                                           String to,
                                           String from,
                                           String subject,
                                           String callback ){
            // 1. Use callback as instance data's root element name
            // 2. Insert instance data into XForm model
            // 3. If necessary, generate other temporary instance data
            // 4. Customize Submission element with to, from and SMTP info.
            // 5. Return XForms in a String
        }
    }
}
```

Figure 5.9: AnalystXForm Java Class Implementing XFormsCreator Interface

This step of the development process requires some non-trivial Java code to be written. Such code should do the steps outlined inside Figure 5.9. We leave the research of ways to further avoid Java programming as future work.

CancelacionHipotecaDirecta - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8080/chiba-web-2.0.0rc1/XFormsServlet?form=/forms/CancelacionHipoteca

**Estado Libre Asociado de Puerto Rico
Departamento de Justicia
Registro de la Propiedad**

CANCELACIÓN TOTAL DE HIPOTECA A FAVOR DE PERSONA DETERMINADA (HIPOTECA DIRECTA)

Verificado por: * Ivan Velez

Fecha: * 10.02.2006

Seccion: Mayaguez

Numero de Finca: 1234567890

Inscripcion: * 987654321

DESCRIPCIÓN:

Descrita conforme con la inscripción 12345

CARGAS Y GRAVÁMENES:

Afecta a las cargas que surgen del Registro.

TITULARES:

Inscrita esta finca a favor de Juan del Pueblo , según surge de la inscripción 12345

TÍTULO QUE SE CANCELA:

Hipoteca a favor de Juan del Pueblo , por la suma de 100,000 , según surge de la inscripción 12345

Done

Figure 5.10: XForms for the Mortgage Cancellation Document

5.6 Process Description

Having specified the external interface of the process and the XForms in the previous sections, the BPEL process is specified to orchestrate all the steps necessary to deal with the PRRD workflow. Figure 5.11 depicts the general structure of the BPEL business process developed for the PRRD that will be explained in this section. We assume the reader has some basic knowledge about the BPEL Specification [11] to understand this section.

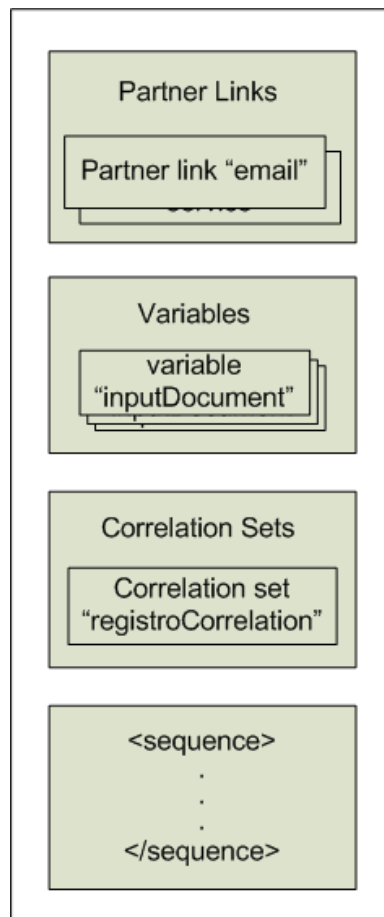


Figure 5.11: BPEL Components for the PRRD Process

First, the interactions between partners are expressed through partner links (see Figure 5.12). Then, the partners section groups the partner links and expresses the capabilities required from a business partner.

The interaction between partners in the process is based on exchanging messages. The notary public sends a message to begin the process, and messages are exchanged with the Analyst and Registrar. Variables are needed to store such messages. In addition, variables can also hold data that is only used for internal processing of the process. Figure 5.14 shows some of the variables used in the PRRD process. The *inputDocument* variable stores the original submitted mortgage cancelation document. Likewise, the *input-*

```

<partnerLinks>
  <partnerLink name="service" partnerLinkType="tns:RegistroPLT"
              myRole="service"/>
  . . .
  <partnerLink name="email" partnerLinkType="tns:Email"
              partnerRole="email"/>
</partnerLinks>

```

Figure 5.12: Partner Links Definition in the BPEL Process

```

<partners>
  <partner name="service">
    <partnerLink name="service"/>
  </partner>
  . . .
  <partner name="email">
    <partnerLink name="email"/>
  </partner>
</partners>

```

Figure 5.13: Partner Definition in the BPEL Process

DocumentEmailResponse and *inputDocumentEmailRegistradorResponse* variables store the annotated or revised document received from the Analyst or Registrar, respectively. The *emailInfo* variable stores the information necessary to communicate with the outgoing email Web service as described in Chapter 4.

Multiple interactions between partners might be involved in a specific instance of a PRRD process that happens over a long period of time. This requires the use of the BPEL correlation sets to be able of identifying a specific instance of a document case. The correlation sets are defined using the properties already defined in the WSDL (see Figure 5.15).

Finally, a single activity specifies the implementation of the BPEL process. This activity consists of nested activities that implement the business logic of the process. This activity can be a *sequence* that groups activities to be done sequentially, a *flow* of activities

```

<variables>
  <variable name="inputDocument" messageType="tns:cancelacionHipoteca"/>

  <variable name="bitacoraDocument"
    messageType="bit:StoreDocumentRequest"/>

  <variable name="inputDocumentEmailResponse"
    messageType="tns:cancelacionHipotecaAnalyst"/>

  <variable name="inputDocumentEmailRegistradorResponse"
    messageType="tns:cancelacionHipotecaRegistrador"/>

  <variable name="emailInfo" messageType="eml:sendEmailRequest"/>
  . . .
</variables>

```

Figure 5.14: Variables in the BPEL Process

```

<correlationSets>
  <correlationSet name="registroCorrelation" properties="tns:ID" />
</correlationSets>

```

Figure 5.15: Correlation set in the BPEL Process

that can be done in parallel, or a *scope* that groups other activities. BPEL supports a number of process definition statements to provide for activities such as assignment, loops, decision making, receipt of messages from a partner, and invocation of web service operation.

The process begins with the creation of a new instance of a particular Registry of Deeds case. The creation of a new instance always happens implicitly on the receipt of a message. Thus, the process has to start with either a *receive* or *pick* activity. For example, the submission of a Mortgage Cancellation document by the notary public results in the creation of a process instance to handle this case.

For the PRRD process, a *pick* is used to be able to receive different types of messages. The different messages correspond to the different types of documents handled by the PRRD. Figure 5.16 shows two picks capable of initiating a PRRD process because

it has the *createInstance* attribute set to "yes". The first one starts a process instance when it accepts a *CancelacionHipotecaDirecta* message since it is the type of the *inputDocument* variable.

```

<sequence>
  <pick createInstance="yes">
    <onMessage partnerLink="service"
      portType="tns:RegistroPT"
      operation="setDocument"
      variable="inputDocument">

      HERE GOES THE PROCESS ACTIVITIES FOR
      MORTGAGE CANCELTION DOCUMENTS

    </onMessage>
    <onMessage partnerLink="service"
      portType="tns:RegistroPT"
      operation="setViviendaDocument"
      variable="inputViviendaDocument">
      . . .
    </onMessage>
  </pick>
</sequence>

```

Figure 5.16: PRRD BPEL Process main activity

The rest of the process description essentially consists of a sequence containing variable assignments, invocations of the outgoing email web service, receipt of the results of human interaction, and decisions made based on the data. The BPEL process was designed and developed so that it invokes the Lynx outgoing Web service by encapsulating the information necessary to communicate with a human partner inside a message that follows the *EmailInfo* schema as depicted in Figure 4.2 on Section 5.6, and taking into consideration that the *callback* element of the *EmailInfo* message must match the WSDL *part name* of the message received in the operation that waits for a response.

We show in Figure 5.17 an excerpt of the BPEL specification preparing and per-

```

<assign name="AnalystAssign">
  <copy>
    <from expression="'analysts'" />
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/to" />
  </copy>
  . . .
  <copy>
    <from expression="'CancelacionHipotecaDirectaAnalyst'" />
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/callback" />
  </copy>
  <copy>
    <from expression="'edu.uprm.ece.egov.registro.
      xforms.CancelacionHipotecaDirectaAnalystXForm'" />
    <to variable="emailTempInfo" part="EmailInfo"
      query="/EmailInfo/class" />
  </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 5.17: Invoking the Outgoing Email Web Service to Interact with an Analyst

forming 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 engine's Web service operation.

Figure 5.18 shows the BPEL operation that waits for a response of the specific type specified in the callback that will get invoked automatically. Note that the message part

```

    <pick createInstance="no">
      <onMessage partnerLink="service"
        portType="tns:RegistroPT"
        operation="analystCompleted"
        variable="inputDocumentEmailResponse">
        <correlations>
          <correlation set="registroCorrelation" initiate="no"/>
        </correlations>
        <empty/>
      </onMessage>
    . . .

```

Figure 5.18: Receiving the Document that Results from the Interaction with an Analyst

name of the variable *inputDocumentEmailResponse* in the OnMessage activity was defined as *CancelacionHipotecaDirectaAnalyst* in the process WSDL. *CancelacionHipotecaDirectaAnalyst* matches the */EmailInfo/callback* element used when the Outgoing Email Web Service was invoked. In this way the process resumes at the right point and continues along the process specification. Afterwards, it prepares the next human partner interaction, and sends a request via the outgoing email web service as shown in Figure 5.17.

Also, in addition to interaction through email, the BPEL process invokes a web service that stores and updates the documents in a Native XML database needed to enable interaction with the workflow documents through the web-based interface. This is the subject of the next section.

5.7 Web-based Interface to the Workflow Application

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 deployment of a web-service based workflow application. 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. The Entity-Relationship (E-R) Diagram used for Lynx is shown in Figure 5.19.

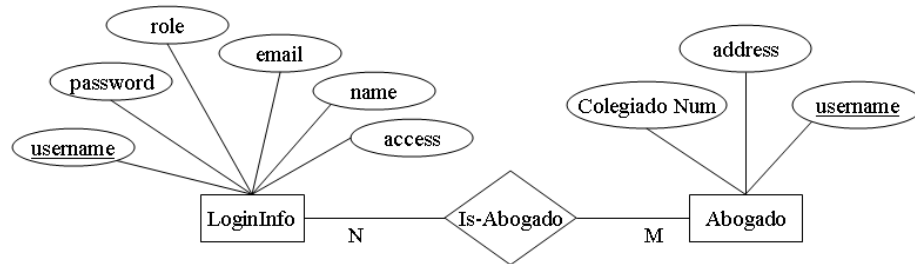


Figure 5.19: E-R Diagram for Lynx Database

Properly authenticated users, such as supervisors, registrars, notary publics or analysts, 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 *BitacoraType* XML schema type imported into every documents schema to store Log information is showed in Figure 5.20. The *BitacoraEntryType* 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.

The Log XML schema includes the same information as the *EmailInfo* schema type presented previously in Figure 4.2. An element named *nextAuth* is needed to specify which user or role has access to a view of a document through the web-based interface. This view can be either a read-only view of the document, or the XForms view generated by the Java class specified in the *class* element. This allows presentation of the correct XForms to the user logged into the system and provides the information necessary to submit the validated

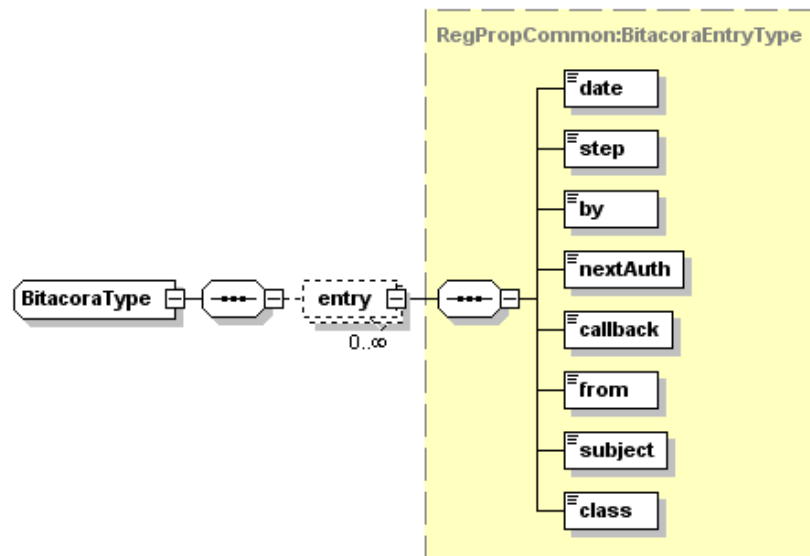


Figure 5.20: Log XML Schema

instance data back to the BPEL engine through the Incoming Email Gateway.

Figure 5.21 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. The web-based interface was developed using Java Server Pages (JSP). Three JavaBean classes (*DocumentTypes*, *XMLDBQuery* and *XMLDBRowInfo*) were also designed and developed to be able to get configuration information, and connect and get information from the Native XML database. These JavaBeans were then used from within the JSP, and with the use of the Java Standard Tag Library (JSTL) the web interface was developed without the need for Java programming, except for the initial work required to develop a reusable JavaBean capable of representing a Native XML database connection. Figure 5.22 shows the use of these JavaBeans. The *DocumentTypes* bean contains the different document types and correlation queries specified in the XML configuration file, *XMLDBQuery* connect to the XML database and gets information using the correlation and XPath query passed as parameters, and *XMLDBRowInfo* contains the information about each document, useful for constructing the web interface.

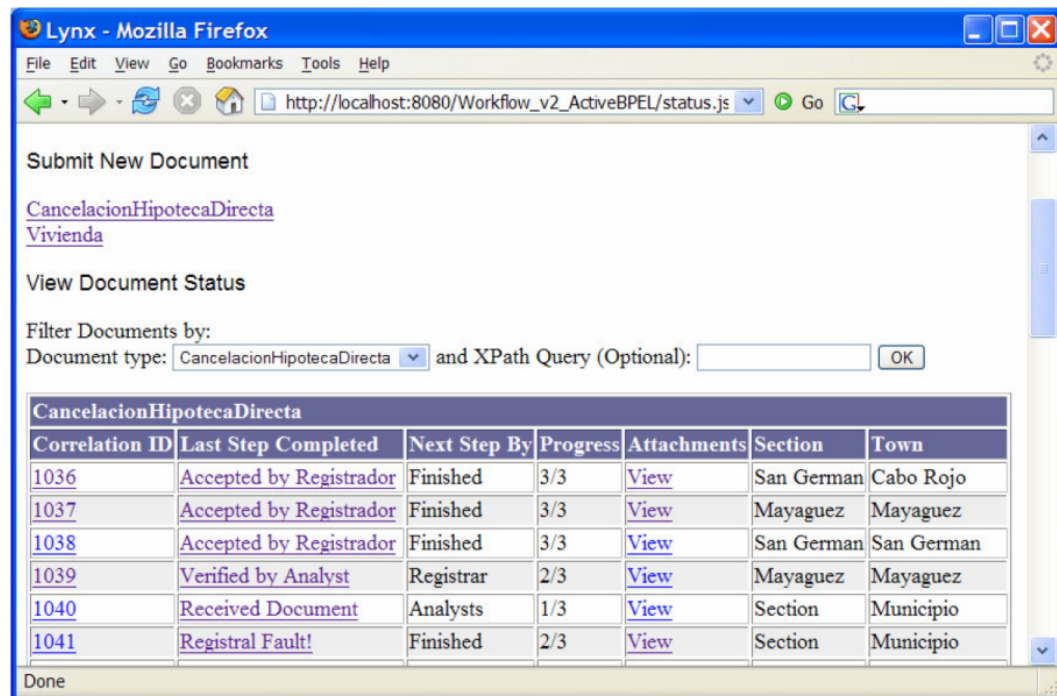


Figure 5.21: Document Status Page

The documents displayed through the Web-based interface are rendered as XForms using the Chiba XForms player. We pass as a parameter to Chiba the XForms that are generated by the Java classes implemented earlier. 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.

The status web pages were done using JSP instead of XForms since we directed our work towards using XForms only to display the documents requiring interaction. However, XForms may also be used to render the status page of the Web-based interface. The generation of an XML instance containing the status information for each document pending processing would be needed. This instance data can then be rendered as an *output repeat* element table in an XForms page.

```

<jsp:useBean id="dt"
              class="edu.uprm.ece.egov.DocumentTypes"
              scope="page"/>
<jsp:useBean id="xq"
              class="edu.uprm.ece.egov.XMLDBQuery"
              scope="page"/>

<c:forEach items='${dt.documenttypes}' var='item'>
    . . .
</c:forEach>

<jsp:setProperty name="xq" property="correlation"
                  value="${item.doccorr}"/>
<jsp:setProperty name="xq"
                  property="xpath"
                  value="${xpath}"/>
<c:forEach items='${xq.rows}' var='results'>
    . . .
</c:forEach>

```

Figure 5.22: Using JavaBeans with JSTL in the JSP web interface

Figure 5.23 and Figure 5.24 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, or any other relevant complementary documents.

ActiveBPEL allows the monitoring of the deployed BPEL processes through 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 5.25.

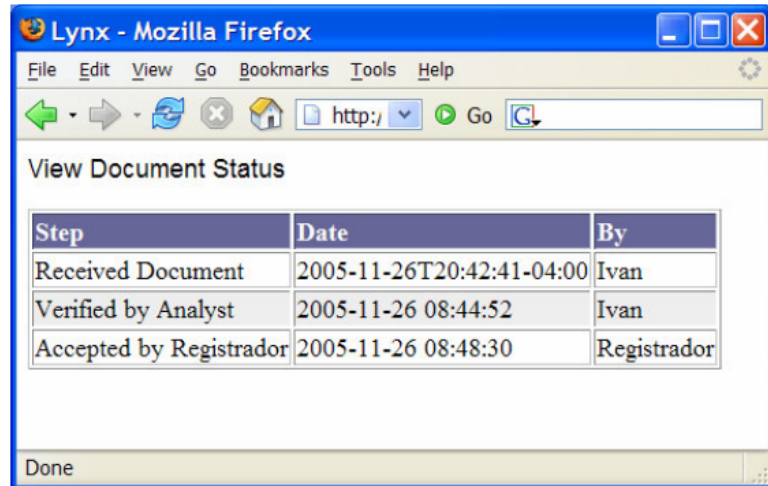


Figure 5.23: Document Status Log View

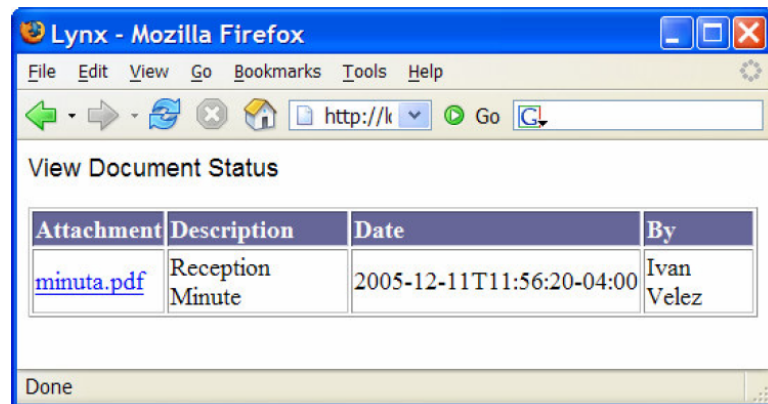


Figure 5.24: Document Status Attachments view

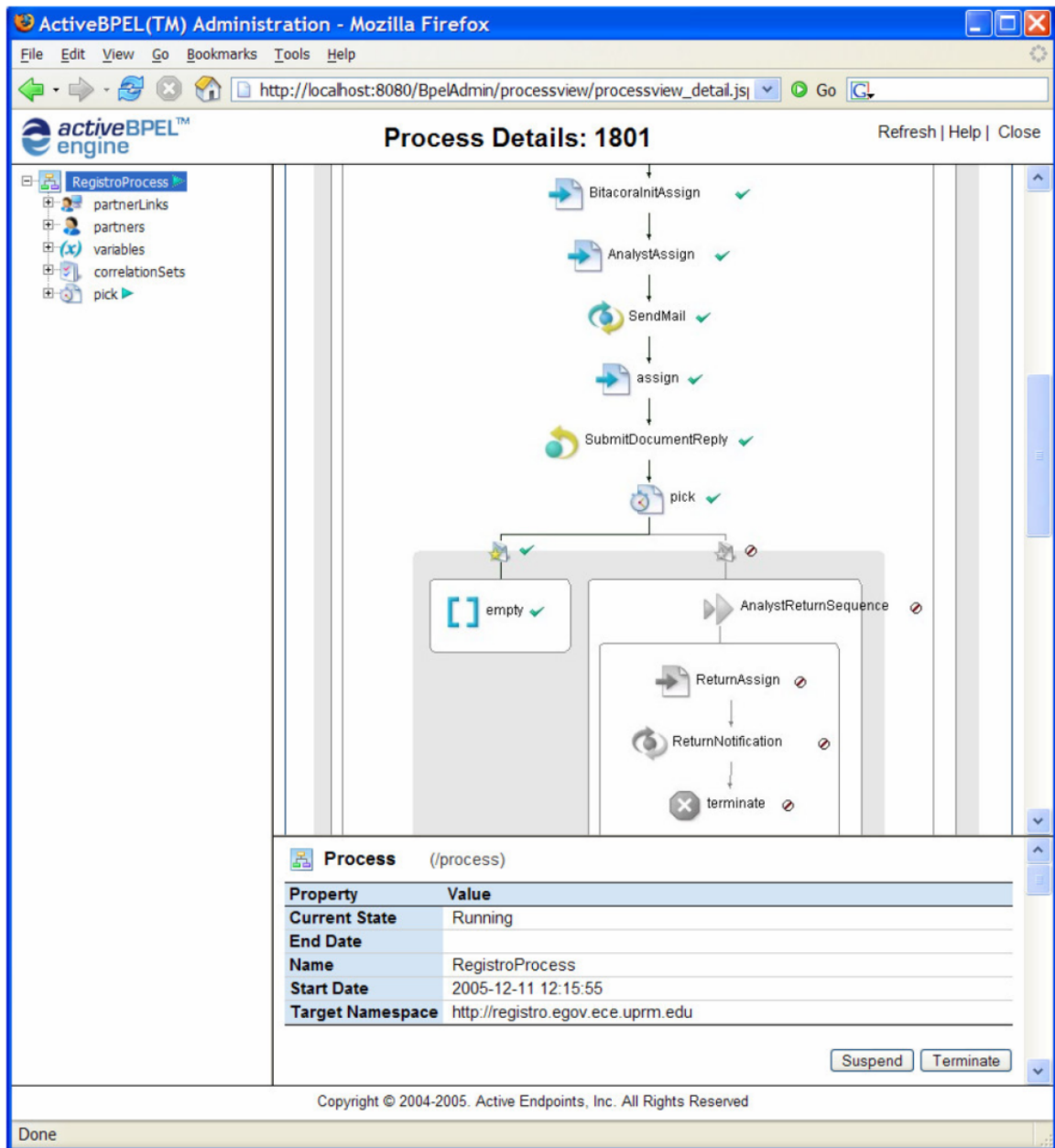


Figure 5.25: Workflow Process Detail Graph

5.8 XML Storage Subsystem

A persistent copy of all workflow documents is maintained in both a database that is used by ActiveBPEL for process persistence, and in an eXist [12] Native XML database that is used by Lynx to store the current version of each document. This is necessary since the ActiveBPEL engine requires a relational database for the persistence of the workflow process state information, while the Native XML database stores the current version of each document to be able to present them through the web-based interface. 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 [45] and XQuery [48] as the query languages and the XUpdate [49] language to insert and update XML documents. This is particularly helpful since some complex XML document structures may be very difficult to map to an intuitive relational database.

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 Lynxs 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 5.26). 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.

```
<config>
  <type>
    <docType>CancelacionHipotecaDirecta</docType>
    <docCorrelation>header/inscription</docCorrelation>
    <relevant>
      <Section xpath='header/section' />
      <Town xpath='datosDePresentacion/municipio' />
    </relevant>
  </type>
  <type>
    <docType>Vivienda</docType>
    <docCorrelation>header/inscription</docCorrelation>
    <relevant>
      <Notary xpath='DocumentoPresentado/NombreNotario' />
    </relevant>
  </type>
</config>
```

Figure 5.26: Document Type Configuration

5.9 Developing a New Application Using 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:

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, roles and emails. 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.

The next chapter will show the deployment of another Web-service based workflow application following the Lynx architecture using the approach described in this section.

CHAPTER 6

A Workflow-based Application to Track the Flow of Legal Cases in the Puerto Rico Judiciary

6.1 Introduction

As part of this research, Lynx was used to develop an application for another government environment. While the development of the PRRD application described in the previous chapter served to illustrate the application development process and concepts required to facilitate the development of any application, this second application serves as an experiment to measure the complexity of deploying a web-based workflow application from scratch using Lynx, following the steps outlined in the previous Chapter 5. This is usable as a roadmap for the development of different Lynx applications. Specifically, we considered an application for the process of lawsuits between two private parts for the Puerto Rico Courts shown in Figure 6.1.

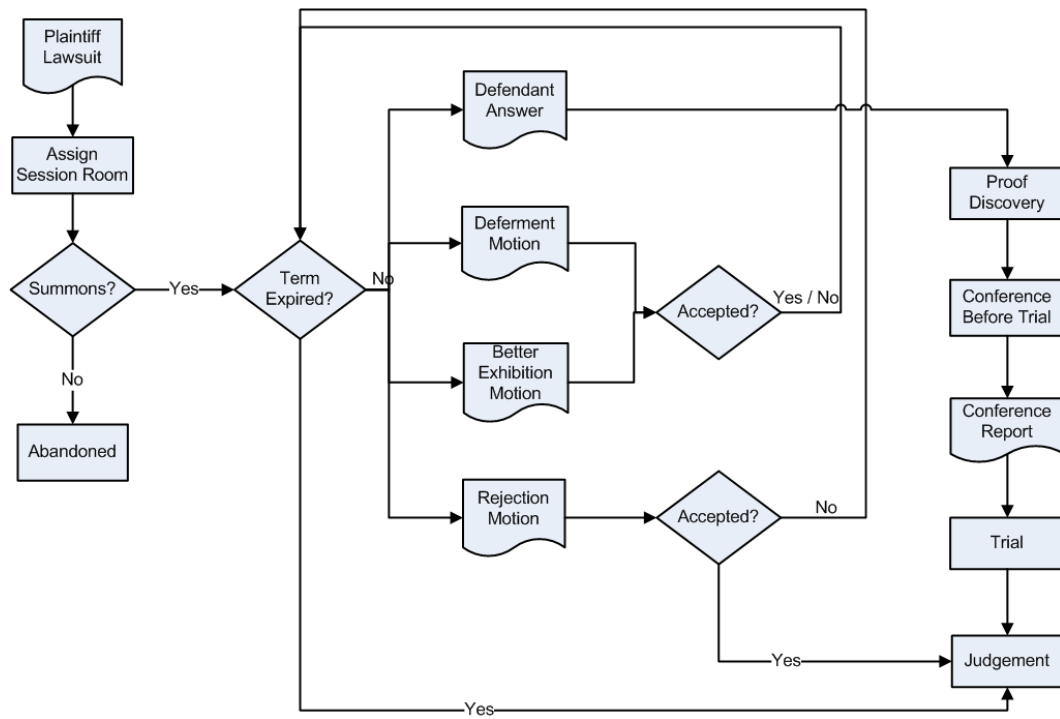


Figure 6.1: Flowchart of a lawsuit case

The typical flow of lawsuit documents involves the plaintiff lawyer first submitting a lawsuit document. The secretary then assigns a session room and judge. Next, the defendant part must be summoned. If the plaintiff does not summons, the case is considered abandoned. Else, the defendant part should answer the lawsuit or present some type of motion within a predetermined time period. After the defendant response is received, the proof discovery starts. Following the discovery, a conference between lawyers is conducted. Later, the trial is carried on, and finally judgement is advised and the sentence assigned.

6.2 Supporting Software Substrate

Deploying a Lynx-based workflow application involves the integration of several different supporting software components. This section describes the deployment configuration details required to deploy an application using Lynx.

1. **Java SDK 1.4.2.** In the first place, the Java Software Development Kit version 1.4.2 or greater must be available.
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.
3. **Chiba-web-1.0.0.** Chiba serves as the XForms processor for Lynx. Chiba is installed as a Web application in Tomcat. The Chiba-web-1.0.0 distribution was patched by modifying the source code of the *org.chiba.adapter.servlet.ChibaServlet* class and re-compiling it to allow the loading of XForms documents from locations other than Chiba's own application context inside Tomcat. As explained in Section 5.6, the Chiba XForms processor implements XForms by rendering them as standard HTML through a servlet. 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, this XForms player component will be replaced with the email or web browser client, or a native plug-in that supports the whole standard.
4. **ActiveBPEL 1.1.6 server.** The ActiveBPEL engine is an Open Source implementation of a BPEL engine, written in Java. The ActiveBPEL engine requires an installed and properly configured servlet container such as Tomcat. The installation scripts that come with the ActiveBPEL engine use the environment variable *CATALINA_HOME*, which defines the top-level Tomcat directory. ActiveBPEL is also configured to run as a persistent engine capable of storing all process state information in a database. This procedure, which involves creating several database tables and configuring a Tomcat JNDI data source, is described in detail in the ActiveBPEL documentation. ActiveBPEL's embedded Axis server is used to host the Web services invoked.

5. **MySQL 4.1 Database.** A relational database, such as MySQL, should be available for use by the Active BPEL engine, and also to store the authentication information for the Lynx web-based interface. ActiveBPEL also supports Oracle, DB2 and SQL Server databases to support process persistence.
6. **eXist 1.0rc1-20060710 XML Database server.** Exist XML DB was installed as a Web application inside Tomcat. A collection named *bitacora* must be created to store Lynx documents. Also, an index of the correlation information specific to the application has to be defined for the collection to facilitate efficient querying of the database.
7. **Lynx Web Application.** The Lynx web-based interface is a customized version of the web application developed for the PRRD workflow application described in Chapter 5. It is deployed as a Web application in Tomcat. Its customization is described in Section 6.7. Also, a JNDI data source needs to be configured for the Lynx web application in Tomcat.
8. **Clients.** For the Lynx prototype developed for this research, 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 capable 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. Therefore, Tomcat 5.0 or later must also be running locally on the client to support the XForms rendering. The custom Lynx attachment MIME type must be associated with the XForms Processor invoker. The invoker can be a simple script that calls a Web browser and passes the Chiba URL with XForms as parameter (for example, `firefox.exe http://localhost:8888/chiba-1.0.0/XFormsServlet?form=file:///1`).

6.3 Document Definition

After having configured the necessary components described in the previous section, the XML Schema definitions need to be defined for each document that the application must manage. For this specific application the following legal documents were taken into consideration: Lawsuit, Summons, Answer, Deferment Motion, Rejection Motion, Better Exhibition Motion, Preliminary Conference Report, and Judgement.

Common schema type information used among different types of documents is placed on a separate schema. Most documents, including the lawsuit, motions and answers, have the following features that are common:

- **Header.** City, case number, plaintiff, defendant, session room and topic.
- **Person.** Personal information about either plaintiff or defendant.
- **Lawyer.** Name, ID, address, telephone number, fax, email.
- **Expositions.** The text of the petitions, allegations, negations, affirmations or judgement.
- **Log and Attachments.**

These definitions common to all the documents are placed in an XML Schema (depicted in Figure 6.2) file that can be imported into any of the specialized XML Schemas that implement the definition of each required document. The imported schema was assigned a namespace *http://ece.uprm.edu/TribunalesCommon* that identifies the set of elements and attributes of the data types to be used in another document. Figure 6.3 shows the XML Schema of a lawsuit document using the imported *TribunalesCommon* schema elements. Figures 6.4, 6.5 and 6.6 show the XML Schemas for the Answer, Summons and Motion documents, respectively. Accordingly, these schemas must be also imported into the BPEL process definition.

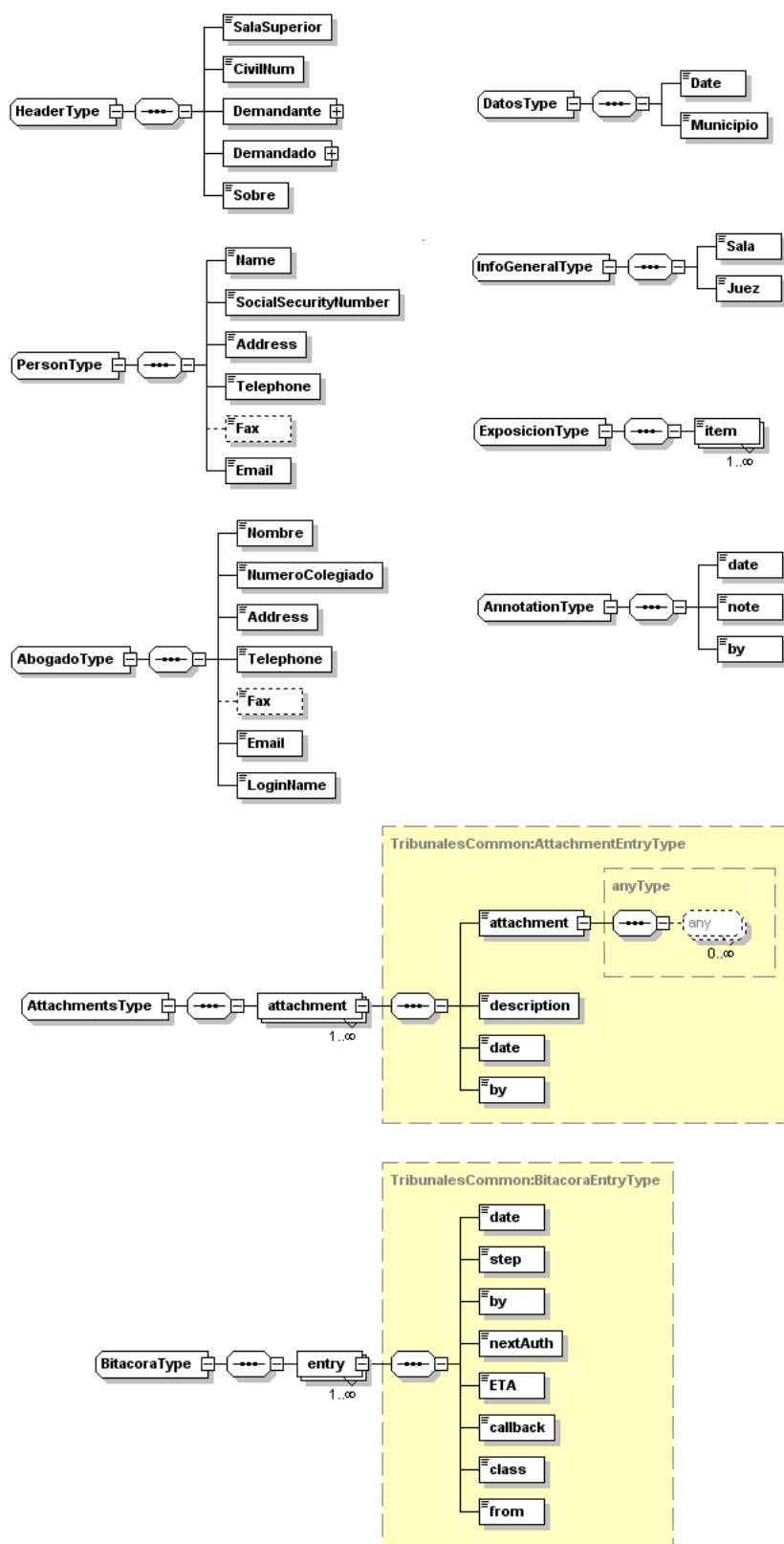


Figure 6.2: XML Schema Diagram for Schema Types Common among Documents

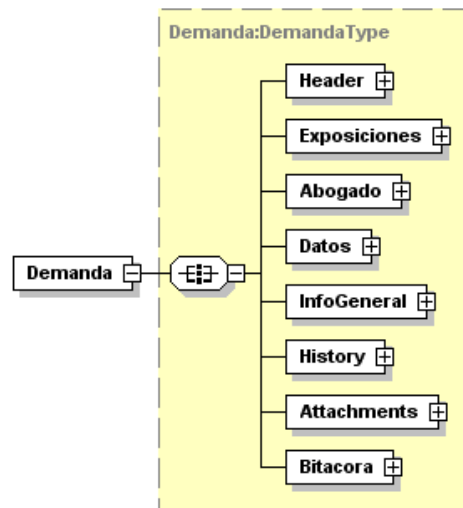


Figure 6.3: XML Schema Diagram for a Lawsuit Document

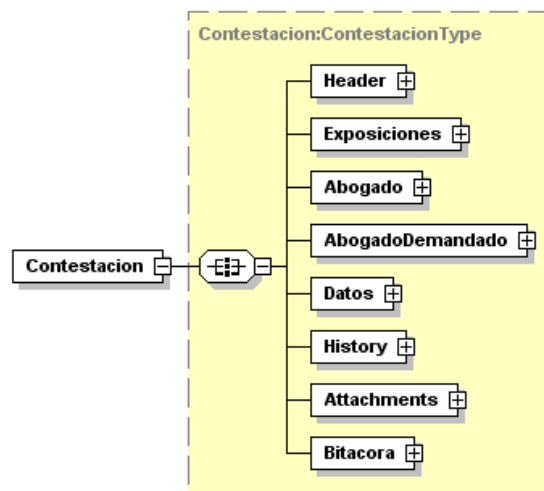


Figure 6.4: XML Schema Diagram for an Answer Document

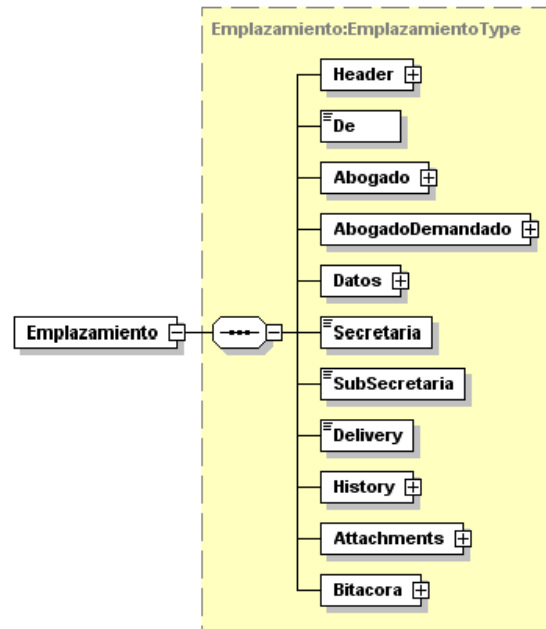


Figure 6.5: XML Schema Diagram for a Summons Document

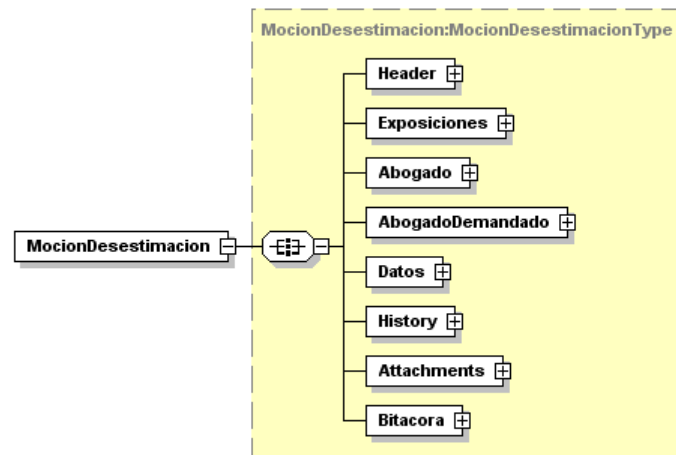


Figure 6.6: XML Schema Diagram for a Motion Document

6.4 External Interface Description

The WSDL interface of the process is then defined after defining the document schemas. This is done recalling what interactions are going to take place, assigning different WSDL part names for each corresponding operation, and specifying the web service operations. Also, the partner links and properties are defined. Figure 6.7 shows an excerpt of the WSDL definitions used for this new application.

```

<wsdl:message name="demanda">
  <wsdl:part name="Demanda"
    type="demanda:DemandaType"/>
</wsdl:message>
. . .
<wsdl:message name="contestacion">
  <wsdl:part name="Contestacion"
    type="contestacion:ContestacionType"/>
</wsdl:message>
<wsdl:message name="sentencia">
  <wsdl:part name="Sentencia"
    type="sentencia:SentenciaType"/>
</wsdl:message>

<wsdl:portType name="EGovTribunalesPT">
  <wsdl:operation name="setDemandaCobroDinero">
    <wsdl:input message="tns:demanda"/>
  </wsdl:operation>
  <wsdl:operation name="setMocion">
    <wsdl:input message="tns:mocion"/>
  </wsdl:operation>
  <wsdl:operation name="setContestacion">
    <wsdl:input message="tns:contestacion"/>
  </wsdl:operation>
  . . .
</wsdl:portType>

<wsbp:property name="ID" type="xsd:string"/>

<plnk:partnerLinkType name="EGovTribunalesPLT">
  <plnk:role name="service">
    <plnk:portType name="tns:EGovTribunalesPT"/>
  </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="Email">
  <plnk:role name="email">
    <plnk:portType name="eml:SendMail"/>
  </plnk:role>
</plnk:partnerLinkType>
. . .
<wsbp:propertyAlias messageType="tns:demanda" part="Demanda"
  propertyName="tns:ID" query="/Demanda/Header/CivilNum"/>
. . .

```

Figure 6.7: WSDL Message Definition for the Lawsuit Process

6.5 XForms Development

After all the software is installed and configured, and the document schemas and process interfaces are defined, all the required XForms are designed and developed. A corresponding class that implements the *XFormsCreator* Java interface must be created for each XForm view for each document following the procedure described in Section 5.5.

The most relevant XForms developed for the Judicial application are shown here. These correspond to the XML Schemas for the different documents presented earlier in Section 6.3. Figure 6.8 shows the XForms for the Lawsuit documents following the Lawsuit schema illustrated in Figure 6.3. Figure 6.9 shows the Summons for the Lawsuit documents. Figure 6.10 shows the XForms for the Answer documents corresponding to schema shown in Figure 6.4. Figure 6.11 shows the XForms for a Motion document to show documents conforming to schema depicted in Figure 6.6. Figure 6.12 shows the XForms for a Conference Report document.

Finally, the XForms shown in Figure 6.13 includes validation needed to accept or reject a submitted Lawsuit document due to errors or missing information that the Court Secretary must verify. The same concept is used for the validation of the other document types. These XForms basically display a read-only version of the document and give the user the option of confirming the document data by pressing a button that will send a confirmation message via email back to the process.

Demanda - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8888/chiba-1.0.0/XFormsServlet Go

EN EL TRIBUNAL GENERAL DE JUSTICIA DE PUERTO RICO
TRIBUNAL DE PRIMERA INSTANCIA
SALA SUPERIOR DE MAYAGUEZ

Demandante:
v.
Demandado:

Civil Núm.:
Sobre:

DEMANDA

AL HONORABLE TRIBUNAL

COMPARECE la parte demandante, representada por su abogado que suscribe y muy respetuosamente, ante este Honorable Tribunal, EXPONE, ALEGA y SOLICITA:

1. El número de Seguro Social de la parte demandante es .

Sus direcciones física y postal son, respectivamente, las siguientes: .

El teléfono es el .

Exposicion:

Anadir Exposicion

En , Puerto Rico, hoy 2006-09-18T07:16:42-04:00

Abogado:
Colegiado #: 007
Direccion:
Email:

Attachments:

Description:
File: Browse...
Add Attachment

Someter Documento

Done

Figure 6.8: XForms for a Lawsuit Document

Emplazamiento OK? - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://192.168.3.251:8888/chiba-1.0.0

**EN EL TRIBUNAL GENERAL DE JUSTICIA DE PUERTO RICO
TRIBUNAL DE PRIMERA INSTANCIA
SALA SUPERIOR DE MAYAGUEZ**

Demandante: Luis
v.
Demandado: Rene

Civil Núm.: CD-20060903001
Sobre: COBRO DE DINERO

EMPLAZAMIENTO

De:

A: Rene,
o sea, la parte demandada antes mencionada
Por la presente se le emplaza y requiere para que envíe al:

abogado del demandante, cuya dirección es la aquí indicada, copia de su contestación a la Demanda, la cual se le está haciendo entrega en este acto, dentro de los 20 días de haber sido diligenciado, o entregado este emplazamiento, (si el emplazamiento se hiciera en la Isla de Puerto Rico), excluyéndose el día del diligenciamiento o entrega, apercibiéndose que en caso de no hacerlo así podrá dictarse Sentencia en Rebeldía en contra suya, concediendo el remedio solicitado en la Demanda.

EXTENDIDO BAJO MI FIRMA y el Sello del Tribunal, hoy

Secretaria

Sub-Secretaria

Emplazar usando metodo:

Done

Figure 6.9: XForms for a Summons Document

Contestacion - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://192.168.3.251:8888/chiba-1.0.0/XFormsSer Go

Demandante: Luis
 v.
 Demandado: Rene

Civil Núm.: CD-20060903001
 Sobre: COBRO DE DINERO

CONTESTACIÓN A DEMANDA

AL HONORABLE TRIBUNAL

COMPARECE, Rene, representada por su abogado que suscribe y muy respetuosamente, ante este Honorable Tribunal, EXPONE, ALEGA y SOLICITA:

Exposicion:

Anadir Exposicion

POR TODO LO CUAL, se solicita respetuosamente de este Honorable Tribunal declare Sin Lugar la demanda, con cualquier otro pronunciamiento que en Derecho proceda.

CERTIFICO: Haber enviado copia fiel y exacta del presente escrito al abogado de la parte Demandante, a su dirección Mayaguez, PR 00680

En Mayaguez, Puerto Rico, hoy 2006-09-18T08:00:57-04:00

Abogado: Ivan Velez

Colegiado #: 123

Direccion: Mayaguez, PR

Email:

Attachments:

Description:

File: Browse...

Add Attachment

Guardar Borrador

Someter Documento

Done

Figure 6.10: XForms for an Answer Document

Mocion OK? - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://192.168.3.251:8888/chiba-1.0.0/XFormsServ Go

**EN EL TRIBUNAL GENERAL DE JUSTICIA DE PUERTO RICO
TRIBUNAL DE PRIMERA INSTANCIA
SALA SUPERIOR DE MAYAGUEZ**

Demandante: Luis
v.
Demandado: Rene

Civil Núm.: CD-20060903001
Sobre: COBRO DE DINERO

MOCIÓN

AL HONORABLE TRIBUNAL
COMPARECE la parte Demandada representada por su abogado que
suscribe y muy respetuosamente EXPONE y SOLICITA:

Exposicion:

Anadir Exposicion

CERTIFICO: Haber enviado copia fiel y exacta del presente escrito
al abogado de la parte Demandante, a su dirección Mayaguez, PR 00680
En Mayaguez, Puerto Rico, hoy 2007-01-01T00:00:00-04:00
Abogado:
Colegiado #: 123
Direccion:

Documentos explicativos:

Descripcion:

File: Browse...

Someter Documento

Done

Figure 6.11: XForms for a Motion Document

InformeConfPreliminar - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://192.168.3.251:8888/chiba-1.0.0/XFormsServ Go

EN EL TRIBUNAL GENERAL DE JUSTICIA DE PUERTO RICO
TRIBUNAL DE PRIMERA INSTANCIA
SALA SUPERIOR DE MAYAGUEZ

Demandante: Luis
v.
Demandado: Rene

Civil Núm.: CD-20060903001
Sobre: COBRO DE DINERO

INFORME DE CONFERENCIA PRELIMINAR ENTRE ABOGADOS

Informe:

En , Puerto Rico, hoy 2006-09-18T08:37:24-04:00
Abogado:Ivan Velez
Colegiado #: 007
Direccion:Mayaguez, PR 00680

Attachments:

Description:

File: Browse...

Add Attachment

Someter Documento

Done

Figure 6.12: XForms for a Preliminary Conference Report Document

EN EL TRIBUNAL GENERAL DE JUSTICIA DE PUERTO RICO
TRIBUNAL DE PRIMERA INSTANCIA
SALA SUPERIOR DE MAYAGUEZ

Demandante: Luis
v.
Demandado: Juan

Civil Núm.: CD-20060828001
Sobre: COBRO DE DINERO

CONTESTACIÓN A DEMANDA

AL HONORABLE TRIBUNAL
COMPARECE, Luis, representada por su abogado que suscribe y muy respetuosamente,
ante este Honorable Tribunal, EXPONE, ALEGA y SOLICITA:

☒ Exposicion: contestacion no acepta alegaciones

POR TODO LO CUAL, se solicita respetuosamente de este Honorable Tribunal
declare Sin Lugar la demanda, con cualquier otro pronunciamiento
que en Derecho proceda.

CERTIFICO: Haber enviado copia fiel y exacta del presente escrito
al abogado de la parte Demandante, a su dirección Mayaguez, PR
En Mayaguez, Puerto Rico, hoy 2006-08-28T09:32:43-04:00
Abogado: Juan del Pueblo
Colegiado #: 123
Direccion: Mayaguez, PR

Done

Figure 6.13: XForms for Validating a Lawsuit Document

6.6 Process Description

The BPEL process of a lawsuit case workflow between two private parts was developed following the same procedure as the one used to develop the sample process for the PRRD. However, the freely available ActiveBPEL Designer [7] was used to facilitate the construction of the BPEL process since it provides a highly visual environment for rapidly developing process definitions that are BPEL-compliant. The ActiveBPEL Designer contains all the features required to help quickly design, test, and deploy business processes including drag and drop activity creation, WSDL categorization, visual representation of all participant interactions, comprehensive static analysis (BPEL validation) and automatic task creation, expression builders, simulation of process execution, deployment and packaging to ActiveBPEL Engine, and remote debugging and analysis of processes executing in the ActiveBPEL engine.

The BPEL process was designed and developed so that it invokes the Lynx outgoing web service by encapsulating the information necessary to communicate with a human partner inside a message that follows the *EmailInfo* schema as depicted in Figure 4.2 on Chapter 4, and explained on Section 5.6, and taking into consideration that the *callback* element of the *EmailInfo* message must match the WSDL part name of the message received in the operation that waits for a response. Also, in addition to interaction through email, the BPEL process invokes a web service that stores and updates the documents in a native XML Database needed to enable interaction with the workflow documents through the web-based interface, discussed in the next section.

Finally, the BPEL process is deployed as a message-style service into the ActiveBPEL server using the free ActiveBPEL Designer. Deploying a BPEL process involves creating a deployment archive file (a JAR with an extension of ".bpr") and copying that to the *CATALINA_HOME/bpr* directory.

6.7 Web-based Interface

The web-based interface that was developed, using Java Server Pages, for the PRRD application presented on the previous Chapter 5 was reused for this new application. However, for this application, the JSP interface was customized to show all the documents, such as motions, answers, and reports, related to a specific lawsuit case. Figure 6.14 shows the status page demonstrating a lawsuit case in the main status page. After selecting a particular case through its *Civil Num.* link, the specific case information web page, shown in Figure 6.15, displays the status of all the documents related to the specific case, allowing to see the Log, the status of each document, and to either view the document, or validate it.

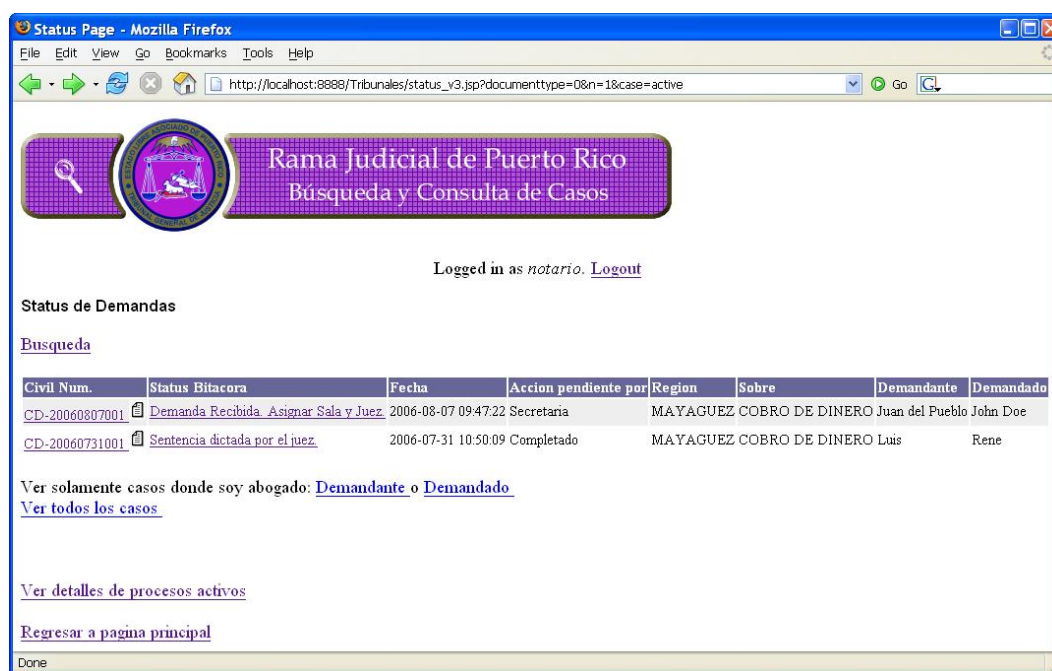



Figure 6.14: Lawsuits status web page

Status Page - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8888/Tribunales/status_v3b.jsp?type=BitacoraLog&corr=Header/CivilNum='CC'

 Rama Judicial de Puerto Rico
Búsqueda y Consulta de Casos

Logged in as abogado. [Logout](#)

[Regresar a lista de casos](#)

Informacion del caso

Civil Num.	CD-20060731001	Region	MAYAGUEZ	Demandante	Luis
Status	Sentencia dictada por el juez.	Materia	COBRO DE DINERO	Demandado	Rene
Accion pendiente por	Completado				

Documentos relacionados

Documento	Status	Fecha	Proximo	Anejos
Demanda	Demanda aceptada por Secretaria Ver.	2006-07-31 09:33:04	None	Ver 0
Emplazamiento	Emplazamiento enviado Ver.	2006-07-31 09:32:37	None	n/a
Contestacion	Contestacion Recibida Ver.	2006-07-31 09:37:49	None	n/a
Mocion	Completar y llenar documento nuevo de Mocion para caso CD-20060731001			
Informe ConfPreliminar	Informe confirmado. Empezar Juicio Ver.	2006-07-31 09:54:31	None	n/a
Sentencia	Sentencia Dictada por el Juez Ver.	2006-07-31 10:49:38	Completado	n/a

Bitacora

Transaccion	Fecha
Demanda Recibida. Asignar Sala y Juez.	2006-07-31 09:00:28
Demanda aceptada por Secretaria.	2006-07-31 09:33:04
Emplazamiento enviado.	2006-07-31 09:35:44
Esperando Contestacion.	2006-07-31 09:35:45
Contestacion Recibida. Esperando Validacion.	2006-07-31 09:38:30
Contestacion Recibida. Someter informe.	2006-07-31 09:51:57
Informe Recibido. Pendiente validacion.	2006-07-31 09:54:46

Done

Figure 6.15: Specific case status page

The other required changes were customization of the XML configuration files specifying the different types of documents, correlation queries, and relevant information (shown in Figure 6.16).

```
<config>
  <type>
    <docType>Demanda</docType>
    <docCorrelation>Header/CivilNum</docCorrelation>
    <relevant>
      <Demandado xpath='Header/Demandado/Name' />
      <Demandante xpath='Header/Demandante/Name' />
    </relevant>
  </type>
  <type>
    <docType>Emplazamiento</docType>
    <docCorrelation>Header/CivilNum</docCorrelation>
    <relevant/>
  </type>
  <type>
    <docType>Contestacion</docType>
    <docCorrelation>Header/CivilNum</docCorrelation>
    <relevant/>
  </type>
  . . .
</config>
```

Figure 6.16: Lawsuit Process Document Type Configuration

6.8 XML Storage Subsystem

The same eXist Native XML database used by Lynx to store the current version of each document on the implementation of the PRRD workflow was used for the lawsuit workflow application.

6.9 Summary

In summary, developing a completely different application requires very little code to be programmed and in particular will reduce the amount of custom GUI code required. Essentially, the only coding required is to create the Java classes that will generate the XForms, and the use of HTML and JSTL tags to construct a web-based interface.

Thus, from this experience of developing and deploying two different workflow applications using Lynx we can suggest that IDEs should provide additional support for these type of applications.

CHAPTER 7

Experiments and Results

7.1 Introduction

This chapter presents a quantitative analysis with experimental results to evidence the system's performance. In addition, this chapter provides a qualitative analysis to assess the advantages and disadvantages of the proposed architecture using the implemented prototypes for the sample Digital Government scenarios.

7.2 Deployment and Configuration

This section describes the hardware and software configuration used to run our experiments.

7.2.1 Server Configuration

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

- **Pentium 4 2.4 GHz CPU, 1 GB RAM**
- **CentOS Linux 4.3.**
- **Java SDK 1.4.2.**
- **Tomcat 5.0.28.**
- **Chiba-web-1.0.0.**
- **ActiveBPEL 1.1.6 server.**
- **eXist 1.0rc1-20060710 XML Database server.**
- **MySQL 4.1 Database.**

7.2.2 Email Server

Any standard SMTP and POP3 server can be used with Lynx. We are currently using Apache Java Enterprise Mail Server (James) 2.2.0. This email server was installed in another computer (Pentium D 2.8 Ghz CPU, 1 GB RAM, Windows XP Professional) accessible through 100 Mb/s Ethernet.

7.2.3 Client Configuration

The clients were simulated by a Java application that sent lawsuit documents with attachments through email according to the scenarios discussed in Section 7.3.1.

7.3 Performance Evaluation

We are worried that the Puerto Rico Judiciary considered over half million cases in 2005 [16]. An experiment was developed and conducted to assess the performance of the system in order to verify that the architecture achieves acceptable performance for the large workload and type of applications supported.

7.3.1 Methodology

This experiment measures the throughput of the system in terms of documents processed per minute. We simulated user behavior by programmatically submitting documents by email. Then, all the documents waiting in the email inbox were ingested by the incoming email gateway.

Using the lawsuit workflow process as reference, we measure the time it takes for the submission of the document, the process instantiation and the initial tasks of the workflow. These initial tasks include assignments, decisions and invocation of Web services to query the XML database, store a document in the XML database, and send a document to a human partner through email. We measure the time it takes to complete the initial tasks in the workflow for batches of 100 documents.

Time is measured for documents of different sizes by adding attachments of different size to the documents. The document sizes used were 2 KB, 70 KB and 200 KB. A 2 KB document is a typical XML document with no attachments. The 70 KB documents had a PDF file attachment that is representative of what a government application such as a lawsuit, or a Registry of Deeds document, would have as a complementary document. The 200 KB documents had a JPG picture attached that represents common documents that could be scanned and then attached. Finally, time was also measured for the submission of random ordered 2 KB, 70 KB and 200 KB documents.

With this information, we computed the average documents per minute our prototype application can process. Finally, the tests were repeated three times and the averages were computed.

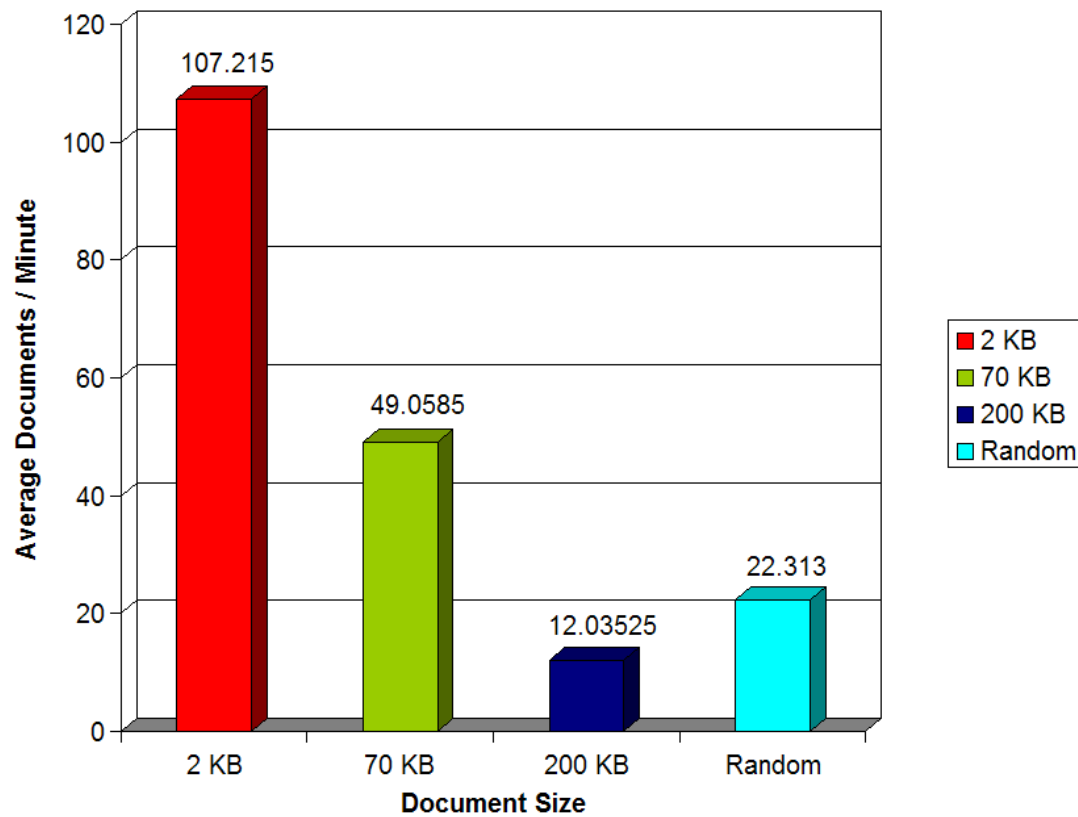


Figure 7.1: Average documents processed per minute

7.3.2 Results

The results of the performance evaluation presented in Figure 7.1 show that, as would be expected, smaller documents are processed faster than larger documents. This is because small documents require less bandwidth to transfer to and from the BPEL engine, and put a lower load on the BPEL execution engine's memory requirements.

Figures 7.2, 7.3, 7.4 and 7.5 show the time taken to complete each task in the workflow for each experimental scenario of 2 KB, 70 KB, 200 KB, and random-sized documents, respectively. It can be observed from the results that tasks 11, 29, 33 and 37 are the most time-consuming (the complete list and description of the tasks is presented in Appendix

A.) The other tasks take a negligible amount of time since they are tasks internal to the workflow. This is illustrated more clearly in Figure 7.6, which shows the average time each task takes as percentage of total time for each experimental scenario.

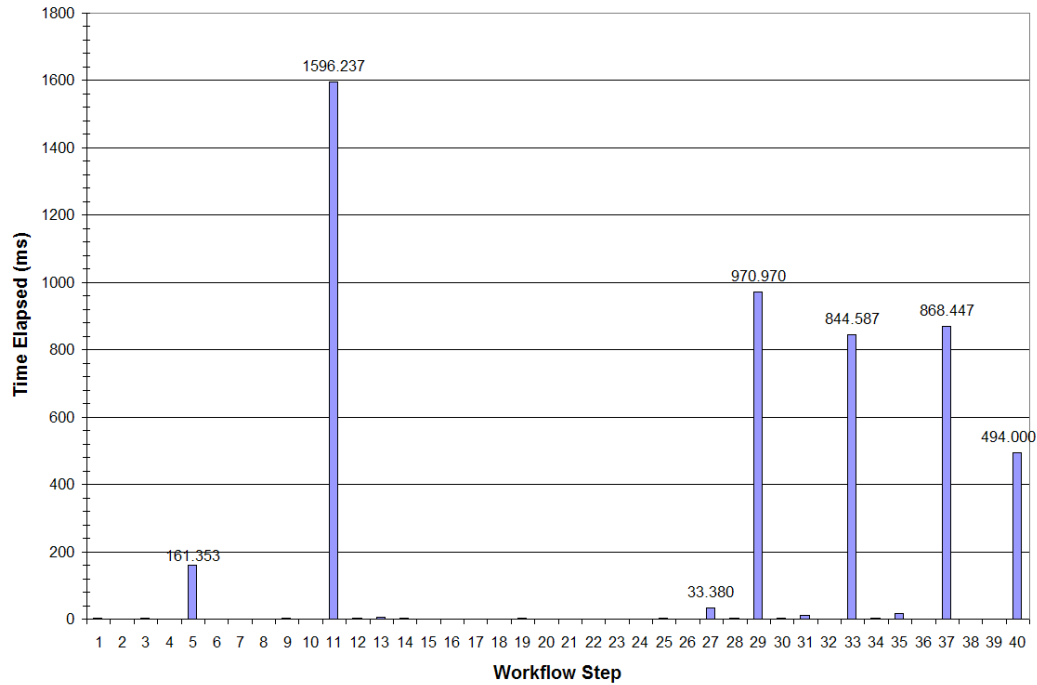


Figure 7.2: Average time taken by each task for 2 KB documents

Task 11: Check if process for this case is already running. This task invokes a Web Service that queries the Native XML database to find out if another document with the same ID is already being processed by Lynx. Indexes were enabled in the XML database for the IDs to improve the efficiency of the query that checks for an existing document. However, for the 2 KB documents, this task takes 1596 ms, or 31.7% of the total time as shown in Figure 7.6. For the 70 KB documents, this task takes 1616 ms, or 25.2% of the total time. For the 200 KB documents, this task takes 3871 ms, or 21.8.2% of the total time. For the random documents, this task takes 2296 ms, or 25.4.2% of the total time. This suggests that the Native XML database has low performance for queries where

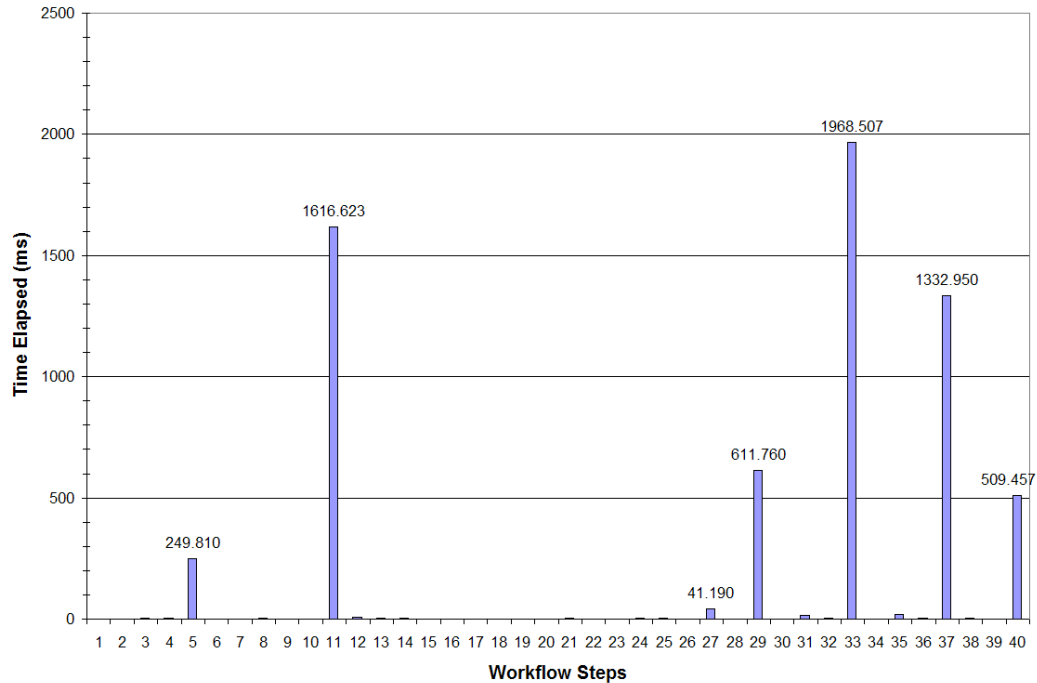


Figure 7.3: Average time taken by each task for 70 KB documents

a specific text must be searched in every element of the XML documents in the database no matter where it occurs (for example, *//id* .)

Task 29: Store Log in Native XML database. This task invokes a Web Service that stores a Log entry in the XML database. For the 2 KB documents, this task takes 970 ms, or 19.3% of the total time. For the 70 KB documents, this task takes 612 ms, or 9.5% of the total time. For the 200 KB documents, this task takes 1176 ms, or 6.6% of the total time. For the random documents, this task takes 970 ms, or 10.7% of the total time. It consumes less time than the '*StoreDemanda*' task (Task 33) because it only stores the *Header* and *Log* information, since it does not need to send any large amount of data such as the attachments.

Task 33: Store lawsuit document in Native XML database. This task invokes a Web Service that stores the lawsuit document in the XML database. For the 2

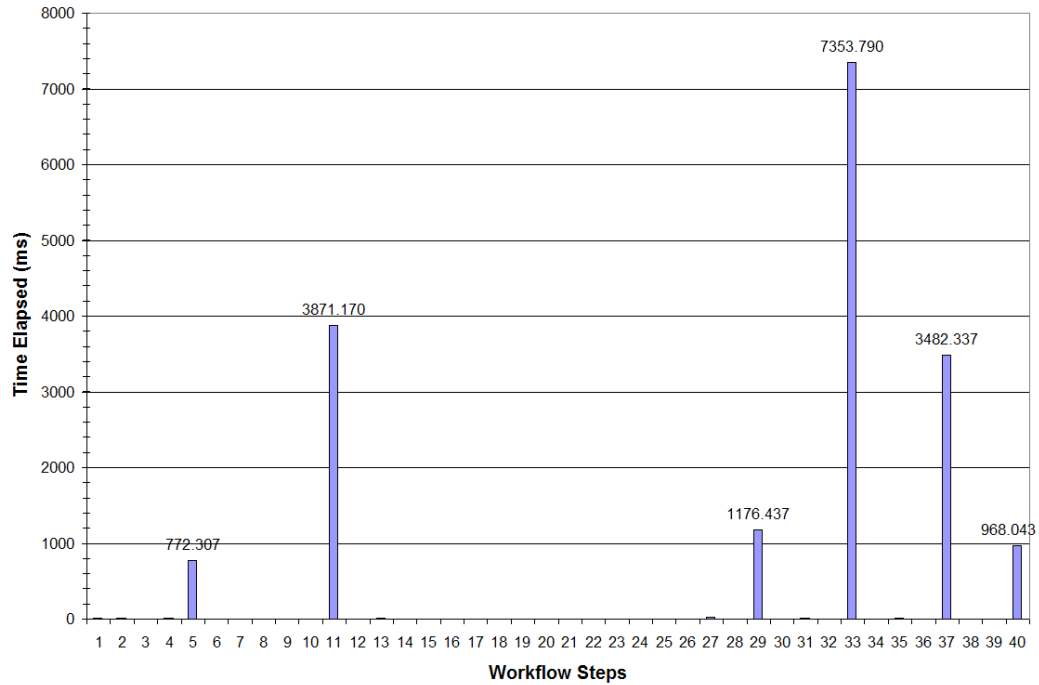


Figure 7.4: Average time taken by each task for 200 KB documents

KB documents, this task takes 844 ms, or 16.8% of the total time. For the 70 KB documents, this task takes 1968 ms, or 30.7% of the total time. For the 200 KB documents, this task takes 7354 ms, or 41.5% of the total time. For the random documents, this task takes 2706 ms, or 30.0% of the total time. It consumes more time than the '*StoreBitacora*' task (Task 29) because it needs to store the whole document, including the attachments that can be large. This is the task that takes the most time during the processes since the Native XML database needs to insert the new XML document and also update the indexes. As expected, the larger the attachments the more time this tasks takes as percentage of the total time.

This overhead introduced by the XML database may be optimized by using a new, experimental, very recent version of the eXist Native XML database. According to the developers, the new version introduces a new indexing mechanism that, when compared to the previous, represents a major improvement when dealing with complex documents and updates.

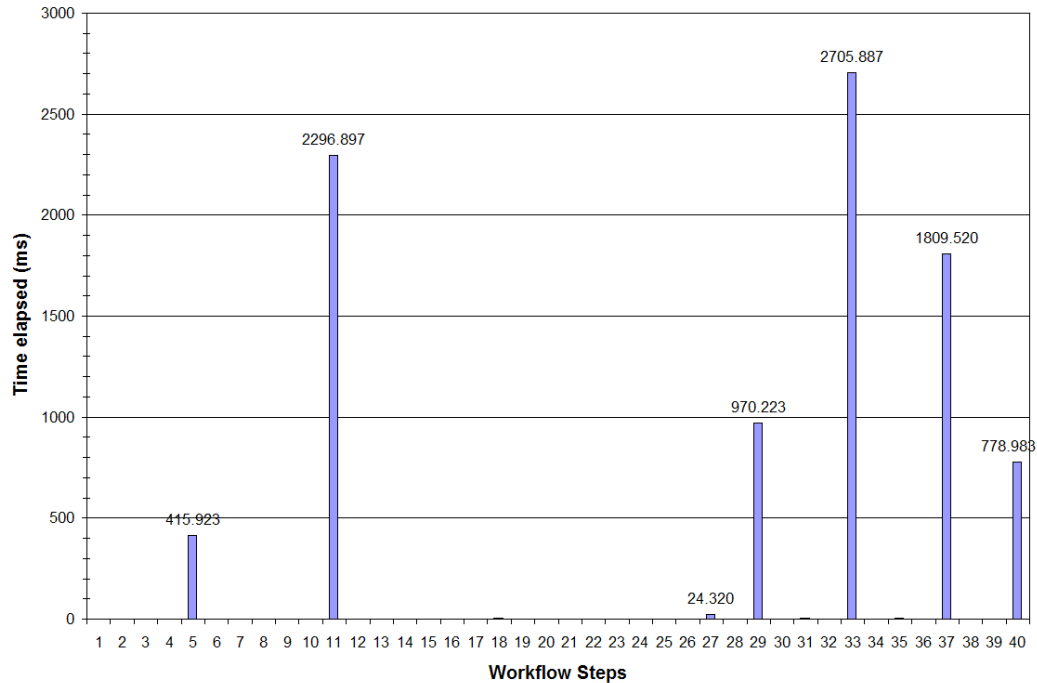


Figure 7.5: Average time taken by each task for random-sized documents

Task 37: Send lawsuit document to 'secretaria' partner. This task invokes the Outgoing Email Web Service to send the document through email to the secretary. For the 2 KB documents, this task takes 868 ms, or 17.2% of the total time. For the 70 KB documents, this task takes 1333 ms, or 20.8% of the total time. For the documents with a 200 KB attachment, this task takes 3482 ms, or 19.6% of the total time. For the random documents, this task takes 1809 ms, or 20.0% of the total time. It consumes a considerable amount of time because it needs to send the whole document including its attachments to the Outgoing Email Web Service. Also, it communicates with an external SMTP server. All these involves transforming the documents from a DOM to a String to be able to send it through email. This may be optimized by using a more efficient XML framework such as the Simple API for XML (SAX), instead of DOM4J, since SAX is more efficient although more complex to use. However, the amount of time it consumes as percentage of the total time remains almost constant.

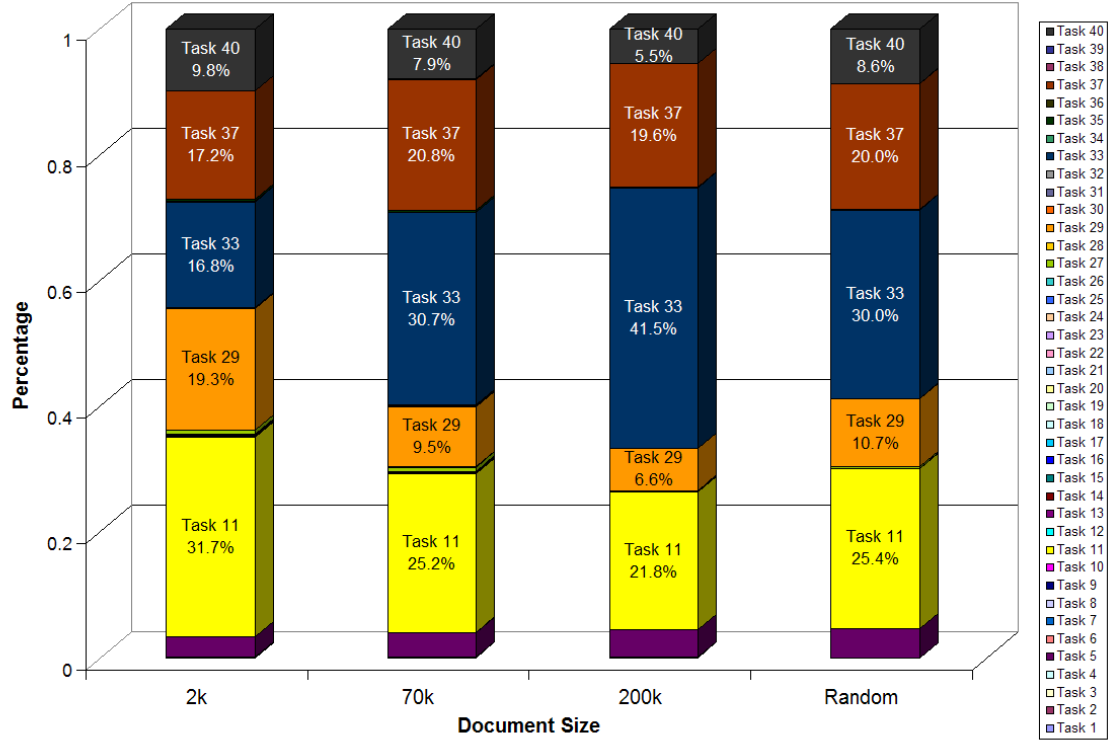


Figure 7.6: Time each task takes as percentage of total time

Therefore, for larger documents it is observed that the tasks that take the most time to execute, thus reducing the overall throughput, are essentially the tasks that involve invocation of a Web service to query and to store the documents in the XML database. This is evidenced in Figure 7.6 that shows that Task 33 is the one that takes more time as percentage of the total time for every experimental scenario except for the very small 2 KB documents. However, the total time for the completion of the tasks for the submitted documents does not equal the sum of the individual completion times for each document. This is because the execution of the processes for each document is done concurrently by the BPEL engine. This suggests that one of the limitations of workflow applications using Lynx is the access to the Native XML database. This can be seen as a trade-off between the simplicity when querying, storing and retrieving documents, as discussed in Sections 2.4 and 3.7, and performance.

We believe the throughput achieved with Lynx is sufficient for a Digital Government application since the real bottleneck in these kind of applications are the human interactions. For example, the Puerto Rico Department of Justice Courts considered 539,467 cases in 2005 [16]. This means that, on average, they considered only 1.44 cases per minute. Given the average throughput measured on our experiment, we can estimate that if on the worst case 12 documents are processed per minute, a system such the one implemented on this research can process at least approximately 17,280 documents each day, even in such a simple application infrastructure. This means that, on average, the system could process more than six million documents each year. That would be sufficient for receiving cases, leaving plenty of resources for the other steps in the process and queries regarding the cases.

However, we consider that better overall performance may be achieved if the BPEL Engine, Axis Web Services server for partner web services, XML database and email servers are all run on different computers. This would leave more resources for the memory consuming and disk intensive operations required by every server, the BPEL execution engine, the XML database, and the email server.

7.4 Qualitative Analysis

We hypothesize that by exploiting a clever interaction of technologies such as XForms, email and Web services we demonstrate the viability of implementing complex interactive applications with significantly less custom coding.

The custom code required for the Puerto Rico Judiciary lawsuit workflow application consisted essentially on the Java code required for the classes that dynamically generate the XForms. This was approximately 200 lines of Java per XForms, most of it consisting on XForms previously designed and then customized with a few lines of Java code as described in Section 5.5.

This amount of custom code is minimal when compared to estimates of code required with that required by alternative architectures in order to test the hypothesis that the Lynx architecture can reduce the amount of custom code needed for the user interface validations, calculations, error handling, form fields initialization, form data submission, logic and processing.

A summary of the steps required to develop an application is presented in order to understand the complex deployment details of form-based architectures such as Struts, JSF, Ajax, InfoPath, before comparing and contrasting them against the proposed Lynx architecture. The steps needed to deploy an application using Struts can be summarized as follows:

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>` elements
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 or 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

As observed from this procedure, an ActionForm bean class that contains the data of the form along with validations, an Action class that processes the request from the form, and a Model class are needed. This requires a lot of custom code for each form. Struts tries to reduce or simplify the need of ActionForm by using DynaForms that only require the form fields to be defined in a configuration file rather than in the form bean itself. Also, validation code can be removed from the bean by using the Struts Validator framework. The Validator framework includes several predefined commonly used validations such as numbers, strings, dates and credit cards. However, the validations must be defined using JavaScript if a custom validation is needed. Moreover, one can end writing more XML for each form. In fact, a rough estimate showed that a 20-line Java class that implements a validation in a FormAction might be replaced by more than 40 lines of XML and properties to do the same thing [25], and still needs JavaScript for custom validations.

The problem with Struts is that it combines HTML, JSP, tags and JavaScript to build dynamic pages. JSF, on the other hand, is more an architecture responsible for interacting with client devices, and its scope focuses on the presentation tier. The steps needed to deploy an application using JSF are:

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

A JSF form needs at least a Bean that manages the data, the JSF page that define the screen, and navigation rules. Furthermore, for more complex forms data conversion, validations and error handlers are also needed. Although JSF includes several standard validators, custom validations require programming and configuration of these validation rules. It would require a validator class, and a tag handler and descriptor for use them in the JSF pages.

In XForms, validation is achieved through binding a control property to a node of the instance data. The properties that can be associated with the instance data model are: *read only*, *required*, *relevant*, *calculate*, *constraint*, and *type*. For example,

```
<xforms:bind id='bind_attachment' nodeset='Attachments/attachment'
  constraint='count(attachments) $> 0'
  required='true()'
  type='base64Binary' />
```

would force the XForms to have at least one binary attachment. Likewise,

```
<xforms:bind id="bind_attachment_date"
  nodeset="Attachment/attachment/date"
  required="true()"
```

```
type="xsd:date"/>
```

would force the attachment's date to be required, and would force the XForms to display a calendar because it has been assigned the XML Schema type *date*. More complex validations are specified using XPath expressions.

In JSF most of the steps can be automated by a visual development environment where the developer can lay out the controls of a form and the navigation rules with drag and drop. In this aspect, JSF is more like a Swing for web pages [18]. However, there are also visual page designers for XForms that make it easy to construct the XForms and test them.

However, 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, these are just Model-View-Controller frameworks for building HTML forms, validating their values, invoking business logic, and displaying results in HTML.

Integrating a Web-services based workflow to allow human interaction using frameworks such as JSF or Struts is not as straightforward as using XForms. One of the main limitations of HTML forms, and of these frameworks, is the difficulty of initializing form data. In order to process a blank form into a filled form, a new document needs to be constructed piece by piece [29]. These frameworks require the beans to do this work. With XForms, form data is provided from an initial XML file, the instance data, which can be either embedded, as shown below, or external to the form definition. Thus, the instance data can be provided directly by the workflow as shown below. Also, no additional programming is needed to save each different type of document required in a workflow application using Lynx. The documents are stored using the same Web service since they are stored and retrieved as XML from the Native XML Database as explained in Section 5.8.

```

<xf:instance xmlns=''>

  <Demanda>

    <Header>

      <SalaSuperior>Mayaguez</SalaSuperior>

      <CivilNum>JD-001</CivilNum>

      <Demandante>

        <Name>Ivan P. Velez</Name>

        <SocialSecurityNumber>000-00-0000</SocialSecurityNumber>

        <Address>Mayaguez, PR 00680</Address>

        <Telephone>787-000-0000</Telephone>

        <Email>email@ece.uprm.edu</Email>

      </Demandante>

      . . .

      <Sobre>COBRO DE DINERO</Sobre>

    </Header>

    . . .

  </Demanda>

</xf:instance>

```

In addition, other approaches such as Ajax and InfoPath require scripting that manipulates the XML Document Object Model (DOM) document directly. XForms do not require any JavaScript at all. XForms feature many declarative Actions that replace scripts. Some of the XForms Actions are: *message*, *setvalue*, *setfocus*, *reset*, *load*, *toggle*, *insert*, *revalidate*, *recalculate*, *refresh*, among others.

XForms also has the ability to respond to the XML DOM events from a browser in a declarative way without resorting to scripts. XForms provides listeners, observers and handlers for all XML events.

For example, to demonstrate this, using JavaScript an HTML form may have a function that copies values from the *plaintiff's name* and the *current date* to the *attachment description* of the document:

```
<script type='text/javascript'>
<!--
function copyValues(){
    var frm=document.forms[0];
    frm.attachmentBy.value=frm.plaintiffName.value;
    frm.attachmentDate.value=now();
} -->
```

Then, somewhere in the page, the *copyValues()* function must be called, for example:

```
<input type="button" value="Copy Values" onclick="copyValues()" />
```

For a small example, this is a lot of script for just copying a value, and is hard to maintain when the scripts get bigger. XForms allows the use of the XML Events and Actions to simplify these kind of problems. The previous example could be done without scripts using a set of *setvalue* actions like this:

```
<xforms:setvalue ref="/attachment/by" value="/Header/Demandante/Name"/>
<xforms:setvalue ref="/attachment/date" value="now()"/>
```

This is easier to understand, maintain, and allows the replacement of scripting languages using a declarative syntax. Approximately six lines of scripting can be replaced by two lines of XForms specification. Yet, scripts can also be used with XForms, but only through the XML events mechanism since there is no built-in support, the script would be part of the host language where the XForms is running, for example XHTML.

Summarizing, XForms can do everything that HTML forms can do, and more, without needing client-side scripting or server side interaction. In particular XForms allows to check data values while the user is typing them in, indicate that certain fields are required, and that the form cannot be submitted without them, submit forms data as XML, submit the same form to different servers, save and restore values to and from a file, use the result of a submit as input to a further form, get the initial data for a form from an external document, calculate submitted values from other values, constrain values in certain ways, such as requiring them to be in a certain range, build dynamic forms without needing to resort to scripting.

XForms stands out with its declarative event model which is extremely powerful in spite of its relative simplicity. Many things can be done through the XForms XML events model. Also, the XForms' XML-based submission model suits well for service-oriented applications. The XForms submission model allows achieving the best separation between the user-facing interface and services underlying the user interface. It is a standard and relies on other W3C standards (XML Events, XML Schema, XHTML, CSS, etc.), which means that it integrates well with all Web technologies. Finally, XForms is cross-platform and language-agnostic, unlike JSF, or even ASP.NET, etc. There is no need to learn Java to program in XForms.

Thus, thanks to XForms, and BPEL, the steps needed to deploy an application using Lynx require little or no scripting or Java code to be programmed:

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, roles and emails. 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.

Lynx provides an open architecture that extends web service based workflows with human interaction using XForms. The development and deployment of an application using Lynx requires 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, XML Schema and WSDL.

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. Also, response may not be instantaneous if working through email. It 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. In addition, Java classes that return the corresponding XForms are

still 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 a given time. Finally, there is also a steep learning curve for BPEL, XForms, and other XML technologies.

CHAPTER 8

Conclusions and Future Work

We have presented the design and implementation of Lynx, a new 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 reduce the amount of custom code required to implement the user interfaces. Also, an evaluation of the architecture was made to assess that the architecture is acceptable for the type of applications supported.

8.1 Research Conclusion

In our research we demonstrate the feasibility of developing Web services based workflow applications with interaction through email, reducing the scripting and custom coding needed through the use of XForms. Our analysis suggests 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, and forms. Lynx also allows complete interaction through email, 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.

We also illustrate the usefulness of the Lynx architecture with the design, development and deployment of two Digital Government applications. The results of our performance evaluation evidence that Lynx achieves acceptable performance by being capable of processing a large amount of documents per minute.

8.2 Future Work

Future work effort can be dedicated towards:

- Implementing XForms e-mail client plugins, or use client-side XForms implementations when they support the whole standard, to eliminate Chiba's server-side dependency.
- Extending the BPEL language to support human endpoints natively as suggested by an IBM and SAP white paper [50]. This could effectively eliminate the need for the outgoing email Web service and the incoming email gateway.
- Implementing security for the Web services transactions and for the email messages.
- Taking advantage of the XQuery language's features to implement the web-based interface, instead of using JSP.
- Automation of the generation of Java classes that generate XForms.
- Testing Lynx with scenarios where workflow processes change frequently.

BIBLIOGRAPHY

- [1] A. Chaudri, A. Rashid, R. Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison Wesley Professional. March, 2003.
- [2] A. Kristensen. *Formsheets and the XML Forms Language*. Eighth International World Wide Web Conference. 1999.
- [3] A. Ranganathan, S. McFadding. *Using Workflows to Coordinate Web Services in Pervasive Computing Environments*. IEEE International Conference on Web Services. 2004.
- [4] BEA WebLogic Integration Business Process Management. <http://edocs.bea.com/wli/docs70/interm/bpmhome.htm>
- [5] Bonita Workflow Cooperative System. <http://bonita.objectweb.org/>.
- [6] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., *Web Services Description Language (WSDL) Version 1.1*, W3C Note, March 2001; www.w3.org/TR/wsdl.html.
- [7] ActiveBPEL, <http://www.activebpel.org/>.
- [8] Axis, <http://ws.apache.org/axis>.
- [9] Apache Tomcat. *The Apache Jakarta Project*. <http://jakarta.apache.org/tomcat/>
- [10] Apache Xindice. <http://xml.apache.org/xindice/>
- [11] *Business Process Execution Language for Web Services Version 1.1*, BEA, IBM and Microsoft, May 2003, <http://www-106.ibm.com/developerworks/library/ws-bpel/>
- [12] eXist Open Source Native XML Database. <http://exist.sourceforge.net/>
- [13] C. Richard, D. Soroker, A. Tawiri. *Using XForms to Simplify Web Programming*. International World Wide Web Conference 2005. May 10-14, 2005. Chiba, Japan.
- [14] Chiba. <http://chiba.sourceforge.net/>
- [15] D. Chakraborty, H. Lei. *Pervasive Enablement of Business Processes*. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications. 2004.
- [16] Defiende la Administracion de Tribunales su presupuesto. *El Nuevo Dia*. Page 28. June 4, 2006.

- [17] D. Ganesarajah, E. Lupu. *Workflow-based composition of Web-services: a business model or a programming paradigm?* International Enterprise Distributed Object Computing Conference. 2002.
- [18] D. Geary, C. Hortsman. *Core JavaServer Faces*. Sun Microsystems Press. Prentice Hall. 2004.
- [19] *HTML 4.01 Specification*. <http://www.w3.org/TR/REC-html40/>.
- [20] *IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J)*. <http://www.alphaworks.ibm.com/tech/bpws4j>.
- [21] *IBM MQ Workflow Email Adapter and Data Handler*. http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/topic/com.ibm.wbia_adapters.doc/doc/email/email34.htm
- [22] I. Velez, B. Velez. *Lynx: An Open Architecture For Catalyzing The Deployment Of Interactive Digital Government Workflow-Based Systems*. 7th Annual International Conference on Digital Government Research. San Diego, California, May 2006.
- [23] I. Velez, B. Velez. *Lynx: An Open Email Extension for Workflow Systems Based on Web Services and its Application to Digital Government*. International Conference on Internet and Web Applications and Services 2006. Guadeloupe, French Caribbean, February 2006.
- [24] J. Garret. *Ajax: A New Approach to Web Applications*. <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- [25] J. Turner, K. Bedell. *Struts Kick Start*. Sams Publishing. 2003.
- [26] M. Kay. *Building Workflow Applications with XML and XQuery*. DataDirect Technologies, Inc. 2006.
- [27] *JBoss jBPM*. <http://www.jboss.com/products/jbpm>.
- [28] M. Dubinko, L. Klotz, R. Merrick, T. V. Raman. *XForms*, <http://www.w3.org/TR/xforms/>
- [29] M. Dubinko, *XForms Essentials*, O'Reilly, Sebastopol, CA, 2003
- [30] *Microsoft Biztalk 2006*. <http://www.microsoft.com/biztalk/techinfo/whitepapers/adapterwp.mspx>.
- [31] *Microsoft Office Online: InfoPath 2003*. url<http://office.microsoft.com/infopath/>.
- [32] *Oracle Workflow*. <http://www.oracle.com/technology/products/applications/workflow/index.html>
- [33] *Orbeon Presentation Server*. <http://www.orbeon.com/>.

- [34] S. Graham, D. Davis, S. Simeonov, et.al, *Building Web Services with Java: Making sense of XML, SOAP, WSDL, UDDI*, Sam's Publishing, Indianapolis, Indiana, 2004.
- [35] *SMTP Biztalk Adapter*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/operations/htm/ebiz_ops_adapt_file_xybi.asp. MSDN. 2004.
- [36] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson, *Web Services Platform Architecture*. Prentice Hall, Upper Saddle River, NJ, 2005
- [37] S. Whitaker, V. Belloti, P. Moody, *Revisiting and Reinventing Email*. Human-Computer Interaction, Volume 20, Numbers 1 and 2, 2005.
- [38] T. Podgayetskaya, W. Stucky. *A Model of Business Process Support System for E-Government*. International Workshop on Database and Expert Systems Applications. 2004.
- [39] R. Bourret. *XML and Databases*. <http://www.rpbourret.com/xml/XMLAndDatabases.htm>. 2005.
- [40] *Understanding BizTalk Server 2004*.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/c245a8af-9f01-410f-b1bc-c43e725bfc27.asp. MSDN. February 2004.
- [41] *Visual XForms Designer*. IBM Alphaworks. <http://www.alphaworks.ibm.com/tech/vxd>
- [42] *Web Services Activity*. <http://www.w3.org/2002/ws>
- [43] W. Tian, G. Yu, G. Wang, B. Sorig. *Incorporate Components into Workflow Application Systems*. International Conference on High Performance Computing in the Asia-Pacific Region Proceedings. 2000.
- [44] *XML Schema*. <http://www.w3.org/XML/Schema>
- [45] *XPath*. <http://www.w3.org/TR/xpath>.
- [46] XML Protocol Working Group, *Simple Object Access Protocol (SOAP) Version 1.2*, W3C Recommendation, June 2003; www.w3.org/TR/soap/.
- [47] *XFormation*. <http://www.xformation.com>
- [48] *XQuery 1.0: An XML Query Language*. <http://www.w3.org/TR/xquery>.
- [49] *XUpdate*. <http://xmldb-org.sourceforge.net/xupdate/>.
- [50] WS-BPEL For People. <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/>.
- [51] *YAWL: Yet Another Workflow Language*. <http://www.yawl.fit.qut.edu.au/>.

- [52] Z. Liang, R. Wong. *A Lightweight Mobile Platform for Business Services Networks*. Proceedings of the IEEE EEE05 international workshop on Business services networks, 2005.

APPENDICES

APPENDIX A

Experiment BPEL Tasks

Table A.1: Experiment BPEL Tasks

Step Number	Task Description
1	Executing [/process]
2	Executing [/process/pick]
3	Executing [/process/pick/onMessage]
4	Executing [/process/eventHandlers]
5	Executing [/process/eventHandlers/onMessage]
6	Executing [/process/pick/onMessage/sequence]
7	Executing [/process/pick/onMessage/sequence/assign[@name='AssignMiscInit']]
8	Completed normally [/process/pick/onMessage/sequence/assign[@name='AssignMiscInit']]
9	Executing [/process/pick/onMessage/sequence/assign[@name='checkIsRunningAssign']]
10	Completed normally [/process/pick/onMessage/sequence/assign[@name='checkIsRunningAssign']]
11	Executing [/process/pick/onMessage/sequence/invoke[@name='InvokeIsDocumentDeployed']]
12	Completed normally [/process/pick/onMessage/sequence/invoke[@name='InvokeIsDocumentDeployed']]
13	Executing [/process/pick/onMessage/sequence/switch]
14	Condition is false : [/process/pick/onMessage/sequence/switch/case]
15	Will not execute [/process/pick/onMessage/sequence/switch/case/sequence/assign[@name='NotificationAssign']] [d]
16	Will not execute [/process/pick/onMessage/sequence/switch/case/sequence/invoke] [d]
17	Will not execute [/process/pick/onMessage/sequence/switch/case/sequence/terminate] [d]
18	Will not execute [/process/pick/onMessage/sequence/switch/case/sequence] [d]
19	Will not execute [/process/pick/onMessage/sequence/switch/case] [d]
20	Executing [/process/pick/onMessage/sequence/switch/otherwise]
21	Executing [/process/pick/onMessage/sequence/switch/otherwise/sequence]
22	Executing [/process/pick/onMessage/sequence/switch/otherwise/sequence/empty[@name='NotifyReceived']]
23	Completed normally [/process/pick/onMessage/sequence/switch/otherwise/sequence/empty[@name='NotifyReceived']]
24	Completed normally [/process/pick/onMessage/sequence/switch/otherwise/sequence]
25	Completed normally [/process/pick/onMessage/sequence/switch/otherwise]
26	Completed normally [/process/pick/onMessage/sequence/switch]
27	Executing [/process/pick/onMessage/sequence/assign[@name='BitacoraInitAssign']]
28	Completed normally [/process/pick/onMessage/sequence/assign[@name='BitacoraInitAssign']]
29	Executing [/process/pick/onMessage/sequence/invoke[@name='StoreBitacora']]
30	Completed normally [/process/pick/onMessage/sequence/invoke[@name='StoreBitacora']]

31	Executing [/process/pick/onMessage/sequence/assign[@name='DemandaBitacoraInitAssign']]
32	Completed normally [/process/pick/onMessage/sequence/assign[@name='DemandaBitacoraInitAssign']]
33	Executing [/process/pick/onMessage/sequence/invoke[@name='StoreDemanda']]
34	Completed normally [/process/pick/onMessage/sequence/invoke[@name='StoreDemanda']]
35	Executing [/process/pick/onMessage/sequence/assign[@name='SecretariaAssign']]
36	Completed normally [/process/pick/onMessage/sequence/assign[@name='SecretariaAssign']]
37	Executing [/process/pick/onMessage/sequence/invoke[@name='demandaASecretaria']]
38	Completed normally [/process/pick/onMessage/sequence/invoke[@name='demandaASecretaria']]
39	Executing [/process/pick/onMessage/sequence/pick]
40	Executing [/process/pick/onMessage/sequence/pick/onMessage]