MASTERARBEIT

Interactive Previews in Document Workflows

Ausgeführt am Institut für

Softwaretechnik und Interaktive Systeme

der Technischen Universität Wien

unter der Anleitung von Ao. Univ. Prof. Stefan Biffl
und Dr. Alexander Schatten als verantwortlich mitwirkendem
Universitätsassistenten

durch

Thomas Zwanzinger, Bakk.techn.

October 13, 2006

Contents

1 Introduction: Document Workflows and Previews 7

2 Interactive Document Preview: Definition and Properties 9

3 Document Workflow: Definition and Challenges 13
3.1 Document Workflow Definition 13
3.2 Workflow Specifications 13
3.3 Workflow Definition Languages 15
3.3.1 BPEL 16
3.3.2 XFlow 19
3.3.3 YAWL 19
3.3.4 XPDL 20
3.3.5 BPML 20
3.4 Document Output Characteristics 21
3.5 Workflow Granularity 23

4 Workflow Engines 25
4.1 JBoss jBPM 25
4.2 YAWL Workflow Engine 25
4.3 Enhydra Shark 26
4.4 BPWS4J 27
4.5 Oracle BPEL Process Manager 28
4.6 Lotus Domino Workflow 28

5 Presentation in the Browser: Thin Client Solutions 31
5.1 (X)HTML 32
5.1.1 Basic Concept and History 32
5.1.2 AJAX — Asymmetric JavaScript and XML 32
5.1.3 XForms in XHTML pages 33
5.1.4 Conclusion 34
5.2 Flash 34
5.2.1 Basic Concept and History 34
5.2.2 Programming Techniques 35
5.2.3 Flash and Workflows: Examples 37
5.2.4 Conclusion 38
5.3 XUL 39
5.3.1 Basic Concept and History 39
5.3.2 Programming Techniques 39
5.3.3 Conclusion 40
5.4 SVG 40
5.4.1 Basic Concept and History 40
5.4.2 Programming Techniques 40
5.4.3 Conclusion 41
5.5 Open Laszlo 42

5.5.1 Basic Concept and History . . . 42
5.5.2 Programming Techniques . . . 42
5.5.3 Conclusion . . . 43

6 Presentation in custom Applications: Fat Client Solutions 45
6.1 OpenOffice.org XForms support . . . 45
6.1.1 Basic Concept and History . . . 45
6.1.2 Programming Techniques . . . 45
6.1.3 Conclusion . . . 45
6.2 Microsoft Infopath . . . 46
6.2.1 Basic Concept and History . . . 46
6.2.2 Programming Techniques . . . 46
6.2.3 Formatting . . . 47
6.2.4 Conclusion . . . 47

7 Example Solution: Interactive Document Previews based on XSL-FO and XHTML/XForms 49
7.1 Introduction . . . 49
7.2 Architecture and Design Overview . . . 49
7.2.1 Extension of the XSL-FO Editor . . . 49
7.2.2 Workflow Definition and Web Application Architecture . . . 53
7.3 Transformation XSL-FO to XHTML and XForms . . . 58
7.3.1 basic elements of the transformation overview . . . 59
7.3.2 Attribute and elements support . . . 63
7.3.3 XForms elements . . . 65

8 Future Work 67

9 Conclusion 69

10 Bibliography 71

11 Figures and Tables 75

Abstract

Document Workflows are workflows whose results are documents. During automated processes, there is often the wish to proof-view documents just before they are created and change details on the content. The thesis discusses what a Document Workflow is about, what important Workflow Languages there are and examines various Workflow Engines. Some basic possibilities to create a proof-view for results of Document Workflows are reviewed, and a distinction is made between Thin- and Fat-Client Solutions. An example solution for a Document Preview is discussed based on a conversion of XSL-FO based content to XHTML with embedded XForms elements, working within a BPEL driven workflow environment.

1 Introduction: Document Workflows and Previews

Starting with the invention of Personal Computers, IT has become a more and more important part of the office infrastructure. It started with simple Stand-alone Word processing Tools and changed with the evolving network and Internet infrastructure to become an integrated software solutions for specific company's needs. Modern offices are mainly process driven and software is trying to reflect this.

In an effort to capture and standardize business workflows, big efforts have been made to try to find a form of workflow description that is able to capture all kinds of possible workflow scenarios. This led to many different solutions over the last couple of years, with competing standards created and supported by important software development companies like Sun, Microsoft or SAP. There were, of course, some serious scientific researches done on different universities which led to a number of solutions proposed. In the latest future, there was a conciliating effort to group the different standards together for a common approach, but since there are already a number of existing applications for former workflow description frameworks, it is still hard to decide which solution is most fitting for a given business workflow.

Such workflows or processes are often Document based, since many offices still rely on printed or to-be-printed letters, accounts etc., even though electronic media such as email has diminished the amount of paper documents necessary. These documents are often generated electronically based on customer data. Although this automation is necessary due to efficiency reasons, often these documents need to be customized for various reason, e.g., with special notes due to a phone call with the customer, to highlight important parts of the document or take a quick look if the document is generated as expected. This possibility can be called an interactive preview of the product of a workflow process.

Since there are various ways to represent documents and customer data beside many different Workflow Engines, this may or may not be an easy task to accomplish. Often document representations are very static like pdfs or postscript files, and some are rather open like proprietary Word Processing files. It is part of the workflow infrastructure to find the right point before the customer data is integrated in a way that prohibits any further change.

A possible solution for such scenarios is to create a form that is on the one hand as similar to the result document as possible, and on the other hand as interactive as necessary. Interactivity can be applied on many levels, starting from simple editable text fields which allow for a change of, e.g, customer data in a letter, over more complicated mechanisms that allow for adding or removing elements from lists and tables to solutions that feature complete control over the layout of the resulting document. Again, it might be difficult to find the right level of complexity that can be supported in an existing environment.

This task is especially difficult since there exist not many document solutions that are very page centric, which is needed in case of a document which shall be printed, and also feature facilities for Editing that can be bound to data. It is not easy to support every given document format in every document-based workflow system as well.

This thesis discusses what the author calls a *Interactive Document Preview*, a kind of preview of a workflow's document result which still allows last changes that might be integrated in the printed document. The thesis wants to give a broad overview about existing workflow languages, an overview how they work and what their abilities are. The idea is to let the reader know about the existing solutions to find the right choice for the given environment they should operate in. The second part of the thesis tries to sum up possible solutions for

creating an interactive preview. Basically, the author tries to separate the different solutions in browser-based solutions and stand-alone solutions. Browser-based solution are commonly called thin-client solutions since the most part of processing is done on an associating server, thus making the client application as small or “thin” as possible, making it possible to run them in a browser. Stand-alone solutions are generally specialized applications that work on their own and the function of the server is more in storing data and handling the workflow. These solutions are commonly called Fat Clients since in comparison of browser-based solutions, they need to be installed on every computer separately.

The remainder of the document is structured as follows: after discussing the nature of a Interactive Document Preview in the next section, the focus will be on presenting various possibilities to create a document output based on a certain workflow. The following chapter discuss what a workflow and especially a Document Workflow is about and how it can be defined. The next three chapters deal with well-known Workflow Engines, how they work and how they support document previews, and how to accomplish the presentation or the rendering of the preview, either in a Web browser or in a custom Application.

In the last section an example implementation of a Document Workflow is presented. The basic input of the workflow is based on XML standards especially on XSLT generating XSL-FO for having a well-defined Document output and an corresponding XHTML document with embedded XForms for the Interactive Document Preview, to be used as a preview of the to-be-generated XSL-FO document. The section discusses how this XHTML-based preview can be derived out of the XSLT document.

2 Interactive Document Preview: Definition and Properties

Most workflow models based on paper documents feature one or another "Preview" of the process result, that is the document that will eventually be printed. These previews on the result share a common characteristic: the view is not editable and all context information is lost. If the preview shows that something is amiss, the workflow essentially has to start from the beginning. This might be or not be a problem, but however, the user is still responsible to remember what was amiss and has to have the overview of the workflow process to change the workflow input so that the error will be corrected. This even might be an impossible task if the customer data is not easily accessible and the user does not know much about where the data is located and which restriction are enforced on the input.

An alternative would be to create an enhanced preview on the result Document: The preview should, based on the workflow input, show the document as it will be printed, but additionally allow to either change critical data *directly on the preview* or provide a link to a dialog for changing exactly this part of the workflow input. After making necessary changes, the workflow can be started anew or be set back to a certain point in the workflow model, and commence to create another preview. This may be repeated an arbitrary number of times until the document is ready to be published. Throughout this thesis, this kind of preview will be called an *Interactive Document Preview* (IDP) since it shows the document based on workflow input, but still has context information to react to user input.

Before we can discuss the abilities of common document workflows to create Interactive Document Previews, we have to define the properties of such a view. These properties can be derived by the situations in which such a view might be helpful, for example the following situations:

proof reading: After adding customer data to a Customer Relationship Management (CRM) System, an employee wants to create a confirmation mail to a new customer. This document is automatically created by the CRM System, which as a last step opens an Interactive Document Preview of this document for letting the employee perform a proof-reading. If there was a mistake, the customer data may be changed directly on the document and submitting the change to the system.

adding additional text to a document: An employee answers a customer phone and based on this call, performs an action on the CRM which generates a mail to the customer. In the IDP the employee adds a note concerning the just answered phone call. The document is generated and printed by printing service to be send.

fill a mail template: While talking to a customer face-to-face, an employee invokes a service which leads to the generation of a letter to be sent for a customer. The customer views the document for approval, and changes some specific text in the letter. Customer related information is automatically stored in the Workflow System while the letter is printed.

Based on these scenarios, some necessary characteristics can be derived:

- The IDP must look as much as possible like the final result of the workflow. This means the document has to capture text layout, formatting and position as well as images, lists or tables

- Elements that are based on customer data should be editable. This includes text elements as well as images, lists or tables (add list elements or table rows)
- Elements might be only editable if the viewer has the particular access rights to change these fields.
- If a document consists of several pages, these pages need to be viewed separately. If changes done on customer data demand another page to be created, this has to be reflected by the IDP as well after submitting these changes.

These necessary properties require a document which is capable of not only representing data but also act upon the content. Assmann [3] calls this kind of documents *Active Documents*. He characterizes them as document that contain both data and additionally software, macros, scripts, etc. . Active Documents can be manipulated by the Document Viewer, e.g., by filling in Forms or choosing options from a list. Documents of this kind are commonplace on the World Wide Web (WWW), often by the use of server technologies like Servlets or CGI, or by Software running on the Client as Browser plug-ins like Suns JRE[41] or Macromedia Flash Player [1].

Active Documents are often build on Templates which are instantiated by a set of data. Therefor Active Documents can often be described by their *implicit* form, which is the template, the piece of software responsible for generating its *explicit* form, the complete document, which itself is pure data. Since Active Documents are based on different components, which is a big difference to non-active documents created by word-processing software, it would be good to have a common architecture for building such documents out of different parts. As opposed to software engineering, were component-based architectures are common, this is so far not the case for Documents.

Assmann states that a document has an architecture when its embedded software (e.g. scripts) are separated from the content they compute. Assmann describes the following types of architecture in documents:

Invasive Documents: These documents expand templates by either parameterization or extensions. Often these "invasive operations" on the template are based on scripts or interactive wizards that perform actions on the template. Although there are many examples for this kind of active document, most of these templates are not type-safe, that is, any kind of content may fill the various variable parts of the templates. Documents of this type are, e.g., HTML documents with Java Script or Perl configuration scripts that store the previous selections and may be rerun.

Transclusion Documents: Invasive Documents may also feature "hot updates", which are updates on the document while the user takes some action on the document, e.g., if he rewrites a value in a spreadsheet document on which calculates are made. Such a document is called "transconsistend".

Stream Documents: A Stream Document is a transconsistend document which has an arbitrary number of input and output components. They are based on a pipeline architecture. Examples for such systems are e.g. Web-based systems for reviewing and researching

[1] http://www.adobe.com/

documents for scientific conferences: documents are uploaded step by step by defining abstracts and document summaries as well as key words and the complete text; by submitting the content, several documents are generated and changed, e.g. information about released papers as well as summary pages for the program committee chair etc.; other users of the system may add comments to a released document, and in a later state a review process will be started; All these actions are stream based and so the resulting documents will also be stream documents, as Assmann concludes.

Staged Documents: Although the previous two document architectures capture many aspects of Active Documents, they cannot explain Web systems like Java Servlet Pages. Assmann explains the concept of Staged Documents: these are documents that are computed step by step, where every computation step creates components of the document that are used by the next step or "stage" of the document. Web applications are basically build on Staged Active Documents. Note that a Staged Document need not be a transconsistend.

Assmann states that these concepts lead to a definition of a *Compositional Active Document Technology*. This is defined as follows:

> "A compositional technique for active documents relies on four concepts: explicit architectures for software and documents (including component models), invasiveness, staging, and transconsistency." *Assmann [3]*

Explicit architecture are common in software engineering since its usefulness is accepted. Such architecture enforce component models for the software and data components for for active documents. JSP tags or applets are common components of such architecture. Standing alone, these components have to be put into context by defining a composition program, in its simplest form a data-flow graph of operations on components.

Comparing such active documents with common software engineering practices, it is not common for software to be invasive, e.g. the embedding of components directly into template components. Component models like CORBA do not feature such actions since they rely on the software being available in binary form. This ability is very important for active documents, however this invasiveness is not very well defined and not typed. XML based metamodels may be the way t achieve better maintainability for invasive documents. On the other hand, transconsistency can be achieved more easily since hot updates on the active document are supported by many interactive technologies since, e.g., applet-servlet interactions are just a specialization of this concept.

A staged architecture is only quite common in web-based documents since they often consist of pure software stages which act together to form a document. The last stage has to be active document architecture since they involve software and document components. An example for this are JSPs, which feature a JSP engine that expands Java tags encapsulated in `<% ... %>` in the final HTML document.

It is clear that today's web documents are staged architecture. It makes also clear that these documents feature a complicated structure, which could be made more obvious if their architecture would be defined more explicitly. Most IDP discussed in this thesis are transconsistend Staged Documents since the wanted properties of a IDP lead to a staged architecture that has to react to user input immediately. However, their architecture is not defined explicitly, since the underlying workflow process is not detailed enough for this and Webservices are not invasive.

3 Document Workflow: Definition and Challenges

Although every Business Process is based on a workflow, workflow definitions are a relatively new and still very active field of research in Software Engineering. The following chapter sums up what workflows and especially Documents Workflows are about, how they may be defined and describes some common Workflow Engines.

3.1 Document Workflow Definition

Before discussing various Document Workflow solutions, a definition for a Document Workflow is needed. Marchetti et.al.[25] uses the following definition for a workflow:

> "A workflow is 'the automation of a business process, in whole or part, during which documents, information or tasks are based from one participant to another for action, according to a set of procedural rules' " *Marchetti et.al. [25]*

More specifically, Marchetti et.al. also discusses a "Document Workflow" is about:

> "As a Document-centric Workflow or Document Workflow (DW) we refer to a particular workflow in which all activities , made by the agents, turn out to documents compilation. It can be viewed as the automation and administration of particular documents procedures. In other words, a DW can be seen as a process of cooperative authoring where the document can be the goal of the process or just a side effect of the cooperation." *Marchetti et.al. [25]*

'Agents', in this definition, are the active parts of the process and process or transform a well-defined input to a particular output. Agents may be classified in external and internal agents. External Agents are actors, either humans or automated software tools, which performs actions based on the particular workflow. Internal Agents are more generic software tools which are therefor mostly build in the workflow, e.g., rule-based document transformation tools, or tools that create documents or aggregate a number of documents. For defining and presenting workflows, workflow charts may be used. These are often simple state-diagram like Charts like in Figure 1.

3.2 Workflow Specifications

Workflows can be considered in different ways, Van der Aalst et.al. [47] define workflows on different perspectives. One perspective is the Control-flow perspective. Activities are defined as well as their execution ordering based on different constructors that allow a definition of flow-control like sequences, choices or join synchronization. In this perspective, Activities are the smallest, atomic parts that may be combined to Composed Activities. The Data perspective takes data definitions and puts them atop the control perspective, defining additionally pre- and postconditions for Activities. The Resource perspective defines roles and actors which may trigger Activities. The Operational perspective defines actions on which Activities interact with underlying applications. These applications may manipulate data which is part of the workflow.

Although all perspectives of a workflow are important, for understanding an workflows effectiveness, the Control-flow perspective is the most important, since the other perspectives

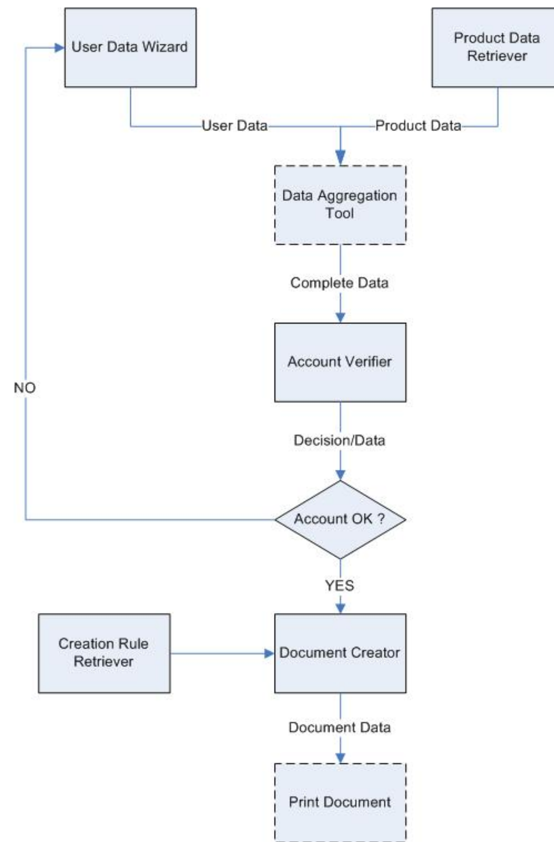

Figure 1: A simple Workflow Chart

either rest on it (Data perspective) or are complimentary. Although most workflow definitions are based on the same control-centric parts like sequence, iteration, splits and joins, these parts are not implemented the same way, especially in complex situations, e.g. , in some workflow languages loops only have one entry point and exit point, while others may have an arbitrary amount of both entry and exit points. This leads to a difference in the expressive power of workflow languages as well as a different suitability for specific workflow applications.

Aalst et.al. [47] tries to identify common requirements for workflow languages based on workflow patterns. The idea is, that it is very complicated to sum up the capabilities of different workflow definition languages. Aalst proposes common patterns in workflows as a measure for the expressiveness of such a language: very expressive languages support most of the patterns directly within their basic language constructs. They are also a way to look at common problems at an abstract level and sum up many common situations and problems in workflows, representing a way to evaluate or "benchmark" a given workflow languages ability to handle such situations.

A basic pattern is, e.g., the Sequence Pattern: an activity in a workflow process is enabled after the completion of another activity. Another example is the Parallel Split Pattern, which splits a thread of execution into several threads that may be executed separately in any order or in parallel. In most workflow definition languages this is implemented by an AND-split. A more sophisticated pattern is the Implicit Termination Pattern, which states that a workflow process may be terminated when the process has nothing more to do, that is, if there is no active activity and no activity can be made active. This is commonly the case in workflow languages since they often rely on a special final node in the workflow which terminates the process. This does not satisfy the pattern since there may be parallel activities left that are still executed. However, it might be easy to transform a given model to a model that has only one final node that terminates the process, but generally this is not the case.

3.3 Workflow Definition Languages

Since more and more companies try to capture their Business Logic in workflows and try to support this Logic with specialized Software Tools, Standardized Workflow Definition Language become more and more popular for offering an abstraction from specific workflow tools. An overview over all existing workflow Languages would justify a whole thesis and is out of scope for this document. Instead this thesis concentrates on interesting and active academic approaches as well as some important standards supported by various companies to define a Workflow Language

Most of these standards are based on XML, since it allows a readable and easy to validate definition. Various Workflow Languages have been proposed, and still there is not a widely excepted solution. Historically, there is also an evolution from various standards to a new consolidating standard like in the case of BPEL or academic approaches.

Comparing Workflow Languages is generally not easy since there is a difficulty in comparing their semantics and expressive power, as Dustdar et.al. [10] points out. Generally, Workflow Languages describe the workflow by using directed graphs with activities, that are invoked. This is done by using a series of call-flow elements like sequences, parallel activities or synchronization points. These elements are not designed and implemented the same in different Workflow Languages, e.g., an *AND Join*, which waits for some activities to finish or *XOR Join*, which waits for one of several parallel activities to finish, are implemented in different ways.

This leads to a situation where it is difficult to incorporate different workflow standards within a single workflow process over different companies.

3.3.1 BPEL

The Business Process Execution Language (BPEL [15]) is an attempt on defining Business workflows based in XML. This standard was defined by the Web Services BPEL Technical Committee, consisting of companies like IBM, Microsoft, Siebel, BEA and SAP. This standard was preceded by the Web Service Flow Language (WSFL) standard defined by IBM and XLang defined by Microsoft. In an effort to harmonize the different standards, BPEL was born. The standard is today managed by OASIS.

BPEL is a statically typed Turing complete scripting language, whose definition is written in XML. It focuses on the interaction of Web Services as smallest part of the Workflow Definition, due to this, BPEL is also called a *Web service Composition Language*. Web Services are independent Software Tools using a common interface definition, the XML-based WSDL [7] language. The interaction between Web services is also XML based by using the SOAP [6] standard. This language defines the construction of messages transmitted between Web Services and the invoker of the Web Service. BPEL relies an WSDL to be sure of the synchronization behavior of the interface and on correct typing of the interface parameters. For security and authentication, on the top of SOAP there may be used WS-Security [31].

What BPEL tries to accomplish is to create another Web service out of composition of other Web services, this composition is called a *process*. Thus, the interface is defined by means of WSDL. The workflow itself is defined, e.g., by `<invoke>` (starting a Web service), `<receive>` (waiting for an external message), `<reply>` (generating some response of a input/output operation), or `<wait>` (wait for a defined period of time). There are also elements that provide loops and conditional statements over these primitives. BPEL defines a set of services on which different operations may be performed, which are expressed by *portTypes*. BPEL also features exceptions.

Zimmermann et.al. [53] state, that through experience they gained the understanding that there are still some functionalities which are often needed but still difficult to implement in this model. An example is the construct of a “Back-Button” functionality in a workflow: a certain step shall be repeated after a unsuccessful data validation. Although it is possible to implement this behavior with a number of control-flow constructs, the solution is not comfortable. They also conclude, that the various techniques (BPEL, WSDL, SOAP, XML, application programming, e.g., in Java) require much know-how from developers or leads to a negative impact on developers productivity.

Since BPEL was created by leading Software companies, there already exist a lot of implementations as these companies have slowly incorporated BPEL into their products, e.g., the Oracle BPEL process manager (see also section 4.5), JBoss jBPM (see also section 4.1), IBM WebSphere Business Integration Server Foundation[2], Microsoft BizTalk[3] or SAP NetWeaver[4].

There is one very important feature missing in BPEL: There is no activity that specifically supports user interaction. Since business workflows are typically not completely automated but rely on user action and input, this concept does not match the reality of business workflows. A

[2]WBI Server Foundation: http://www-306.ibm.com/software/integration/wbisf/
[3]MS BizTalk: http://www.microsoft.com/germany/biztalk/default.mspx
[4]SAP Netweaver: http://www.sap.com/platform/netweaver/index.epx

common example is the creation of a new account: In a simple scenario, another user has to approve the creation of a new account and the account is created. In a more complex scenario, the creation of the account has to be approved by several persons. There might be the need that these people do not know about the decision of other users since this might degenerate the decision result. In other cases, a decision process must be declared in a way that a user does not even know that other users are involved in the decision.

In another scenario, a user may have to assign task to other users. Other situations demand that, if some timeout for an action or user task is reached that there is an escalation, that is, that another user is informed that an activity has not been finished and that some action should be taken, or another user should take an action.

This example should show that including people interaction within workflows is not a simple task. Most current implementations of the BPEL standard ease the inclusion of user interaction by supplying special webservices which are commonly called Tasklists. A Tasklist is a list of actions that have to be taken by a human user. When the user finishes a certain action, the Tasklist is updated, which may lead to the invocation of other webservices or the sending of SOAP messages to commence the running workflow. In this way, the workflow engine supports user interaction without extending or violating the BPEL standard. However, these facilities are not standardized and are not made visible within the BPEL specification.

Since the lack of user interactivity support is one of the major disadvantages of BPEL, some effort was made to enhance BPEL to include this kind of action. The result is a whitepaper [23] created from IBM and SAP for an extension of BPEL called BPEL4People, which enhances BPEL with options to include user tasks. Some vendors are have already implemented the proposal in their workflow engines or have announced to do it in some way, e.g., Oracle BPEL Processmanager. However, since there is still no concrete specification of BPEL4People, the claimed implementation of the proposal can only be seen as a reference to the abilities that are stated . Since BPEL4People is only a proposal, workflow engines may vary in the way they implement it.

In the discussed whitepaper, the authors identify some basic scenarios for user interaction, including the following:

People Activities: These activities represent a communication step with a human. A common scenario is that the user has to finish some task. When the task is finished, the workflow may commence.

People initiated activities: Often some people in a company are allowed to start a certain activity, while others are not allowed to do this. To represent this in BPEL, users shall have different roles. Some activities may only be started by people that have a specified role in the company. This concept is very important for reflecting employee hierarchies in companies.

People Managing Long-Running Processes: This case includes scenarios where, e.g., a workflow waits for a user task to be completed within a certain time limit. If this action does not happen within this timeframe, there are several possibilities: the workflow may commence with the data arrived so far; the workflow may be aborted; or a special user, an administrator may be notified and charged with the decision how the workflow should continue. Since working with incomplete data may result in undoing already taken actions, the last possibility is often required.

Transition between Human and Automatic Services: It should be possible to switch tasks to tasks with human interaction from automated tasks and vice versa in a non-disruptive way.

To make all of this possible, the *people activity* was introduced as a new type of activity. People Activities are not realized as a piece of software, rather they represent an action the user has to take to finish this activity, which is called a *task*. It is possible to assign tasks to users either at development time, deployment time or runtime. The infrastructure for runtime tasks for users is generally stored in organizational structures outside of BPEL, e.g., in a LDAP directory. To link such a structure to concrete persons, a *People Link* is introduced.

A People Link represents a group of persons who are associated with a certain People Activity. To bind a group of persons to a concrete number of persons, a *People Query* is introduced which defines a set of persons in the organizational structure in a way that is specific for this structure, e.g., for a SQL Database, a query could be a simple SQL script.

Besides these grouping of persons and binding of persons to activities, there are generic roles for users: *Process Initiators* are users which create an instance of a process; *Process Stakeholders* are persons that can influence the progress of an activity; *Potential Owners* are users which may entitle to claim and complete an activity, these potential owners become actual owners by claiming an activity; and *Business Administrators* are users which may perform special administrative actions on process, e.g., resolving missed deadlines; In contrast to the process stakeholder, it may have control over a set of instances of a process.

A concrete task waiting to be done by a People Activity is now resolved if a concrete user claims this task, performs the required action and signals the activity that the tasks has been done. This user has to be allowed to claim this task by being in the group of users defined by the People Link, which has been resolved to a concrete set of people by a associated People Query. These tasks to be done by users have a set of properties, typically information on what has to be done, what the priority of the task is or a set of data which is needed by the user to perform the task. The work done to complete the task itself is invisible to the workflow engine, as it might, e.g., involve a phone call to a travel agency to reserve a flight. A claim to a certain task can also be revoked.

To work with tasks and people activities, a client application is needed for the user which presents to him task he might claim and work with. In this way, task may be in one of 4 distinct states: the Ready state is the initial state in which the tasks is created; the task moves to a Claimed state in which it is claimed by a user, it may move to the Completed state when the user finished the action to take, or the task might go to the Fail state when the task cannot be completed.

Finally, BPEL4People also suggests Services which are implemented by persons as a standalone task which is defined outside of any process definition. This service might be reused by several processes.

BPEL4People is a try on defining a complete integration of user interaction in BPEL. By trying to capture the full extend of interaction, the result is a very complex set of new elements in the BPEL workflow language. However, there are already work-around solutions by software vendors which implement a BPEL driven workflow engine. These solutions do not extend the BPEL standard and work very well in most situations, although they are not standardized and may not be able to be exported to other workflow engines. Since user interaction is commonly a very significant part of business workflows, BPEL4People might eventually be implemented

by software vendors in the future, although the lack of a well-defined specification may prohibit general usage.

3.3.2 XFlow

The XML-based XFlowML, the XFlow Markup Language, was first defined by Marchetti et.al.[25]. In the XFlow document the logic of a workflow completely independent of the underlying Workflow Engine is captured. The authors try to define the workflow in the point of view of the agents running the workflow: agents are defined with a particular "role" and a set of possible input sources and output destinations, featuring an "action" definition. This definition includes access policies to document fields. Additionally, elements may contain conditional elements from the World Wide Web Consortium (W3C) XSL[9] Language Recommendation, the `<xsl:choose>` and the `<xsl:if>` elements. Those elements use XPath statements to test on data instances.

Specific agents are described in the a "rolechart", which also maps one or more roles to one agent. Data Definitions are defined in a XML Schema [8] document. Based on these 3 documents, an underlying Document Workflow Engine may execute. Marchetti et.al.[25] provide a Case Study and a reference implementation for XFlow. Based on XFLow there is also an approach for a XFlow-based implementation called XFlow 2 [18]

3.3.3 YAWL

The "Yet Another Workflow Language" (YAWL) is based on the idea of using high-level Petri Nets [43], which have been enhanced with color, time and hierarchy, to describe the control-flow perspective of an workflow. Van der Aalst et.al. use the approach to create and verify this Workflow Language by looking at Workflow Patterns[47] (see also section 3.2)

Van der Aalst et.al. analyzed many current Workflow Languages and argue, that these Language fail to support often not more than 50% of a predefined set of Workflow Patterns natively, that is without going a long way to work around missing features. The YAWL language proposed can handle these Workflow Patterns by the use of Petri Nets.

Van der Aalst et.al.[46] explain in their work that this approach is feasible because Petri Nets have a formal semantic, are state-based and have spawned various analyzing techniques. On the other hand, there are several shortcomings like a missing support for subprocesses or the difficulty to define global actions like the removing from tokens without knowing were the token actually is.

A YAWL specification defines a number of Extended Workflow Nets (EWF-nets) which are also called Top Level Workflows and are defined by a tree-like hierarchical structure. Elements of this structure are Tasks and unique Input- and Output Conditions. Tasks can be instanced multiple times and even be terminated when a certain number of Tasks finish successfully. Joins for Tasks include AND-, XOR- and OR- Joins. An important advantage of YAWL is that it gives the language a solid formal semantic to work with.

Additionally to the control-flow perspective, the other perspectives like data perspective need to be defined to create a complete Workflow Language. Van der Aalst et.al.[45] describes these aspects in their work.

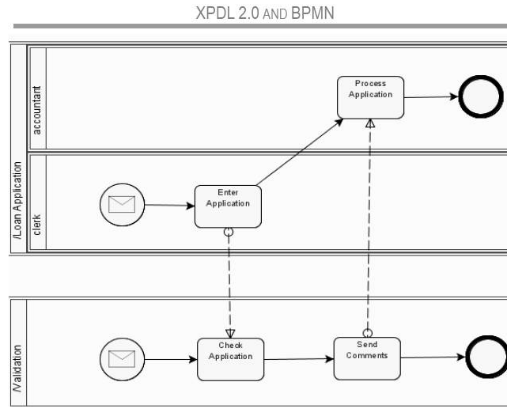

Figure 2: A simple XPDL workflow chart with Activities and Events [38]

3.3.4 XPDL

The XML Process Definition Language was created by the Workflow Management Coalition (WfMC), a consortium of various companies. Shapiro [38] explains that it was build around the experience gained with the Workflow Process Definition Language (WPDL), which was published in 1998. XPDL was created because of the arising XML standard. It ports the syntax from WPDL to XML, but leaves the semantics of the former WPDL as it is. Both standards missed a specific graphical representation, which was later created in 2004, the Business Process Management Notation (BPMN). BPMN also enhanced XPDL in some specific mechanisms which eventually lead to the creation of XPDL Version 2.

XPDL distinguishes between *Activities* and *Transitions*. Activities contain the business logic, containing, e.g., WDSL Definitions for Web services that should be called etc., Transitions define the control-flow, which conditions there are, etc. . XPDL features the standard XOR and AND types of forking and merging of control-flows as well as events like Exceptions or Message-based triggering. Activities and Transitions may be grouped in a *Compound Activity*. Compound Activities may be called for execution in an independent process, in this case the Activity can be called with parameters. There are various other elements like loops, multiple instances or Compensations.

Several tools exists to run Simulations on XPDL definitions. Such a Simulation may give valuable insight in the performance of activities, schedules or resource characteristics. However, XPDL does not feature a formal semantics model like YAWL.

3.3.5 BPML

BPML is a standard defined by BPMI.org [34] and is supported by various companies like SAP, Sun and Versata. Van der Aalst [48] evaluates this Workflow Language with a set of predefined Workflow Patterns. It directly competes with, e.g., the BPEL standard. BPML depends heavily on Web service related standards like WSDL and is sometimes also referred as being a Web Service Composition Language.

BPML consists of 5 main elements:

Activities: These elements perform the basic function that drive the workflow process. Activities may be composed, these are called *complex* Activities in contrast to *simple* Activities

Process: A Process is a complex Activity that can be invoked by other processes. An independent Process is called a *top-level* Process, while processes running in the context of another process are called *nested* Processes; nested process are either called by by the *call* and *spawn* activity

Context: Contexts are used to define an Environment in which Activities operate, they can be used to exchange data and coordinate execution;

Properties: Properties are part of a Context and have a name and a type, just like a variable in most programming languages and act as attributes of a process.

Signals: These elements are used for synchronization; they work like messages that are sent in certain situations and can be waited for

Defined Activities include simple Activities like call, action and raise, as well as complex activities, e.g., flow-control like choice or for-each. BPML is based on XML. Van der Aalst concludes in his evaluation that the support for the predefined patterns is comparable to BPEL. Moreover, Moon et.al. [29] defines a transformation algorithm from BPEL to BPML and vice versa, for example basic activities in BPEL become simple activities in BPML. The combination of receive and replay activity can be replaced by an action activity that has a child call activity which invokes the specified process. Although some transformations are simple, some concepts cannot be translated directly, e.g., event and fault handling of BPEL. The authors conclude that although a transformation is possible, an automatic transformation is not, due to the fact that the expressive power from BPEL is different to that from BPML. However, with human interaction it is possible.

3.4 Document Output Characteristics

The result of a document workflow is a printable document. These documents share some characteristics: the smallest unit is a *page*, which has a fixed width and length; such a document may consist of one or several pages. Based on this characteristic there are many possible available document types that more or less satisfy these criteria and may be used as workflow output. Since most printers work with the postscript input, these data types have to be converted to postscript to be printed. The most common document types are:

Postscript: Postscript documents are actually *programs*. King [22] explains that these program steps are used to create an image, generally directly within the imaging device. Basic elements of postscript page content include graphics, defined by lines and Bezier curves as well as sample image data. These elements could be rotated, positioned and scaled.

Since there is only one program per document, and a document mostly consists of more than one page, this means that for viewing one page of a document, the whole document has to be generated. Although this is not practical for viewing documents on a monitor, it is acceptable for printers.

PDF: PDF was introduced by Adobe to replace postscript to improve the performance of viewing documents. The page content elements of pdfs are similar to postscript. These elements are not embedded into a programming language but are organized in objects. This allows to independently compute single pages out of a multi page document. Pdfs may also include other information like bookmarks, thumbnails or hyperlinks. On the World Wide Web(WWW) pdf is considered the primary file format for printable documents.

HTML: HTML is the basic language of the WWW and consists of semi-structured text. Since the basic idea of HTML is to be viewed by a browser which interprets the language with additional style information (generally done by Cascading Style Sheets (CSS)), it was not designed for easy printing and does not support the concept of a page layout.

XSL-FO: This data format is based on the Extensible Markup Language (XML) and uses a more generalized, but strict approach of a semi-structured, textual description language. XSL-FO describes "flow" content in human readable language, beginning with page size and structure up to text blocks and graphics. Being part of the XML Family, it supports other XML languages directly like Scalable Vector Graphics (SVG) or XChart, a chart description language. The idea of XSL-FO is to be generated based on arbitrary XML data and the XML Transformation language XSLT. The document itself is usually converted by a renderer to other file formats like pdf or postscript, making them easy to print.

(La)TeX: TeX is a text-based document description language which is build on user-defined or build-in macros. A TeX processor generates postscript or pdf output based on the data, with style information coming from macros. This format is often used for scientific papers or large documents like books.

ODF: The Open Document Format (ODF) [32] is a open standard originally used in the OpenOffice.org office suite, and was approved as a OASIS standard in 2005. It was standardized to provide a common office data format to facilitate the exchange of documents between different software vendors. Common file extensions used by ODF files are .odt for text documents or .ods for spreadsheet documents. A ODF document either consists of a XML file with the root element `<office:document>` or a number of files compressed in an JAR archive. Since archives may also contain binary content easily, this method is mostly used.

Although there is already some support for ODF, e.g., in OpenOffice.org and KWrite, MS Office suite 2003 does not support ODF. Microsoft will release a translator software to support it, though, and may support it in coming versions of their software.

RTF: The Rich Text Format (RTF) [26] is a text file format created by Microsoft and supported by most word processors. Like XML documents, RTF is designed to be human readable. It is maybe the most common file format supported by word processors. Although RTF is a 7-bit format, by using escape sequences it is possible to include, e.g., Unicode characters. The format uses control code starting with a backslash character and followed by control commands.

The most common of these output formats is pdf, often generated out of XSL-FO documents since many Workflow Engines or Frameworks are build on XML based documents.

3.5 Workflow Granularity

Workflow definition languages are a very useful tool for creating an abstract description for a given business workflow. There is definitely a big advantage in such a model, however, like with most technologies, workflow definition languages have to be justified within the context they shall be used.

Here is an example of a business workflow: The user selects a customer within an application for creating an payment account. The registered customer is retrieved by using an available webservice. After that, the user enter items that the customer has bought. To do that, another webservice is started which retrieves the production to be bought. The created list of product items is then sent to another webservice which creates an account for the customer which shall be printed.

Clearly this business workflow can be modeled with any workflow definition language. The control flow is rather simple and the webservices can be easily integrated into the workflow process. However, there are several points in this workflow that generally do not fit well with a workflow engine driven approach: frequent, synchronous user interactivity and a small granularity of activities.

Lets have a look at the activities that can be derived from the above workflow description. Let us assume that every activity is implemented by a webservice, since this a common prerequisite for using a workflow engine:

- The registered user is retrieved with a webservice from any database.
- Product items are looked up by another webservice, possible more than one.
- The account is created by using a webservice which takes the account data and transforms them into the account document, which is in turn printed directly by the user.

First of all, the workflow is not very complicated on the surface. There are only 3 webservices involved, and they do not perform very much: two of the webservices represent only a data lookup, the last action involves some data transformation process, which generally does not need much computational power and is performed rather quickly. All of these webservices only contain synchronous actions, that is, the user will wait for the webservice to answer its request immediately.

This fact reduces the benefit of webservices significantly. One of the major advantages of workflow engines is that they do not require synchronous action but support asynchronous messaging. Often a process may be finished days or weeks after it has been started. Working with webservices just like they were better function calls robes them much of their benefit. On the other hand, the overhead involved in calling a webservice is significant. Matjaz et.al. [20] state that compared with Java Remote Method Invocation (RMI), webservices can be slower in an order of a magnitude. He measured the instantiation and the method invocation time by using a simple call scenario where the called party did not do any processing but rather just answered the call immediately. This is important since Matjaz wanted to find a good comparison value. The reason for the big difference between those two methods is that XML serialization is inherently much slower than binary serialization that is used with RMI.

This shows us that one important thing to keep in mind while using webservices and workflow engines is that having many small webservices leads to a significant performance overhead. If a user has to wait for the result of these webservices, this might no be acceptable. Instead, a

solution including more specialized solutions like calling Servlets or remote procedure calls is more efficient and suitable.

There is also the fact that workflow engines generally lack a good support with debug facilities. Finding errors in workflow scenarios, especially if the webservices are many but do not do much, can be difficult.

In the scenario discussed, there is also a lot of user interaction going on: During the workflow, the user has to provide information about the customer and which products he wants be buy. User interaction itself is not a very strong feature of workflow engines, e.g., BPEL does not define any constructs for user interaction, which in turn lead to the introduction of BPEL4People (see also section 3.3.1). There is often additional overhead necessary to integrate user interaction within workflow systems by, e.g., wrapping them in another webservice which is in turn triggered by the user application if it has finished.

Since the need for good tools for handling user interaction is obvious, many workflow engines include webservices that facilitat user interaction. Often these webservices feature "work-lists", which represent a list of task a user has to perform to finish a certain task. Users have to access these webservices with external applications and look up their tasks. These task form a "To-Do" list for the user. When he has finished an action, the webservices has to be informed about that by the external application, the webservice in turn sends an message to the webservice which added the task to the user to signal the completion of the user task. Although many workflow engines provide these mechanisms, they are not standardized in any way, and effectively form a block within workflow definitions which cannot be ported to another workflow engine if these webservices cannot be integrated with the other workflow engine as well.

Here is another example of a business workflow: A customer wants to lend money from a bank. The bank employee collects the information from the customer, the amount of money he wants to lend, personal information, for example his address and profession. The bank employee realizes that the amount of money the customer wants to lend is too for him to grant, so he tells the customer that the decision will be made in the following days if he gets the money. With the collected information, the employee starts the workflow, in which two senor employees are informed by mail to take part in the decision process. After both have agreed, the process starts putting the credit information to the bank data system, and sends the employee an information per mail that he may grant the credit to the customer. The customer is automatically send a letter which states that the money will be transfered to his bank account.

This scenario is much more suited for being implemented by using workflow engines: the task can easily be implemented by webservices, since all the actions are asynchronous, may take a while to complete and are thus not performance relevant. There is also not much user interaction involved.

The conclusion is that although workflow engines are very good tools to model business process, a point has to be made that not in every case the use of them is very efficient nor suitable. Workflows that require much user interaction, quick responsiveness and require a lot of synchronous use of webservices are better implemented using existing techniques like HTTP requests to Servlets or remote procedure calls.

4 Workflow Engines

Since every business process defines a workflow in one way or another, IT Solutions for companies mostly have an underlying workflow as well. However, most of these solutions consists of different components and actions that are more or less hard-wired together. They lack a central definition of their workflow model, their actions, etc. . With the advent of Workflow Definition Languages, Workflow Engines were implemented which operate on a given Workflow Definition Language.

A *Workflow Engine*, also called *run-time system*, has a well-defined interface and an underlying execution mechanism which performs processes and activities and coordinates in the workflow involved components. The instructions for the workflow as well as data bindings, roles and security information come from the Workflow Definition.

Today there are numerous Workflow Engine implementations for different Workflow Languages. The following sub-chapters summarize some of the more common Workflow Engines. The Workflow Engines are chosen to feature different Workflow Languages as well as different license policies ("Open Source" versus commercially available software)

4.1 JBoss jBPM

jBPM is a Workflow Engine designed by JBoss, a division of Red Hat [19]. It is written in Java and works from either a application server or as a standalone Java package. It is licensed with the Lesser Gnu Public License (LPGL). It is possible to use one of the following Workflow Definition Languages: BPEL, for Web Service Applications or jPDL for embedded process coordination requirements. jBPM is currently in its version 3.1.1 .

The Workflow Engine consists of several modules:

- A Request Handler, which forwards tasks to the right process or user
- Several Interaction Services, which expose existing application interfaces to the processes
- A State Manager, for managing state information like variables by using, e.g., databases
- A Process Monitor, for tracking and auditing the status of a Workflow
- A Process Library, which stores the process definitions (e.g. BPEL documents)

jBPM was build to be usable with various Workflow Definition Languages. This is done through the internal use of jPDL, which is also a directed graph core engine. On the top of this, other Languages may be integrated, as is done with the BPEL support. JBoss also features a graphical user interface for creating Workflow Definitions, the Graphical Process Designer (GPD). Current development efforts are the creation of a native security and authentication support as well as an extension for the existing testing framework.

4.2 YAWL Workflow Engine

Yet Another Workflow Language (YAWL) is a Workflow Definition Language defined by Van der Aalst et.al.[46]. There is already a big implementation effort to create a complete and broad realization of this Workflow Language[33], however, there is no complete implementation yet.

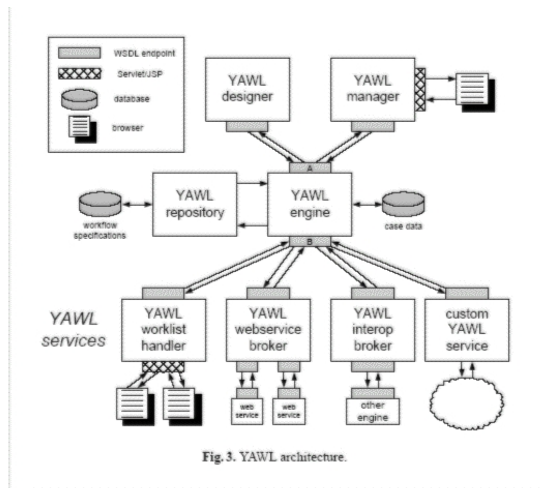

Fig. 3. YAWL architecture.

Figure 3: YAWL Architecture[45]

Van der Aalst et.al.[45] provides an architectural description of an implementation of YAWL. The central component of the architecture, as shown in Figure 3, is the *YAWL Engine*, which instances Workflow Definitions out of the *YAWL Repository*. Acting elements in the YAWL definition are the YAWL Services, including predefined Services like a *Worklist*, which represents a list of actions to be taken by an actor in the system and which have to be finished to trigger other events. This is interesting since most Workflow Engines provide an build-in support for such elements, whereas this approach decouples it from the engine and makes it a standard YAWL Service. The Access to existing Web services is achieved by the *Webservice Broker*, which handles the mostly diverse interfaces of Web services and inserts their events and action into the YAWL system. By the *YAWL interop broker*, it is also possible to allow to connect with another Workflow Engine.

The data perspective of YAWL relies heavily on XPath and XQuery, shying away from any non-standardized solution. Local and Global variable bindings are resolved based on these standards. This aspect is as well as the control-flow perspective is already fully implemented and even extended with newer research results.

The operational and resource perspective is based on the YAWL Services, which require a WSDL definition and are thus based on a Webservice interface. The resource perspective is currently worked on since there is the wish to extend the functionality with the results of the research done in the area of Workflow Resource Patterns. There is also Tools support including a Designer Tool and persistence support. The implementation is available through the GNU Lesser General Public License (LGPL)[52].

4.3 Enhydra Shark

Enhydra Shark is an Open Source Workflow Engine, supported and extended by the Enhydra.org community[51]. It is based on the XPDL Workflow Language published by WfMC without any proprietary extensions and is completely built in Java. It is not specifically written for use in a web application technology like Servlets or Java Beans, it is a simple extensible library, where modules may be replaced and modified. This makes Enhydra Shark more a

Workflow Engine Framework, with an existing standard implementation for XPDL. Enhydra also provides an editor for the visual generation of workflow models with XPDL, the Enhydra JaWE (Java Workflow Editor). Enhydra Shark is available by the LGPL.

The Workflow Engine will be released in its version 2.0, based on the second beta, the version will support extended plug-in support, bug fixes and includes DODS version 7, which is an open source relational/object mapping tool.

Enhydra Shark is used in various Open Source projects like OfBiz, which is a enterprise automation software project, in which Enhydra Shark is integrated as the internal workflow engine. Open University Support System (OpenUSS), which is a publishing and information portal system, uses Enhydra Shark as well. AKBANK is a online banking company and its retail banking project works with this Workflow Engine.

4.4 BPWS4J

BPWS4J is created by IBM Alphaworks and implements a Workflow Engine based on BPEL [16]. IBM Alphaworks creates implementations on current new technologies and is more to be seen as an research implementation of such a technology rather than a complete IBM product. It is also not supported and not recommended for commercial use, although these project may be integrated in future IBM products.

BPWS4J includes a runtime engine to execute BPEL Workflow Descriptions, a tool for evaluating BPEL documents and some sample workflows. There is also a plugin for Eclipse[5] to create and modify BPEL documents. It features a tree-view of a given BPEL document as well as support of requirements specifications. BPWS4J is written in Java and is tested with Websphere Application Server 5.0 [6] and Apache Tomcat[7]. It is currently in its version 2.1 from Mai 2004. BPWS4J is more like a research project as a real business application and may be seen as a kind of example implementation of BPEL.

For starting the Workflow, IBMs Workflow Engine needs a correct BPEL document for the workflow specification, a WSDL file for the Webservice interface it should offer for the workflow, and WSDL descriptions for the services that may be invoked by the workflow. After startup, the runtime makes the process available through the Webservice interface and, if necessary, provides the WSDL description for this Service.

Haataja et.al.[13] use BPWS4J to integrate supply-chains, using BPEL as a Workflow Language and BPWS4J as a the underlying Workflow Engine and evaluate the performance of components used and the suitability of BPEL as well. They tested the version 2.0 of BPWS4J and came to the conclusion, that in this version, BPWS4J is not a mature program yet:

- BPWS4J does not support the complete BPEL specification, e.g., there is no true asynchronous messaging, dynamic partner binding is not implemented yet, and not all data types are supported;
- Due to a bug, two BPWS4J Engines cannot cooperate together since BPWs4J does not support SOAP messages containing multi-reference encoding, which BPWS4J itself uses;

[5] Eclipse IDE : www.eclipse.org
[6] Websphere Application Server 5.0: http://www-306.ibm.com/software/webservers/appserv/was/
[7] Apache Tomcat: http://tomcat.apache.org/

- Some concurrency and performance problems were encountered;
- BPEL4J does not contain any direct security mechanisms, which is an important feature for workflow systems across company boundaries;

Although there exist Workarounds for the most annoying bugs, Thus Haataja et.al. conclude that there is still some work to do and that BPWS4J has yet to prove that is not “only” a research project.

4.5 Oracle BPEL Process Manager

Oracle, being one of the founders of the BPEL Workflow Language, has created a Workflow Engine based on BPEL, the Oracle BPEL Process Manager [35]. Currently it is in its version 2.0 and was implemented in Java. It is available for testing or prototyping purpose by the Oracle Technology Network (”OTN”) License.

The Oracle BPEL Process Manager comes with an eclipse plug-in for creating BPEL documents. This is a graphical tool with which one can easily construct complex workflow scenarios. Additionally you have to provide a WSDL interface description to make the resulting workflow accessible. The tool creates a jar file containing all necessary files, which can be deployed to the BPEL Process Manager.

The Process Manager includes a testing page for simple BPEL process where one may start process by inserting all necessary parameters in a automatically generated HTML Form. The result document is then presented to the user. Additionally, there is a trace output viewer which can be used to view the messages created and passed between the different Web Services, as well as look through the control-flow decisions made internally.

An interesting element of the BPEL Process Manager is the User Task element: Very often in workflows, user input is required. To simplify this, the User Task is a service that requires some user input to succeed. It is accessible externally by a Java API. Typically in a process that requires user input, an upstream process element creates the User Task and assigns some data with it. This suspends downstream process nodes. That task and its associated data may then be accessed through an external application’s user interface. Once the user has completed the required external process, the User Task API is called to assign a status of complete (or any other appropriate value). This in turn activates the remaining downstream processes.

4.6 Lotus Domino Workflow

IBMs Lotus Domino Workflow is a standalone application that works on the top of IBM Lotus Domino. It is used mainly as an extension of the Lotus Domino Server and it allows to create business processes whose activities can be better tracked and performed with less errors and more consistency. The latest version of Domino Workflow is version 7 and is also build for easy integration for other Lotus products like Lotus Domino Document Manager. However, it is possible to make the workflow model accessible through a Web Service Interface.

Workflow Definitions are created with the Workflow Architect, a visual tool that publishes the created Workflow Definitions to the Workflow Engine. Lotus Domino workflow uses no public Definition Language Standard, but features a proprietary solution. Debugging is possible within the Architect. The workflow engine consists of Lotus Domino Databases, used for

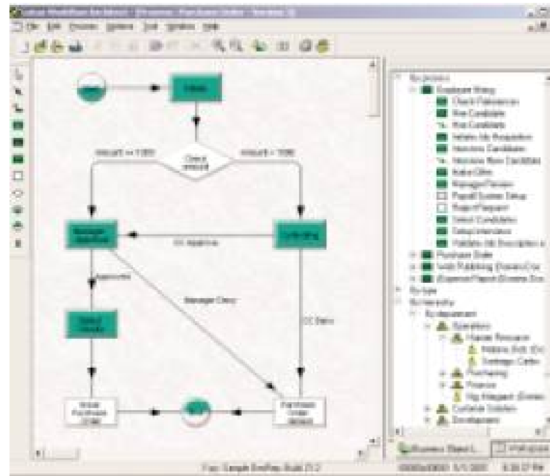

Figure 4: Domino Workflow Architect[17]

storing, e.g., Workflow Definitions, state information and archives. Three Lotus Domino Application are created for the Workflow Engine: A Process-definition Database, which stores the Workflow Definition; a database used to store Definitions and Groups for roles and resources part of the workflow, and an application which stores definition of build-in elements for creating workflows.

A third major application is the Domino Workflow Viewer, which makes it possible to view status and context of a running Workflow. Regarding to Van der Aalst effort to compare workflow implementations [44], Domino workflow lacks in the support of 6 patterns, making it still an average implementation in this comparison. However, as Van der Aalst points it out, the fact that Lotus Domino Workflow is tightly integrated in a Groupware systems may reduce the need for certain patterns.

5 Presentation in the Browser: Thin Client Solutions

With the advent of HTML in offices and further the introduction of HTML Forms and Java Script, developers soon realized that a Webbrowser may be more than simply a tool to read and download data from the Internet — it is also an interactive application platform. Since then, more and more technologies allow to make a browser a application platform.

The basic building block of interactive Websites displayed on Web Browsers is HTTP. It allows to retrieve and put content from or to a HTTP server. Although first only used to retrieve HTML pages and images, this protocol also allows to transfer arbitrary data from the Web browser to a server and vice versa. For retrieving content, the HTTP command GET is used, for pushing content on a Web Server, mainly PUT, POST and for parameter content also GET is used.

Build on this mechanism, the following technologies try to use a Web browsers facilities to create a Web Application on the browser. In a Client-Server architecture, this is typically called a *Thin-Client*: Since there is no need to install any applications but Browser extension or plug-ins, which may be installed via the browser as well, no effort need to be put on software distribution on individual PCs, or maintenance of Software on them. Furthermore, the necessary computational power on the Client is minimal, only the browser needs to be able to run.

The main disadvantage of Thin Clients are several:

- Most computational load is on the Server System, it has to calculate any operational results, has to manage data, security and authentication mechanisms, which is a non-trivial task
- Web servers become a single point of failure and, if they crash, block any further usage of the application
- Since there are many browsers, implementations of standards differ which makes it difficult to design Web pages that can be viewed on different browsers and look the same

However, the cost benefit of a Thin Client architecture is not neglectable and thus Thin Clients become more and more popular. This is also powered by the fact, that browser technology is advancing rapidly, allow to parse XML based languages directly on the client rather then pre-compiling them on the Server. Various plug-ins and browser extension allow to transfer some former server jobs to the client reducing the load on the sever. Some of the following technologies for Thin Clients rely on these browser extensions. However, when using script languages to perform actions like input validation on data, it is important to keep in mind that the server-side application has to check the received data again, since generally it is not a good idea to trust the client applications, since heightens security risks.

The most commonly used browser is still MS Internet Explorer, currently in its version 6, and is shipped with every Windows XP installation. Although Internet Explorers plug-in support is somewhat laking comparing other browsers, executable ActiveX enhancements allow to build very rich dynamic Webpages. However, these elements are also very dangerous in the light of antivirus protection. In its version 6, Internet Explorer supports basic parsing of XSLT stylesheets. CSS 2 is partly supported, which matches the efforts of other Web browser.

Mozillas latest attempt on Web browsers is Mozilla Firefox, which is rapidly gaining popularity, although its users are still a minority compared with the users of the Internet Explorer. It

features an open plug-in support and can display XUL based User Interfaces, which makes it possible to create Application GUIs on a Web browser. It also supports CSS 2 to some degree but differs in the implementation with the Internet Explorer.

There are several other Webbrowser implementations like Opera and Safari, as well as older browser implementations. The diversity of the used Webbrowsers and their different capabilities have to be taken account when designing Thin Client applications. However, in a companies intranet environment, generally there is only one browser installed on office computers, which greatly helps in creating quality Thin Client solutions and makes it possible to use browser technologies that are not generally supported (e.g. XUL).

5.1 (X)HTML

5.1.1 Basic Concept and History

The base technology for implementing Thin Clients Solutions is of course HTML. HTML has now been around for several years: 1993, IETF published the HTML standard in its version 1.0. Later the W3C overtook the responsibility for the HTML standard evolution until 1999 were it published the latest HTML version 4.0 .

With the advent of XML, HTML, which features the same tagged structure, was ported and standardized by a W3C recommendation to the strict XML syntax and is called XHTML Version 1.0. XHTML 1.1 separates XHTML in various modules for the purpose of allowing designers of devices which want to use XHTML to define which part (modules) of XHTML is supported. Such devices may be mobile devices, digital TV solutions, etc. . A recommendation for XHTML Version 2.0 is still in progress.

(X)HTML is the single most commonly used technology for bringing Websites through the web browser to the user. Several other standards help to create sophisticated websites: CSS allows to separate content from style and JavaScript allows limited client-side scripting and interactivity. Audio and Video plug-in technologies rely on HTML as a hook for their plug-ins, e.g., if the plug-in is not supported, a HTML hyperlink may link to the plug-in download portal.

HTML as well as XHTML is build with structured tags just like any other XML document. HTML allows to define Tables, Lists, import images, etc. . Additionally, with the support of CSS, floating bodies and more sophisticated table layout is possible. However, HTML does not feature any drawing or animation facilities.

An important part of HTML is its Forms support. Forms allow for user input, Form elements include Textfields, Checkboxes, Comboboxes, etc. . By the use of JavaScript, input validation is supported via scripting, a validation by DTD or XML Schema is not possible, especially since Form elements can not be bound to a specific XML data definition.

5.1.2 AJAX — Asymmetric JavaScript and XML

HTML, being designed originally as a purely static document format, is still a very inefficient tool for creating dynamic content as generally many interactive actions require to load a whole website down, even if the actual change of the website is small. Under the term AJAX (Asymmetric JavaScript and XML) tools are created for reducing this overhead. Data is exchanged with the Webserver by using XMLHttpRequests, using mostly XML for data representation. The idea is that rather then creating HTML content on the Webserver, XML data is transfered between client and server, triggering a rebuild of the website on client side if necessary. This

means that only data for changes on the website are transmitted and not a whole new site. This is done by heavy scripting on client side with Java Script. Several web technologies already utilize AJAX techniques, e.g. Microsoft's .NET or JavaServer Faces (JSF).

However, using AJAX has some drawbacks as well, as Strahl [40] states. In his article he reviews the way AJAX is often used in webpages and comes to the conclusion, that although AJAX offers benefits, often these benefits of a better user interactivity comes with a significant performance loss, which is not intuitive at all and contradicts the main idea of AJAX. One of the major advantages of AJAX should be that it is not necessary to reload a complete page to change a page's content. However, in practice AJAX techniques are used to add interactivity that was not even considered when using non-AJAX powered websites. This often leads to a significant amount of data which is passed back and forth between server and browser.

Using AJAX also comes with another price: Added complexity. Adding AJAX scripting code to a website is still needing a lot of scripting although AJAX libraries help a lot. However, by using AJAX scripting one still introduces additional JavaScript code into the website which is hard to debug and maintain. By using this technique, you shift responsibility for input validation or query lookup to the client, but due to browser incompatibilities and less debug facilities, this code is hard to implement and to maintain. In some cases, this code "only" makes your website more responsive, but does not make the load on your server less, since, e.g., input validation still has to take place on the server since generally the client application cannot be trusted.

Additionally there is the problem that AJAX needs JavaScript and still there are a lot of users which do not have activated JavaScript in their browser. From a business companies point of view, creating a website which may not work very well for some users is not a good idea. So some efforts have to be taken to make the website work with or without JavaScript in an acceptable way.

Strahl concludes that AJAX offers possibilities for creating web applications that behave like Rich Client Applications, which look just like any normal application on the client computer. However, it is very difficult to do so, and to do in an efficient way that can still be handled on the implementation side. But is it not better to create, e.g., a Java application running on the client computer in first place, which communicates with the Server? What about using .NET applications distributed via websites for such interaction? AJAX is still a very useful technology which just has to grow up a little bit more.

5.1.3 XForms in XHTML pages

XHTML, as a part of the XML family, allows for integration of non-HTML tags directly in the XHTML document. Although the build-in support for those XML technologies is very limited, the many standards are either very useful for XHTML website or are specifically written to be integrated within XHTML.

One of them is XForms, which is a W3C recommendation for Web Forms. Although there is already a Forms support in HTML, this support somewhat lacks in several areas: for example input validation relies on JavaScript, which is not easy to implement and maintain, and the result is completely unstructured.

XForms is build around the idea to separate Form content from style. Instead of defining Textfields and Comboboxes, Input or Select elements are defined. How, for example, a Select Element is rendering device (mostly a browser) is up to the browser or external definitions: it might either be a series of Checkboxes, a Listbox or a Combobox or any other selection

mechanism. This is especially useful for small devices like mobile phones which may choose the style of the element by their rendering possibilities.

XForms requires to bind any form element to a XML data field. If an XML Schema definition is provided, the input may also be validated upon the definition directly on the rendering device. Output elements allow to display information based on XPath expressions over the bound XML data structure. XForms elements may also be hidden from the user based on XPath expression evaluation.

XForms was designed to be used within a XHTML 2.0 document. XForms is generally not yet supported in Web browsers although there are various, either free or commercially, available plug-ins for some browsers. XForms may also be used in the future for generally defining User Forms outside of the XHTML context.

5.1.4 Conclusion

HTML was the basis of Thin Client Solutions and is still enough to design remarkable dynamic document representations. Although (X)HTML is not a definition language based on pages, it is still easily capable of representing such pages. Due to the CSS support in browsers, it is possible to do remarkable things with (X)HTML documents. The various Form elements available and the JavaScript support of most browsers is enough to create a Interactive Document Preview. That said, with the advanced support of XML-based languages that may be embedded in XHTML files in browsers, the possibilities will even grow. Scalable Vector Graphics, for example, which is described in one of the following chapters, makes a very good addition to XHTML, since it allows to improve websites visually even more and add much to interactivity.

The future support for XForms is very interesting for creating Forms that can be used on various devices, especially for mobile devices, since there is no need to write a Form especially for a device, since XForms tags do not include layout information and are rendered specifically on the client device in a way that fits the device best.

However, currently the lack of XML data definition support in standard XHTML requires to add a significant amount of scripting to the XHTML document to allow the necessary dynamic behavior, and may require several round trips to the server for more complicate tasks on the view. The support of XForms in browser adds some important features like expandable list to XHTML documents and is better suited for dynamic document representations.

AJAX technologies allow XHTML pages to get the responsive feeling that Rich Internet Applications should offer. However, there is still a mismatch in what the technology can offer and what is actually implemented on websites. Often the use AJAX leads to websites which are more responsive and offer some enhancement in usability, but generate more load on the webserver than before and require a significant implementation effort. It is important to verify if there is enough know-how in using AJAX and enough advantages in using it before adding this technology to a website.

5.2 Flash

5.2.1 Basic Concept and History

Macromedia Flash is a powerful Web Animation Product from Adobe [1]. Flash 1.0 was first released in 1997, based on its predecessor Future Splash. It featured the ability to easily create animations and display them on Web Browser over the WWW if the Browser contained the

necessary plug-in. While Future Splash Animation Player was based on Java and very slow in its implementation, the Flash Player was a binary plug-in. Macromedia Flash will in future versions be called Adobe Flash, however the author of this thesis will use its current name.

Flash itself is created via the Flash Professional Integrated Development Environment (IDE). It creates a Shockwave File (SWF) which can be run within the Flash Player which is essentially a Virtual Machine. Flash consists of vector and raster graphics, with can be animated. Due to the small size of the Flash Player Plug-in and the possibility to easily create animations, it is a convenient solution for adding animated content to a web site.

In early versions, Flash could only be used for animations and had little possibilities for interaction. Later versions included first the *Macromedia Generator*, which allowed enabled to separate content from design. The second version of Macromedia Generator allowed for Server Side Creation of SWF. Macromedia Generator was than replaced by the Coldfusion Server. Later versions of Flash also include the ActiveScript Language which allows to add scripts to Flash, similar to JavaScript for HTML Websites.

Flash files are generally embedded in HTML, although this is not standardized by the W3C Consortium. Recent Web browser like Firefox and Internet Explorer include a version of the Flash Player Plug-in. Flash Animations have become a common part of Websites since it adds many multimedia features to HTML Websites. Especially Flash Adds are very common. The evolving possibilities of Flash, its scripting language ActiveScript and the Coldfusion server, more and more serious business applications are build based on Flash.

5.2.2 Programming Techniques

Flash features a timeline element split in frames. Frames may contain arbitrary content and are shown one after the other in a predefined speed, measured in frames per second. This allows for creating animations, and since this was the first major use of Flash, Flash animations are still called “Flash Movies”.

One significant part of the IDE is the possibility to draw vector-based graphics. Flash features similar structures like SVG. Probets et.al.[36] describes the basic structure of Flash Vectors: Vector objects are described and then brought on the stage by a transformation matrix. Since the transformation matrix may be changed in every frame, this allows for animation and scaling of vectors. Primitives are lines, curves and position parameters. These primitives are assigned properties like fill and stroke colors.

Since SVG and Flash feature both the creation of vector graphics, Probets et.al. compares SVG with Flash. Although both share basically the same primitives, SVG is hierarchically structured were Flash is essentially linear. Groups, compound vector primitives, in SVG support subgroups which inherit properties from up the hierarchy. In Flash, groups may not contain subgroups although they can be assigned unique identifiers (which is also possible in SVG). These groups can be assigned transformation matrices for frame to frame transformation of the whole group.

Interactivity can be added to Flash by *ActionScript*, currently in its version 2.0, which will be discussed in this section. ActionScript is a Script Language comparable to JavaScript in HTML. Actually, both are based on the ECMAScript Standard adopted by the European organization ECMA[8]. This standard was created because there were two very similar scripting

[8] http://www.ecma-international.org/

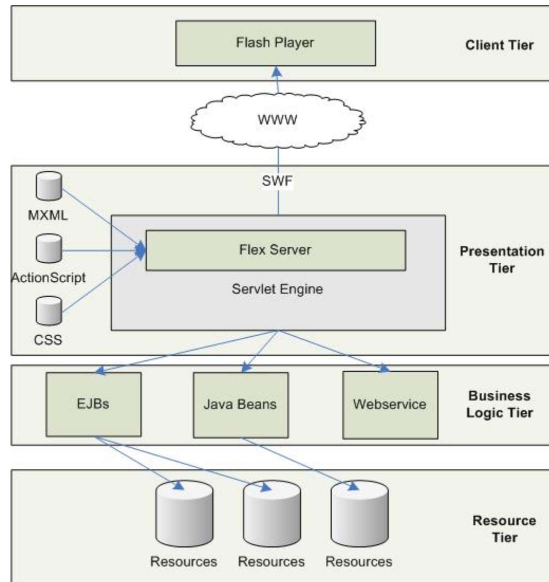

Figure 5: An example Web Application Architecture with Flex

languages evolving for HTML Websites, *JScript* from Microsoft and *JavaScript* from Netscape. ECMAScript roughly corresponds to JavaScript 1.4, and ActionScript is based on ECMAScript.

Due to this identical ancestors, JavaScript and ActionScript share the same rules for creation of variables, arrays, functions, logical expressions and flow-control, as described by Surveyer[42]. The languages differ in their predefined objects. In case of ActionScript, these are elements like the Stage, which corresponds to the content of a frame, or the Movieclip. Additionally, ActionSricpt features components for Form creation like Textfields, Checkboxes or Radiobuttons, which in JavaScript is part of HTML. For working with XML, ActionScript contains two class, “XML” and “XMLSocket”, while JavaScript lacks a corresponding feature.

ActionScript makes it possible for Flash to send or receive data over HTML or TCP/IP from external sources like any kind of Web server or locally stored Text/XML files. Note that Flash files may only retrieve external data from the local file system *or* from a network system due to security reasons. Data retrieved this way is stored either in variables or special container classes like the “XML” class.

Build on the top of Flash and ActionScript is Macromedia Flex, currently in its version 2.0. A description of Flex can be found on the Macromedia Website [14]. It features a XML-based description language MXML [9] and new programming libraries to be used with ActionScript and was introduced to improve the abilities of Flash related technologies to create Rich Internet Applications (RIA).

It consists of a *Flex Server*, which runs in a Servlet Container. Its responsibility is to combine ActionScript and MXML files to a binary SWF file and transmit it to the Client, the FLash Player. MXML files are thus only executed on the Server. In an n-tiered architecture, Flex can be seen as the presentation layer, responsible for UI specification, Input validation and Data binding for Form elements, an example architecture is shown in Figure 5. The benefit of this architecture is, that many actions that in common Web Forms based on HTML, for

[9]Macromedia Flex: http://www.adobe.com/products/flex/

Figure 6: A Flash Application

input validation or data formatting round trips to a Web server are necessary, whereas the Flex application runs in the Flash Player and may perform these actions.

MXML files contain tags for creating arbitrary Flash elements like all kind of common Forms, but also data grids, tree views, tab navigators or menus. These tags are augmented with ActionScript, e.g., adding Handlers and Error Handlers to elements or providing data binding to Form elements. ActionScript functions are usually defined in another file and are imported by MXML import tags. MXML features the Document Object Model Level 3 Events (DOM 3)[10] and Cascading Style Sheets (CSS) by the W3C. It also supports communication with Web services and interaction with J2EE Objects.

5.2.3 Flash and Workflows: Examples

There are some examples for the integration of Flash often with Flex with existing Web servers and workflow technologies. In a Whitepaper from SAP[4], the authors present a solution in the area of business information retrieval: Business Information available from different resources shall be integrated in a single Web Application. This is necessary to enable fast decisions within the company the software should be used in. The focus is on decision makers, who are not technicians and demand a flexible, user friendly tool which presents them only the information they need in a appealing way. There were already other solutions which enabled this kind of data retrieval in way, but in use these solutions lacked the expected flexibility. These solutions should now be enhanced with an easier and more intuitive User Interface (UI) with Flash.

A significant part of the implementation was the retrieval of information from different data sources. The solution allowed to integrate data source into the application with a Java Connector Architecture (JCA) [11] compliant resource adapters (e.g. JDBC, SAP Query Connector) and Business Warehouse Info Cubes.

The problem was solved based on SAP NetWeaver Visual Composer and Macromedia Flex. SAP Netweaver acts as the development tool to create individual UI. It is the integrational part of SAP products since the SAP Business Suite. It is a flexible tool which is able to cooperate

[10] DOM3: http://www.w3.org/DOM/
[11] JCA: http://java.sun.com/j2ee/connector/

with a number of useful technologies like Microsofts .NEt, Java or standards like HTTP, XML and Webservices.

The goal was to create a rapid development designer so that the created GUI can be changed quickly and conveniently. However, since not developer will use the tool, no programming or scripting should be necessary to create these views.

The SAP Visual Composer arranges data elements that shall be part of the UI and binds them to data. This can be done without writing a line of code. The vies are created as portal pages and SAP iViews. These document can be uploaded to the SAP Enterprise Portal in which they can be accessed by a browser. The UI is transformed to either HTML, Dynamic HTML or Flash (with Flex). The application supports three distinct views for creating UIs: A Model view shows a graphical representation of data flows, visualizations and events; a What-you-see-is-what-you-get (WYSIWYG) view lets you fine tune the resulting UI; and a Preview View allows to test-drive the result before compiling it and uploading it to the SAP Portal. In the background, a the UI is defined with the SAP NetWeaver Visual Composer Language.

Flex is used in this scenario because it has several advantages compared to traditional XHTML solutions. One of them is Flex ability to store state information and act upon them, which significantly reduces the communication overhead between the server and the browser client. The visual benefits from Flash over HTML are also very significant, especially for an application which should ease the retrieval of information and display it to the user in the most convenient way. The ability to interact with the most important data interfaces, Web-services, XML over Http and Remote Method Invocation with Java, allows to transfer much computational effort to the client. Important security capabilities, cashing as well as session management is supported.

All these capabilities are supported server-side by using two separate components within the SAP Netweaver Web Application Server: a Visualization component which consists of Macromedia Flex components and SAP Flex components; and a Control Model which is the Visual Composer Server which handles data access and control.

5.2.4 Conclusion

Flash is a proprietary technology of Macromedia allowing to build Web Applications that run in a virtual machine on the browser via the Flash Player Plug-in. Since all modern browsers support Flash Plug-ins or already have them integrated when being shipped, this is no obstacle for creating serious applications. Another prerequisite is using the Flex Framework, since it allows to build Flash Applications out of a MXML and ActionScript. This makes it possible to separate Content from the User Action Control. The data model and business logic is part of the underlying architecture, which may be a J2EE Framework, another Servlet (since Flex runs in a Servlet container) or a Web service. This makes it flexible enough to be used by various underlying Workflow Engines.

The most obvious disadvantage is that by using Flash with Flex, you are bound to a specific product of a vendor, thus creating a dependency which may not be acceptable. The Flash Player is part of the Macromedia Licensing Program[12], thus making it impossible to be included in a pure open source software. Although the specification of the SWF files is published, there is no other common implementation of the Flash Player, nor is the SWF format an open standard, it

[12]Macromedia Licensing Program: http://www.adobe.com/licensing/

is controlled by Adobe.

Nevertheless, to nearly unlimited potential of vector graphics, form elements and user controls makes it possible to create very sophisticated Interactive Document Previews. Since MXML makes it possible to create a UI based on an XML technology, it can be integrated into existing XML based Frameworks.

5.3 XUL

5.3.1 Basic Concept and History

XUL (pronounced Zool) is a GUI description language designed by the Mozilla applications [30] for their project. XUL is based on XML and provides an easy way to describe standard widgets like buttons and textboxes in a portable way. It was already introduced in 1998 when the Mozilla Browser became to focal point of Netscapes Open Source initiative. XUL is not a public standard but makes heavy use of existing standards like CSS, Javascript or XMLSchema. In its functionality it is also comparable to MS XAML technology which will be part of the upcoming Windows Vista [28].

XUL is mostly used for extensions of the Mozilla projects like Firefox or Thunderbird [30]. However, they may also be used to load applications in XUL capable browsers, just like, of course, Mozilla Firefox, e.g., there is a Mozilla Amazon Browser Application using XUL to create a rich Client UI for search book at www.amazon.com.

5.3.2 Programming Techniques

The top level element of a XUL file specifies the type of UI to crate, e.g., there is the `<window>` element for creating standard windows or the `<wizard>` element for creating application wizards. Below this element, form elements may be added like textboxes, buttons or list as well as images or progress meters. Additionally you may add XHTML elements just like in normal XHTML files, omitting a the `<head>` and `<body>` tags. XUL element may contain the attribute flex which causes them to grow if the application window is resized. Multiple flex values on different XUL elements define a relative growth regarding the elements, e.g., an element with flex=5 with an element flex=1 grows 5 times as much if a window is resized than the other element.

For positioning element in a window, a *box model* is used: Elements grouped in a box stay together and are aligned as specified: element under the box element `<valign>` are aligned vertically. Combined with box elements like `<grid>` which allows to layout element like in a table, complex but still flexible layout can be created. With the `<bulletinboard>` element, absolute positioning is also available which might be very important for many applications. There is also a Tree element in XUL, which is very convenient in many applications as well as menus or toolboxes.

EventHandlers coded in JavaScript may be added by either using special attributes on XUL elements like the *oncommand* attribute or by accessing the element with JavaScript directly and adding the EventHandler. XUL works very well with RDF (Resource Description Framework) [5] based data. RDF is a XML based Data Description Language which is build on the idea, that every resource element is defined by a resource, an attribute and object, which is called a RDF-triple. Originally, this kind of data storage and definition was created to match the subject-predicate-object pattern used in semantic languages. It is used in XUL to, e.g., fill

lists or trees. RDF is also used for saving the state of, e.g., a window locally in a file for later retrieval.

XUL applications may be stored in Mozilla Firefox chrome system, which lessens the security restrictions based on XUL files run in the browser. Although XUL files may be run using conventional syntax “http://...”, it is recommend, for security restriction reasons, to store them in the chrome system and reference them with a *chrome* URL like “chrome://...”.

5.3.3 Conclusion

Although XUL is very useful when creating Applications UI independently from programming languages, this is not important for creating a preview of a document. Nevertheless, XUL still allows to integrate XHTML and thus may be used as an extension of common HTML form elements. Especially menus may be very helpful in allowing more options to add flexibility and a good User Interface to a Interactive Document Preview.

XULs RDF support may be also exploited when using XUL, since this is the easiest way to integrate data in XUL. However, this might require to convert ones database which is most likely not build on the top of RDF.

5.4 SVG

5.4.1 Basic Concept and History

Scalable Vector Graphics (SVG) is a XML language for defining static and animated Vector Graphics. It was developed by the W3C and followed individual formats from various companies like the Vector Markup Language (VML) [27] from Microsoft or Precision Graphics Markup Language (PGML) from Adobe Systems. It is currently in its version 1.1.

SVG might be used instead of external image references within XHTML. However, SVG may also be used as a standalone language viewed in a browser if the browser supports it. This is not generally the case, SVG is still not supported widely in todays web browsers, e.g., MS Internet Explorer still requires a plug-in for SVG support, Mozilla Firefox supports SVG in its version 1.5 to some extent.

There is already a lot of tools support for SVG, including pure SVG creation tools like the Flame Project [39] . There is also some support in current image creation software like Corel Draw or Adobes Photoshop. There is also some programming language support for SVG, e.g. the Apache Batik [2] which allows to create, manipulate and view SVG in Java.

5.4.2 Programming Techniques

Since SVG describes vector graphics, SVGs basic elements are used to define geometric objects like `<rect>` for rectangles or `<ellipse>` for ellipses. All element include attributes to define, e.g., size, stroke and color. There is also the possibility to include external images or other SVG sources to a SVG image. Text can be easily included by the `<text>` element and can also be formatted as in CSS. The grouping element ¡g¿ is used to group elements for defining shared attributes and for better handling of a collection of objects. The `<defs>` element allows to predefine groups for later reference with `<use>` or to predefine more sophisticated color-filling by gradient elements. They allow to add a smooth transition from one color to another. Hyperlinks can be easily added by the anchor `<a>` element, much like in HTML.

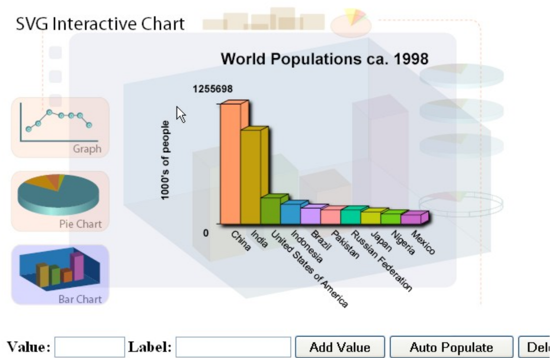

Figure 7: A SVG application

It is possible to manipulate SVG groups by using transformation elements like `<scale>` or `<rotate>`. Clipping can be defined very flexible by defining a `<clipPath>` with nested elements that define the clipping area. Very complex geometrical object can be created using the `<path>` element. By embedding Java Script functions into a SVG image, interactivity is added to the SVG image. Based on the DOM model, Java Script can be used to arbitrary change anything in the image and this might be also exploited to add AJAX techniques (see also section 5.1) to a SVG image. Another way to add animation capabilities to SVG is to use Synchronized Multimedia Integration Language (SMIL) [50], which makes it easier to create animations that would be very complicated to do with direct Java Script manipulation.

Although SVG is a sophisticated vector graphic description language, it was not designed to create a complete GUI for software applications, which is made obvious in the lack of form support in SVG itself. In the SVG version 1.2, this may change: the working draft for version 1.2 contains editable text fields. This would allow to create a complete SVG based widget library. However, by integrating SVG in HTML, form support can be added. Java Script is used to connect the actions taken on the HTML forms to the SVG image. Another idea is to integrate XForms elements into SVG. Although this is theoretically possible and supported by standards, at the moment there is no renderer support for the combination of both standards.

5.4.3 Conclusion

SVG is a very sophisticated tool to create dynamic websites. However, since it does not support forms directly, but has to rely on HTML, it is not easy to create an application that renders a preview of a document on the browser. This is made more difficult by the fact the todays browser do not support SVG fully. Current plugins work very well, e.g., the Adobe SVG Viewer [13] and may be used to create SVG-based web applications because the integration of HTML form elements works well.

[13] Adobe SVG Viewer: http://www.adobe.com/svg/viewer/install/main.html

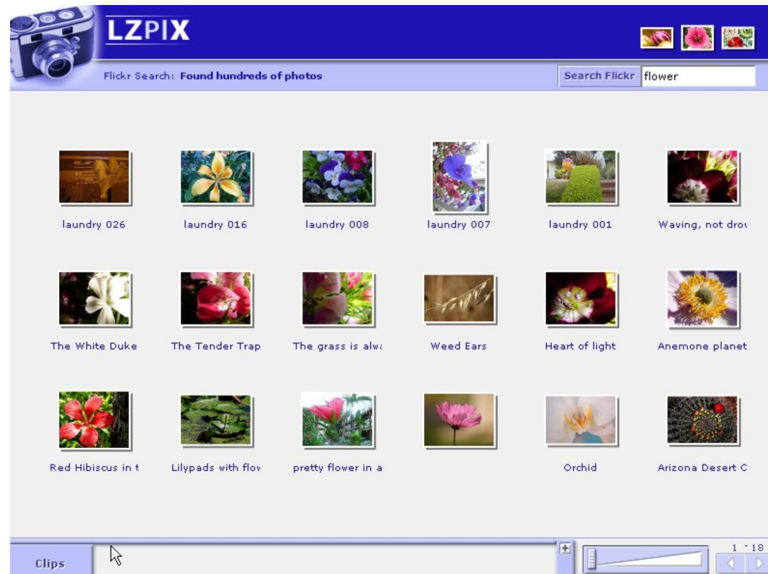

Figure 8: An example Open Laszlo Application

5.5 Open Laszlo

5.5.1 Basic Concept and History

Open Laszlo [24] is an Open Source platform for creating rich internet applications. It is currently in its version 3.3.3 and is available under the Common Public License [14]. It was first called the Laszlo Presentation Server (LPS) and its first release was in 2002. In 2005 its name was changed to Open Laszlo and the version 3.0 was released. It is implemented as a Servlet to be run in Tomcat version 5.0.

Open Laszlo is a high level UI description language based on XML. The XML document runs on the server and is compiled into Macromedia Flash (see also section 5.2). Future version will contain transformations into DHTML, as well as evolving technologies like possibly SVG 1.2. There is an IDE for creating Open Laszlo applications, the IDE4Laszlo. It is based on Eclipse and still under development, currently in the version 0.2.

5.5.2 Programming Techniques

Open Laszlo defines a set of XML elements for creating standard widgets: `<text>` creates simple text labels, `<window>` creates a window, `<button>` a button. A `<view>` element is used as a basic container for, e.g., images, videos or as a container for other components. There is a number of layout managers to arrange widgets in Open Laszlo comparable to layout managers in Java.

There is also an interesting "object orientation" in Open Laszlo: it is possible to create a "class" by the `<class>` element. It works like declaring a `<view>` but giving it a distinct name with the Name attribute. The class can be "instanced" by declaring the element directly with its name, e.g., the class "box" with the element `<box>`. Inheritance is also supported,

[14]Common Public License: http://www.opensource.org/licenses/cpl.php

the basic attributes of the parent class are accessible by the “parent” object used like a object in Java Script. Attributes and Methods may be defined which alter the characteristics of the object by referring to its attributes by the “this” object. Java Script events may also be declared directly in the class definition. There is also a Drag and Drop support. The scripting support in Open Laszlo makes it possible to create simple animations by using Event Listeners.

Since the application is rendered using Flash, the limitations and advantages of Flash apply here as well. Interesting is the fact that there will soon be a implementation of Laszlo that will use DHTML for rendering, making Open Laszlo independent of a Flash Viewer. In case of DHTML, AJAX (see also section 5.1) techniques will be used to minimize the response time of the application.

5.5.3 Conclusion

Open Laszlo is an interesting way to generate a document preview by using a high level description language but still being flexible with the rendering on the client. This would also negate the main disadvantage of Flash, being the dependency on a standard completely controlled by a single software vendor.

6 Presentation in custom Applications: Fat Client Solutions

6.1 OpenOffice.org XForms support

6.1.1 Basic Concept and History

OpenOffice.org is an Open Source Office suite comparable with Microsoft's Office Suite. It share similar programs like a word processor, a database or a spreadsheet program. Its data files are based on ODF (see also section 3.4), which makes it easy to open these files with other similar office programs. OpenOffice.org was originally created by Star Division under the name StarOffice. In 1999, Star Divisions was bought from Sun Microsystems, which published the source code and created OpenOffice.org based on the StarOffice code in 2000. The current version is available under the Lesser Gnu Public License (LGPL).

The current version of OpenOffice.org is 2.0 and it features a support for XForms (see also section 5.1). XForm elements may be incorporated in a standard OpenOffice.org Writer document. Writer is a Word processor just like MS Word. To create a document with XForm support, it is sufficient to create a special XForms document in Writer, which then allows to create document normally but incorporate XForm support. Writer is, just like MS Infopath, the design Tool for creating the XForm-based Forms as well as the XForm engine, used to fill out the form and submit it.

6.1.2 Programming Techniques

Writer allows to add buttons with actions or define handlers for various elements based on Macros written in OpenOffice.org Basic. Writer also supports some none-standard control elements like a Navigation bar for data records. Actually, Writer does not really support a XForms UI but only the back-end of the XForm standard, instead it reuses normal OpenOffice UI elements. This is especially obvious in the fact, that there is no `<repeat>` element in Writer. Working with the business logic part of XForms allows for client side data binding and data evaluation. The result data can be sent via HTTP protocol get, put or post.

Writers data format is completely based on XML and consists of various files. These files are stored in an .odt file which is a compressed archive. One of the files is the `content.xml` which contains the XForm data model and elements. Although there is a XHTML export for Writer documents, XForm elements or models are not exported, which would have been a very significant feature. Instance data for the form can be provided by a HTML link.

6.1.3 Conclusion

OpenOffice.org Writer allows to create forms that are based, at least on the file format and data binding, on XForms. Since Writers word processing capabilities allow to create sophisticated documents, it is capable to create Interactive Document Previews.

Writer's support for XForms can be compared to MS InfoPath, since they both try to accomplish the same thing: create a convenient Forms UI that allows some Client-side scripting, data evaluation etc. Although both accomplish the same goal, the main difference is that Writer is a full fletched Word Processor whereas InfoPath only contains basic Formatting elements. However, for creating Interactive Document Previews, this is still sufficient. InfoPath offers significant more options for connectivity, e.g., it supports database access and Webservices.

Instance data may also be derived of one of these interfaces. This makes it easier to integrated InfoPath in an existing workflow system.

6.2 Microsoft Infopath

6.2.1 Basic Concept and History

Microsoft Infopath is part of the well-known Microsoft Office Suite, the first time introduced in the Version of 2003. It converts user input by forms into XML and transfers the data to a backbone system. Infopath could be seen as an answer to two other recently evolving technologies, XForms and Adobes PDF Forms. These two technologies try to find a way for more intelligent and flexible Formular-based user interaction.

Infopath was first announced from Microsoft with the name *XDocs*, which was changed during development to its final name. With this technology, Microsoft wanted to find a more flexible way of distributing Forms to Users in Microsoft Office. Until then, Forms often were implemented in MS Access Databases, which force a very strict structuring of the Forms, or were based on flat files without any clearly defined structuring. Vonhoegen [49] concludes that often a middle-way is required, a semi-structured database which can be validated upon. The current XML technology XML Schema or Document Type Definitions (DTDs) allow for validation of XML data. Microsoft incorporated these standards in Infopath to allow easier validation upon data and compatibility to other technologies.

In MS Infopath, Users may do basically one of two things: they either create or fill User Forms. Since the Data structure of Infopath is not part of any standard, although it relies on some XML standards, Infopath has a proprietary file format, which basically may only be viewed directly with the Editor. If a Form is successfully filled, the result is a XML Document, whose structure was predefined. This Document may be “sent” either per email, with HTML, through a Webservice, stored within a database or by a self-defined script.

For one output definition, it is possible to create several views for different purposes. This allows for a better customization of Forms. Infopath files may be accessed in three ways: First, they might be stored on the local file system and simply opened with the Infopath application to fill the form; it may be stored on a Webserver in a virtual folder for retrieval; or it may be put on a Sharepoint Server.

6.2.2 Programming Techniques

Basically, working with MS Infopath is like working with one of the other MS Office Products or other Word-Processing Applications. It features the well-known features like Spell Checks, Formatting options tables etc. Thus for many Users proficient with the standard MS Office Applications, Infopath is easy to work with.

The starting point for creating a Infopath Document is defining an structure for the result document. Although this may also be done in the Editor itself, more commonly it will be supplied by a XML Schema, through a Webservice or a Database Schema. Out of this data definition, Infopath allows to choose several commonly known widgets like textboxes or extensible lists to be bound on the data.

Infopath allows the inclusion of JScript or VBScript for defining Action Handlers, e.g., for data validation. What is missing in Infopath is the possibility to add .NET-based modules to Infopath Documents, since this would allow a good integration with existing technologies.

Infopath stores their documents with the extension .xsn. This is actually a .cab packed file package, which may be extracted in Infopath by a menu option, or by simple renaming the file and manually extracting it. The result is a couple of files:

manifest.xsf: The manifest file includes information about the files and components included in the .xsn file that has been extracted; It can be used to add symbols or menu items to Infopath when filling the form, for defining error messages, adding script files or adding printing settings; It is based on XML and follows a proprietary definition;

.xml: This document contains the sample data used for pre-filling forms when a Infopath Document is opened for filling out a Form; The XML document follows the predefined document layout;

.xsd: This file contains the XML Schema definition of the result document;

.xsl: Various XSLT files may be contained in the .xsn file for presentation and transformation purposes; they may convert, e.g., the sample data to a html-based result document with added CSS style information, which is used to define the formatting of the form

.htm, .gif, ...: HTML and image files are included in the file format for user-defined UI elements like pictures;

.js: These contain the scripts used by the Infopah Form; they contain business logic as well as data validation or action handlers;

When manipulating these files directly, one has to consider some limitations of Infopath, e.g., Infopath cannot work with every kind of XSD, because it needs some kind of unambiguous definition: if there is an “any” element in the XSD, the attribute `minOccures=''0''` has to be used.

6.2.3 Formatting

InfoPath offers some formatting options which makes it possible to create a Document-like Form, as shown in Figure 6. It was created by using various tables for positioning, like in HTML. InfoPath does not allow to configure CSS styling directly in the InfoPath Application, although a CSS file can be extracted out of the .xsn file. By changing the CSS file and re-compressing the file, the file is correctly changed, so this is an options to modify the file dynamically. However, this is a very awkward way to do this, decompressing and compressing a file is not very performant.

6.2.4 Conclusion

Infopath is Microsoft’s solution for a Front-End for an underlying XML-based workflow process. Its most remarkable feature is its easy-to-use Forms Designer, so that no special training is necessary to create a From based on an underlying, e.g., Web service. Since many Workflow Engines support communication via Web services, this alone makes it possible to integrate InfoPath into an arbitrary workflow.

Still, InfoPath is a proprietary software solution and, although also available as a standalone software, is mostly tied to MS Office and, of course, on the windows platform, which may

make it unattractive for many solutions. Although InfoPath integrates scripting support, there are tight limits for InfoPaths extendability: no .NET source code integration or VBA support, although this is common for other MS Office products.

7 Example Solution: Interactive Document Previews based on XSL-FO and XHTML/XForms

7.1 Introduction

As the previous sections of this thesis have show, there are many ways to create a document preview on the top of a Document Workflow. The following chapter describes an example implementation of one of these concepts. The implementation consists of two basic parts: the extension of an existing XSL-FO editor for creating similar XHTML documents and an implementation of a workflow which deals with creating an interactive document preview based on a given data input and XSLT transformation files. It has to be noted that the only the extension of the editor will be used in a productive environment whereas the workflow implementation is a proof of concept on a chosen architecture for implementing a document workflow with an Interactive Document Preview as discussed in this thesis. The resulting workflow implementation does not have production quality, since it should only give an impression of the obstacles that may arise and the benefits that can be made obvious.

The author of the thesis implemented an extension to an editor which allows to create a XHTML based document having the same appearance in a browser as the XSL-FO document when transformed into PDF document. This is achieved by parsing the internal object model used within the editor representing the designed document. This model is fully aimed at creating XSL-FO content, making it a non-trivial task to transform this model to a XHTML based object model which is in turn transformed into a XHTML document. This document is designed to act as an interactive document preview of the XSL-FO document designed in the editor.

Since a significant idea of the discussed preview is to make it interactive by adding facilities to change its data content, XForms elements are embedded into the XHTML document. The abilities of XForms allow the XHTML document to have, e.g., standardized extensible lists. Unfortunately, only a few browsers support XForms natively, so it is necessary to provide a server-side solution for Xforms. This is achieved by including the Chiba Project web platform [37] into the project.

These components, XML data, an XSL-FO, an XHTML transformation stylesheet and an Xforms processor, allow to create a workflow that creates, based on XML data, a document which may be previewed in any browser and then, when accepted, be transformed into a XSL-FO document. This simple workflow can easily be implemented on any Workflow Engine, without taking in account the problems arising the significant amount of user interaction. The result, the PDF document, may be viewed and printed in any PDF viewer.

7.2 Architecture and Design Overview

7.2.1 Extension of the XSL-FO Editor

The XSL-FO Editor used as a platform for the creation of XSL-FO and XHTML documents is a significant part of the effort of this implementation, as the transformation of a XSL-FO based model to a XHTML model requires some effort. The editor is under development of Qualisoft [15], an IT service provider with locations in, e.g., Germany and Austria. It uses Eclipse

[15]Qualisoft: www.qualysoft.com

as an underlying framework.

Creating a XSL-FO document that incorporates XML data in an editor is naturally implemented by using an appropriate XSLT model. Since the user of the Editor has to provide the XML file which will be incorporated into the resulting document, the editor allows to take data fields from the XML document and incorporate them in the graphically designed XSL-FO document. The editor enables, e.g., to insert `<xsl:value-of>` and `<xsl:for-each>` elements into the document.

XSL-FO is thus generated by first parsing a created XSLT model to a XSL-FO document. This has the effect that the XML data is already hard-wired into the XSL-FO document, making it hard to create an Interactive Preview Document out of this document, but this is one of the goals of this implementation. Due to that, the transformation step to XHTML must not be based on the resulting XSL-FO, but rather on the XSLT document that ought to generate the XSL-FO, since the information where the XML content is coming from is still included, and how it has to be applied as well.

The editor has an internal model of the graphical objects that form the XSL-FO document. The extension provided by the implementation of the author is based on this model. It converts the model in a similar XHTML based model which can in turn be transformed in a concrete XHTML file. Additionally, XSLT objects are also part of this model which has to be taken into account as well.

The author of the thesis was involved in implementing parts of the editor that are concerned with the actions described above. The author did not implement the changes that have been necessary to include the created classes into the editor. The GUI of the editor which is created by using eclipse with GEF plugins have not been changed by the author of the thesis, nor was it necessary to change the inherent logic of the editor.

Figure 9 shows a class chart of the implementation of the author. The class charts shows the basic classes created by the author. Classes set in red are classes that already existed in the editor implementation. There are several important interfaces that are used by the implementation of the author.

There are some important classes which the extension of the editor had to work with:

InfinicaDocument: This class represents a document within the editor. It contains the generic data model that is used with the editor to represent the data structure of the designed XSL-FO document. It also handles the parsing of the document into several file formats by using various renderers. The supported file formats include XSL-FO, PDF, RTF and, with the extension implemented by the author of the thesis, XHTML.

Generic Model Classes: The GUI works on an underlying data model of the designed XSL-FO document. This object model is derived from the XSL-FO standard. It contains classes like *Block*, or *Table* for representing the appropriate XSL-FO elements and contain all styling information that are necessary for the a later conversion to a complete XSL-FO document. There are a lot of classes in this model, some of which can be transformed into appropriate XHTML elements, and some of which cannot be converted. The model consists of elements that are based on similar XSL-FO elements as well as XSLT elements.

ReflectiveVisitor: Some central classes of the author's implementation use this class which implements a generic visitor pattern to be used with the Generic Model Classes. This

Figure 9: Class Chart of the Extension of the editor

implementation uses the Java Reflection capabilities to visit the elements of the traversed object model.

XMLReader: For rendering the created object model with Saxon, the XMLReader interface is used so that the resulting object model created during transformation of XSL-FO to XHTML can be rendered into a XML document.

An important part of the implementation is the conversion from the existing Generic Model Classes to a model that is based on XHTML and XForms. The extension includes the XHTML Model Classes representing XHTML elements and the Xforms Model Classes for XForms elements. The classes for XHTML contain common XHTML elements like *Div* or *Img*. These classes are very light and do not contain much information besides a set of attributes that are part of the represented XHTML element and some utility functions. Some classes do not consist of much more than a constructor. All of these classes may form a tree model that represents a common XHTML document tree, on the top being an *HTML* object followed by a the child elements *Body* and *Head* etc. .

An important step is the conversion from this XSL-FO based model to this new XHTML model. This is realized by using the Visitor design pattern: the class XhtmlStylesheetVisitor takes as a basic input the root element of the editor's model and iterates through all of the elements in this tree model. Meanwhile the XHTML model is created by inserting an appropriate XHTML element for every XSL-FO element that is traversed. Some elements can easily be represented by a single XHTML element, some need a more sophisticated structure in XHTML, as is described in the coming sections.

Beside having a XHTML model, there was also the need to have a special class that represents a page in the XSL-FO model, the PageXhtml class. This class contains logic that transforms the given information from the PageMaster elements of XSL-FO to a corresponding XHTML table element. This is the reason why this class inherits from the XhtmlTable class.

The Xforms Model Classes contain classes representing the XForms model. One tricky part of the implementation was to somehow incorporate the coming XML data information which has to be used by XForms elements into this model. There are basically two methods to do this: either create a URI reference to the XML document for later retrieval, or incorporate the whole XML document into the XForms `<xf:instance>` element. The latter was chosen since there is the possibility to create XSLT elements from the Generic Model Classes as children in the class representing the `<xf:instance>` element. The XSLT model was created in such way that it creates a identity transformation on this place of the document from the parsed XML document in the transformation step that takes place after the XHTML model is created.

The result of the XhtmlStylesheetVisitor is a XSLStylesheet class. This class represents a XSLT document. This might not be intuitive, since we just transformed the Generic Model Classes into an XHTML document. However, there is another transformation step taking place in which the XSLT elements that still exists in the XHTML model are transformed and then rendered into the resulting XHTML file. Since the XSLStylesheet class is also integrated into the rest of the Editors environment, it is also a very good way to insure as little intrusion into the existing editor as possible. The author's efforts in this direction have been successful: the whole implementation could easily be integrated into the editor with only changes in the GUI to add the new possible actions realized by the extension.

The transformation from the XHTML model to a XHTML file was done by using Saxon, the implementation of this step is placed in the SaxXhtmlVisitor which is driven by the SaxXhtm-

lVisitorReader. Here again the Visitor design pattern was used to traverse the XHTML model tree to form the XHTML document. This was simple since the information is nearly completely contained in the model. Only some conversion in attributes are done in this step, mainly eliminating XSL-FO attributes which were taken from the XSL-FO model which are not supported in XHTML. Additionally, this Visitor added information to the XHTML model that can be parametrized, e.g., the destination for the submission element of XForms. The Visitor can also be used to create two different results: either a document that is a complete XHTML document or an XSLT document which still contains some XSLT tags, for, e.g., parsing it with a given XML data file. This has two reasons: while using the Editor, it is important to preview the document and see a complete XHTML result given a XML data input. On the other hand, for integrating the XHTML preview in a workflow, it is important to a XSLT document that is not already parametrized with a XML document.

The parsing of the XHTML model into a concrete file happens in the XHTMLRenderer. It starts the rendering process using the standard XML transformation facilities included in Java.

The most difficult part to implement was the generation of paged layout on a non-paged document format that is XHTML. The approach of using simple tables to get a feeling of a page proofed to be sufficient most of the time. However, there is no easy possibility to incorporate a correct behavior of extensible list of XForms into this paged layout. These lists may be arbitrary long, and if this length would generate a new page in XSL-FO, this is not reflected in the current implementation. However, the result, although being not equal to the resulting XSL-FO, is good enough to get a feeling how the result document will look like.

Since the editor is still in development, the author of the thesis used a test environment for creating the discussed XSL-FO object model from scratch and using this model to test the XHTML implementation. This worked very well and the result after integrating the XHTML implementation into the editor proved to be very good.

7.2.2 Workflow Definition and Web Application Architecture

As already stated in the previous section, there are some main components that need to be addressed and connected. Figure 10 shows a simple flowchart representing the components and their interaction. Figure 11 shows the same workflow in BPMN notation (see section 3.3.4). The main components of this simple workflow are the following:

XML Data: The document that shall be the result of the implemented workflow is based on a set of customer data. This data is accessible as a file on the webserver. Since it is based on XML, it can easily be transformed by XSLT processors.

XSLT stylesheet for XSL-FO transformation: One of the two stylesheet resource in this workflow is used to transform the XML data into a XSL-FO representation of the result PDF document. This document only uses XSLT 1.0 elements, although the used XSLT processor supports XSLT 2.0 as well.

XSLT stylesheet for XHTML transformation: The second stylesheet is responsible for creating the XHTML representation of the result document. The document uses CSS style properties to define the presentation. Additionally, there are embedded XForms elements.

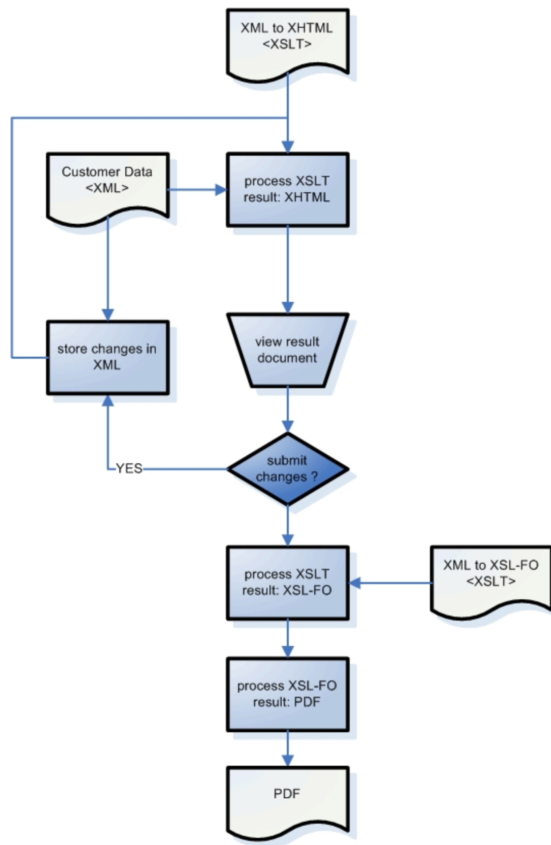

Figure 10: Simple Flowchart of the Workflow

Transformation engine: The XSLT transformation engine used in this implementation is Saxon version 8.7.3 [21]. Saxon is an Open Source XSLT processor capable of parsing XSLT 2.0, XPath 2.0 and XQuery 1.0. It is written in Java, although there also exists a .NET port.

Figure 12 shows the Layer Model of the implementation: Build upon the Operating System, the Tomcat Servlet Engine is used [12]. This Apache product is available by the Apache License and is widely used and supported in the Open Source Community as well as in commercial products. The version Tomcat 4.1 is used due to compatibility issues with Chiba. As a Servlet Engine, Tomcat is used to run several Servlets necessary for the workflow to work. Axis [11] is an implementation for supporting the development and distribution of Webservices. It is also a Apache Foundation project and available under the Apache License. It comes with a very useful tool for generating WSDL stubs for arbitrary Java classes and makes Webservices available. All activities of the discussed workflow will be accessible by a Webservice interface and will use Axis to achieve this.

The second Servlet running in Tomcat is BPWS4J (see also section 4.4). It is the Workflow Engine that runs the workflow. As it is based on BPEL, there is also a workflow description file that defines the concrete workflow like in figure 11. BPWS4J was chosen due to its free availability and being an implementation of probably the most supported Workflow Definition Language.

The last Servlet used in this implementation is Chiba which is one of several Server-Side implementations for XForms. Although XForms is a highly useful tool for creating web forms or wizards, it still lacks support from nearly every browser. Although there are several plugins available for Internet Explorer or Mozilla Firefox, these do not offer the flexibility of a server implementation. In this implementation the Chiba version 1.0.0 is used which does not provide the significant bug fixing as well as AJAX support compared with the coming version 2.0.0. Although there is already a release candidate for this version, it proofed to be not stable enough for this implementation.

Chiba consists of several modules, the most important being the core library and the Servlets. The library handles the transformation of the embedded XForms code in the XHTML file to simple XHTML and Javascript. Most of the transformation process is handled by XSLT stylesheets which can be easily edited to serve the specific needs of the project it is part of. For this implementation it was not necessary to change the XSLT files of Chiba. Although the core library may be used without the corresponding Servlet, the authors choose to use it to simplify the implementation.

The first step in implementation was creating the framework in which the preview should run. The workflow is based on BPEL, so the first step was creating the appropriate Web Services. The implementation of these was straight-forward since the single actions are clearly defined and do not need much effort.

The CreateXhtml webservice creates the XHTML document from the given XML data supplied by the user as well as the XSLT document for transforming the XML data to the preview. The URI of the result document is passed as a result of the webservice. Then the user is presented the resulting XHTML document which is an Interactive Document Preview. The document is placed in a directory within the Chiba implementation. This results into the document being parsed by the Chiba Servlets which transforms the XForms elements into XHTML elements.

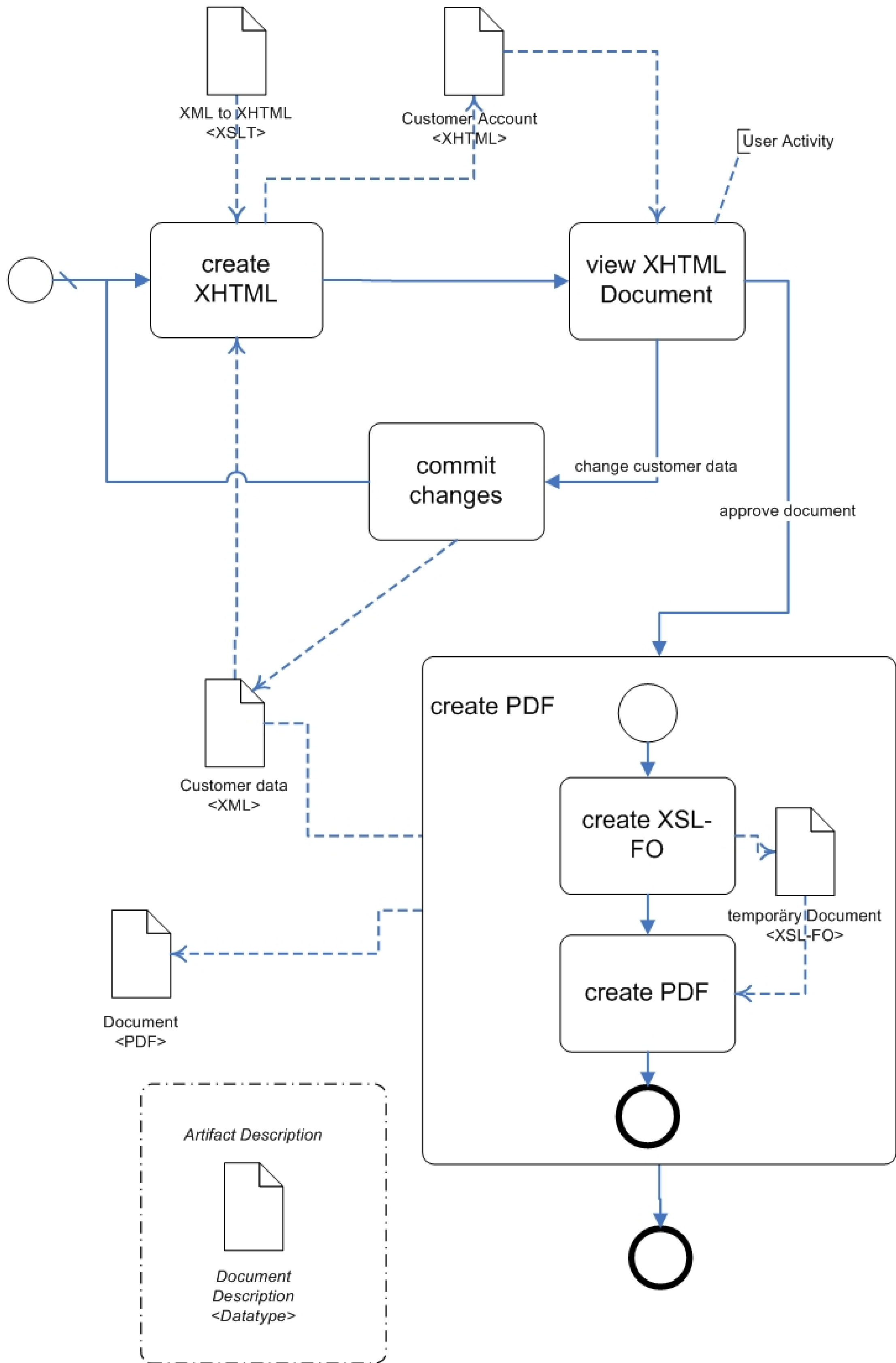

Figure 11: BPMN chart of the Workflow

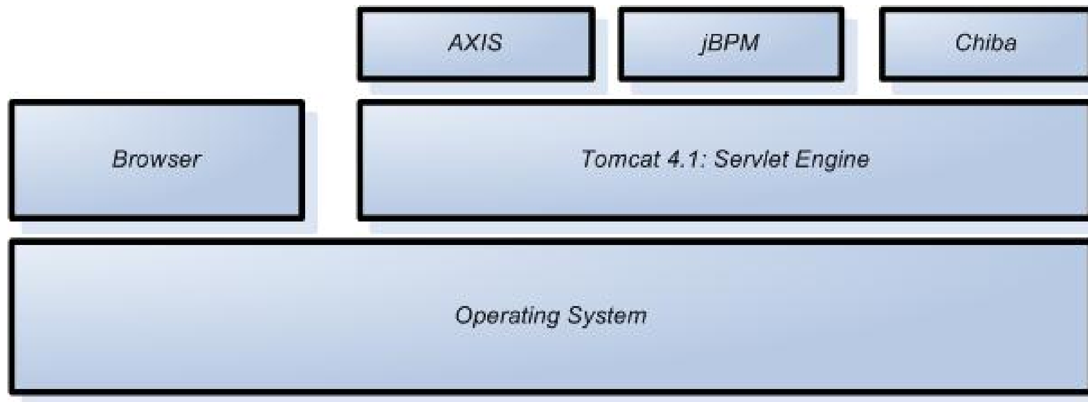

Figure 12: Layer Model

One advantage of using Chiba is that after pushing the XForms submit button, the XML data is extracted from the shown document and sent to whichever address is specified, continuing the workflow using Servlets. Since the Chiba project is still in development and not 100 % stable and bug free, there were some isues in integrating the document in the workflow. Since there is still a bug in Chiba make it not possible to redirect the HTTP request to another website also parsed by Chiba, the author had to create another website which only contains a link to the newly created interactive preview containing the changes the user did on the interactive preview.

For comparing the result of the document preview, the possibly changed XML data and the formerly used XML data, the XMLUnit [16] library available at sourceforge.net was used, since it provides an easy comparison mechanism for XML files. The result of the webservice is either true or false, either resulting in a new creation step of the XHTML document or creating the final PDF document. In the first case, the changes are first stored, replacing the old input document from the user with the new XML document containing the changes from the action on the interactive document preview shown to the user. If the latter case, the document URI reference is sent to the CreatePdf Service which transforms the XML data into a PDF document similar to the interactive document preview, finishing the workflow.

Creating the WSDL interface definition was done by using the AXIS tools that are available for Apache Ant [17]. These tools create a WSDL file for an existing Java class. This WSDL file is in turn used to automatically create classes which make it possible to call these webservices for testing purposes. Deploying the Web Services in Axis is very easy, since this can be done by adding the Java class to the root folder of the Axis web application directory in the tomcat servlet container and renaming it with a .jws extension. The Java file is than compiled by Axis and made public as a Webservice.

The BPEL workflow engine BPWS4J also works with AXIS for providing a Webservice interface for the workflows that run with it. The necessary BPEL file with a corresponding WSDL file for its interface was written by the author without use of tools, mainly working with the example implementations that are provided with BPWS4J. Setting up the BPEL-driven workflow engine BPWS4J was a task of trial and error due to the lack of good documentation.

[16] XMLUnit: http://xmlunit.sourceforge.net
[17] Apache Ant: http://ant.apache.org/

It is very hard to find errors in a BPEL definition file since the engine does not provide any meaningful exceptions, but since the BPWS4J is more a proof of concept rather than a serious business implementation, this should have been expected.

It was not that easy to incorporate the workflow engine in a way that easily controls the flow of pages that should be displayed in the browser. Servlets were used to do this, these Servlets were triggered by the Webservices and themselves signaled the Webservices when the user was finished with his input, so that the workflow could continue. However, in fact this implementation removed the user interaction activities completely from the BPEL workflow definition, although these are valuable parts of the workflow. Reflecting on the finished implementation, this way of handling user interaction was seriously flawed and very static. As discussed in chapter 3.3.1, BPEL is specified with user interaction in mind, and this is shown by this implementation. It would have been better to either use another workflow engine with a better user interaction support or to find an early implementation of the BPEL workflow language with the BPEL4People extension.

7.3 Transformation XSL-FO to XHTML and XForms

One significant part of the implementation is the transformation of a XSL-FO based model to a XHTML based model, including additional XSLT objects, thus one research topic was how to perform a conversion from XSL-FO to XHTML. This is no easy task since there are several differences between XHTML and XForms, which are mostly based on the fact that XSL-FO describes one or a series of pages, while XHTML does not support the concept of a page. XHTML is viewed in a Browser, and therefore the content may become arbitrary broad and long. In contrast, XSL-FO has to take page definitions into account. This makes bringing the page concept to XHTML an important topic. For this implementation, a very simple approach was chosen: the page layout is created through a series of tables, each one defining a page, each with specific cells of the table referring to a region in a XSL-FO document. Since using `<table>` elements for layout is not recommended, future implementation may use CSS styling mechanisms to simulate a page layout in XHTML. The difficulty in generating page layout in XHTML is that it is hard to figure out when a new page shall be generated instead of adding content to the current table cell corresponding to the XSL-FO `region-body`. The author of this implementation uses a simple approach for supporting it: it requires an explicit setting in XSL-FO to generate a new page by using the block-attribute “break-before”. If this attribute is encountered in a `fo:block` element, a new page is generated. Although this is a very restrictive requirement for XSL-FO documents, it is an simple approach which is easy to implement and does not require a complicate mechanism to calculate when a page is full.

Figure 13, Figure 14 and Figure 15 try to summarize the transformation process from XSL-FO to XHTML. The overview follows no standardized set of rules but tries to go step by step through the transformation process, beginning from elements that are included into the XHTML file all the time to specific transformation rules. Although one cannot exactly follow the description to make a working parser since it does not contain all details that have to be taken into account, it is sufficient to handle the basic conversion mechanism.

7.3.1 basic elements of the transformation overview

The main idea of the overview is, that starting from the root element of the to-be-translated XSL-FO document, the reader follows the *named arrows* by the XSL-FO element they encounter. The exception are the page defining elements, that is the `<fo:layout-master-set>` and all its children. These elements cannot be translated directly into XHTML elements since there is no meaningful equivalent in XHTML. Instead, the information contained in this elements is used throughout the document, either by defining an XHTML attribute or by influencing the layout defined by XHTML tables. Arrows that have no name express elements that follow in sequence to the previous element, rather then as a child, which is the default.

The arrows lead to *rectangular boxes*, which describe XHTML elements that have to be inserted into the XHTML tree. For example, starting with the arrow named with the element `<fo:root>`, the reader of the overview can follow the arrow to the box describing the `<html>` tag. Arrows that are named "is child" must be followed, the following box contains elements that must be the child of the previous box. This is necessary for XHTML elements that need to be part of the translation but are not directly the result of a given XSL-FO element.

A third central element of the overview are attributes, which are described in text next to a box, which is indicated by the vertical line beside it. The text follows a certain syntax: the XHTML elements attribute is named and followed by a semicolon, after that the corresponding XSL-FO element is named, followed by a "@" character and the specific attribute to that element. A second case is that the named XHTML element is proceeded by a "@" character as well. In this case, the attribute is taken directly with its name into the corresponding element, whereas all other attributes have to be part of the elements *style* attribute.

The XHTML style attribute sums up all CSS attributes a XHTML element may contain. Although there are a lot of attributes that can be written directly with its name into the element, like the XHTML `<table>` attribute "width", it is easier to generally treat the attributes as CSS style attributes. However, there are exceptions, in which this is not possible. In this case the notation with the leading "@" was chosen.

Although the similarities of the style attributes between XSL-FO and XHTML allow to bring the values of most XSL-FO attributes directly to the XHTML attributes, in some cases a calculation has to done to get meaningful values for XHTML. Here the calculation is provided instead of only the corresponding XSL-FO attributes name. A special case are the `<fo:block>` and nested element's *Formatting-attributes*, which are described later in this thesis in a list. They are too numerous to be written in the overview and are just referenced. However, mostly these attributes can be taken directly into XHTML elements without further conversion, if they are supported at all. At last, the attribute text may contain informal layout information for table elements or condition expressions for certain elements.

Another element of the overview are *dashed boxes* which are a reference to already described XSL-FO elements. The meaning behind this reference is that the reader has to look up the arrow which names the specified XSL-FO element and has to follow it from there.

The last element of the overview is the diamond shaped element which expresses a condition or a loop over many possible children of the previous element. For example, one of these shapes has the following text content: "Content of TableCell which (fo:flow or fo:static-content)@flow-name matches @region-name of region element;" This means that the following elements are the content of the specific TableCell that this shape follows. Since more than

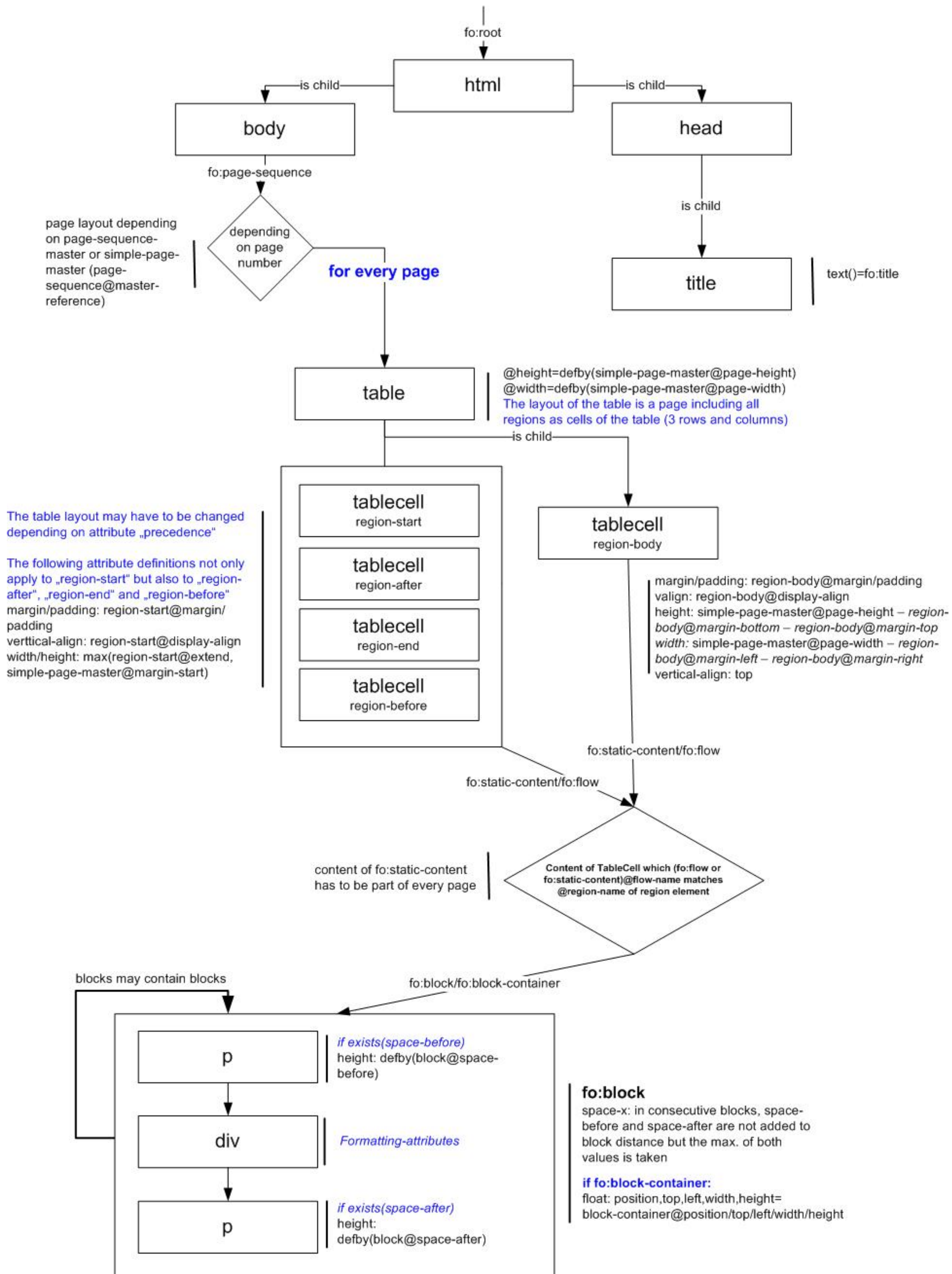

Figure 13: Overview Conversion XSL-FO to XHTML: page construction and fo:block/fo:block-container

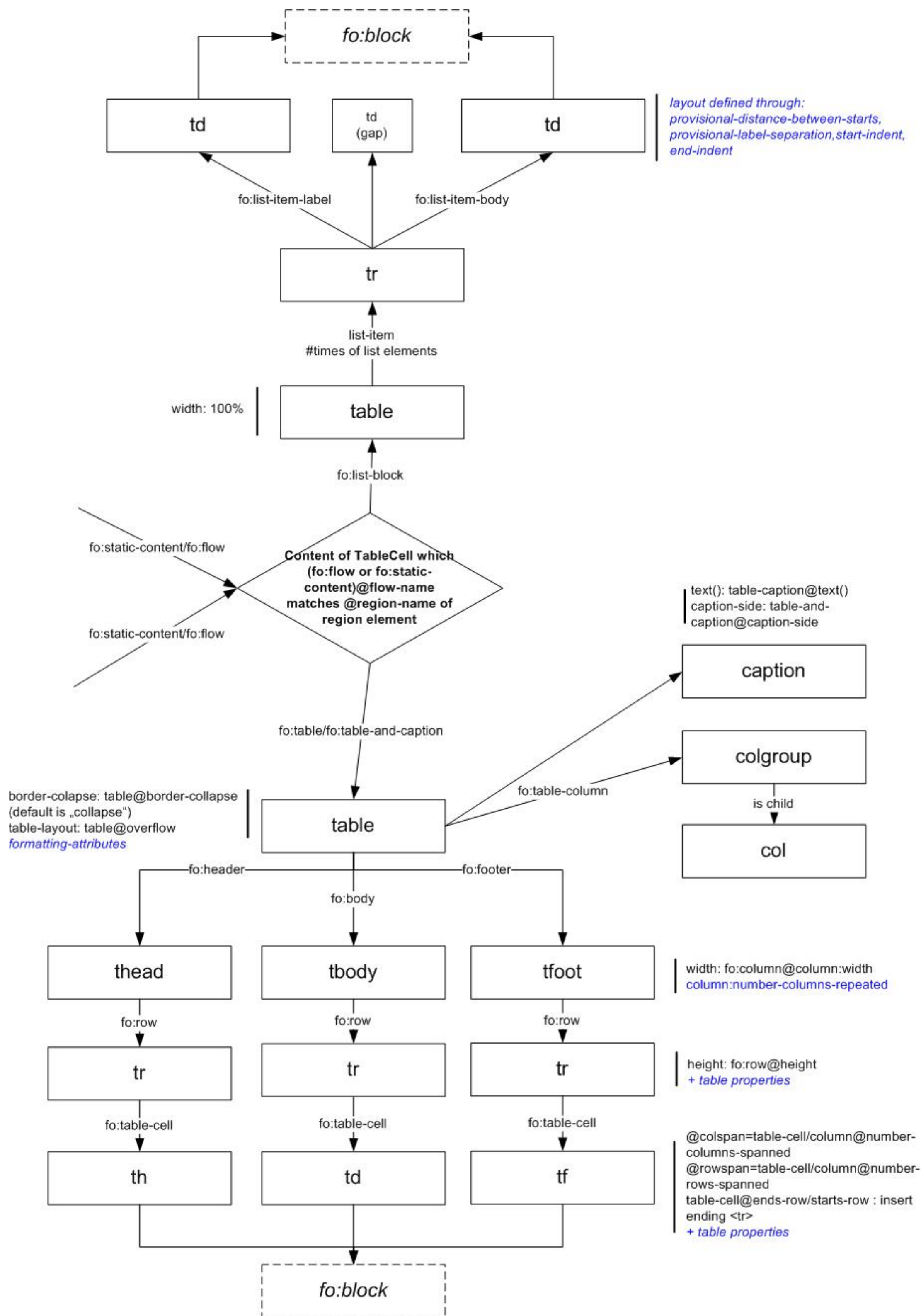

Figure 14: Overview Conversion XSL-FO to XHTML: fo:table and fo:list-block

one arrow leads to the shape, the TableCell has to be populated that represents the content of the region which is defined by the “flow-name” attribute of the previous `<fo:flow>` or `<fo:static-content>` element.

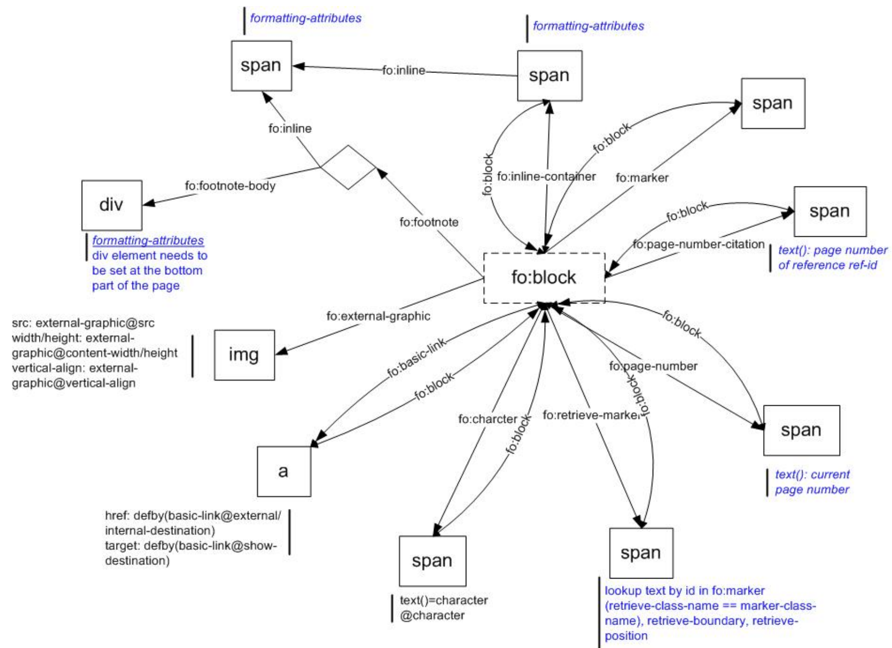

Figure 15: Overview Conversion XSL-FO to XHTML: inline elements

background	border
clear	clip
color	column-width
font-style	font-variant
font-weight	font-size
font-family	font-stretch
letter-spacing	line-height
margin-x	max-height
max-width	min-height
min-width	padding-x
text-decoration	text-indent
text-shadow	text-transform
white-space	word-spacing
visibility	text-align

Table 1: Formatting Attributes — supported:

7.3.2 Attribute and elements support

Since XSL-FO is a paged media, some of its layout information are more sophisticated than in XHTML, e.g., no color profile information can be provided in XHTML. This information is only important when the rendered document document needs to be printed. Another set of attributes that cannot be transfered to XHTML are hyphenation rules since hyphenation is not supported in XHTML.

Due to this, a large number of styling attributes cannot be converted from XSL-FO to XHTML. Table 1 tries to sum up which formatting attributes can be supported by modern browsers (no more than two years old). Table 2 sums up the formatting attributes that cannot be supported in XHTML. and which have a differently named equivalents in XHTML. These attributes are typically part of XSL-FO block or inline elements.

Additionally, there are XSL-FO elements that do not specifically target paged layout and content like attributes for voice output, e.g. the attribute “role”. There are also some attributes that work in specific elements, but not in others, Table 3 gives an overview about these attributes. At last, there are XSL-FO elements that can only be converted to XHTML with a lot of computational effort or simple have no equivalent. These are the following:

color-profile: It is not possible to define a color-profile for XHTML since this is no issue for content displayed on a computer.

declarations: This element is also used for defining color-profiles and is not supported.

initial-property-set: This element defines the formatting properties of the first line of a block element. Since it is very hard to find out in HTML exactly were a line ends when rendered in the browser, it is not supported in the author’s implementation.

instream-foreign-object: Although there is no equivalent in XHTML to the instream-foreign-object, it might be possible that the browser may be able to render its content, e.g., if its content is a SVG tree. The author’s implementation ignores this element to prohibit rendering issues with different browsers, however, by checking the namespace

alignment-adjust	alignment-baseline
baseline-shift	dominant-baseline
border-start-precedence	border-end-precedence
border-after-precedence	border-before-precedence
end-indent	font-selection-strategy
font-size-adjust	hyphenate
hyphenation-after	hyphenation-before
hyphenation-start	hyphenation-end
keep-after	keep-before
keep-start	keep-end
last-line-end-indent	linefeed-treatment
line-height-shift-adjustment	line-stacking-strategy
position(only with float)	relative-position
script	space-start
space-end	text-align-last
text-altitude	text-depth
treat-as-word-space	white-space-collapse
white-space-treatment	widows
wrap-option	block-progression-dimension
overflow (only with float)	color-profile-name
country	language
inline-progression-dimension	reference-orientation
writing-mode	score-spaces

Table 2: Formatting Attributes — not supported:

XSL-FO element	Attribute name
external-graphic	scaling, scaling-method
basic-link	destination-placement-offset, indicate-destination,
region-body	column-count, column-gap
table-cell	empty-cells (only in xhtml:table supported)
character	glyph-orientation-vertical, glyph-orientation-horizontal, suppress-at-line-break
root	media-usage, source-document
table	table-omit-header-at-break, column-number
table-cell	relative-align
list-item	relative-align
block-container	absolute-positioning
region-body, region-start region-end, region-before, region-after	reference-orientation
block	span

Table 3: Not supported attributes in specific elements

of the embedded content, an implementation may choose whether to support the foreign object and include it in the result XHTML document, or not.

leader: This element is difficult to simulate in XHTML since it is generally hard to find out how many characters can still be inserted in a text block until the end of line is reached.

multi-case, multi-properties,... : There are some elements in XSL-FO which allow to switch properties or property sets within the XSL-FO document by checking against conditions. Although this logic may be simulated by JScript code, these elements are rarely used and the author did not include them in this implementation. The elements that are not supported are: multi-case, multi-properties, multi-property-set, multi-switch, multi-toggle and wrapper.

7.3.3 XForms elements

Since an Interactive Document Preview is, of course, about interactivity, the conversion from XSL-FO to XHTML has to take a look at XSLT information that is part of the information we work upon. This implementation uses XForms elements to insert textfields and lists into the XHTML document. This is feasible because the implementation uses Chiba to transform XForms elements to equivalent XHTML and JScript. Thus, using XForms is not strictly necessary for creating such a document preview, but it simplifies the implementation without negative side effects.

For data binding and further processing information, XForms uses the `<xf:model>` element and its child elements. The implementation has to provide this information:

`<xf:instance>`: This element of XForms is used to reference the data this document is based on. Generally there are two ways to accomplish this, by either referring the XML

document by a proper URI, or by including the data directly as a child subtree of this element. In this implementation, the latter is chosen since the data is fixed in the moment of creating the document. As the document XML data is transformed with a XSLT stylesheet, it is easy to incorporate the whole XML content in the result XHTML document.

<xsl:bind>: XForms allows to specify data binding more precisely by using this element: A node in the XML file may be referenced by a unique name and the data type can be specified, allowing for a build-in data verification. This is not used in this implementation, since this would mean to parse an additional file, the corresponding DTD or XMLSchema file to the XML data and incorporate the result in the XHTML document which is a complicate mechanism and not in the scope of this implementation.

<xsl:submittion>: This element specifies the URL for processing the XHTML document after the preview has been changed or been agreed to. This element has to be used by the implementation, the necessary information is specified during creation in the editor.

Two XSLT elements are important: the `<xsl:value-of>` and the `<xsl:for-each>` element. `<xsl:value-of>` parses concrete values from the XML input file into the XSL-FO document, thus customizing the document. For this kind of information, it is rather easy to create a suitable XForms element: the `<xf:input>` field already serves this purpose. The content of the “select” attribute of the `<xsl:value-of>` element is moved to the “ref” attribute. Additionally, the data instance information has to be added to the attribute.

The `<xsl:for-each>` is more difficult to interpret. In this implementation, the for-each construct is interpreted by a `<xf:group>`, an nested `<xf:repeat>` and two buttons for adding elements to the created list. This converts the for-each element into an expandable list in XHTML. This works fine since we can extract the information about the root data element the for-each works upon and insert it in the corresponding `<xf:repeat>` attribute. The child element of `<xsl:for-each>` will be parsed as children of the `<xf:repeat>`, so that Chiba will iterate over the elements in the XML data and create the same result as the XSLT transformation. Of course, since XSLT features many additional elements like conditional expressions and switches, there may be cases where this transformation does not lead to a satisfying result, but in many cases it will suffice. It is a topic for future implementations to try to incorporate other XSLT elements, e.g., by using JScript.

8 Future Work

This thesis tries to give its reader a good overview about the existing technologies for workflows and possibilities for creating an interactive preview. Future research may be done by performing example implementations using different workflow languages as well as other technologies for displaying such a document. Thin Client solutions using, e.g., Flash can certainly be used to create rich and very interactive documents. A comparison with the author's example implementation could give valuable information about the effort and the result that is associated with such architectures.

An interesting idea is to evaluate the possibilities that browsers offer for printing. With good browser printing facilities, it would be possible to print documents directly from the browser, making it easy to convert the preview document featuring possibly many text fields to a static version that can be printed directly.

Future implementation may also try to incorporate a more sophisticated transformation for XSLT elements. For example, the example implementation ignores some flow-control elements like `<xsl:choose>`. These elements may either be implemented by using JScript constructs that handle the control-flow or by using the XSL-FO `<fo:switch>` element. It should also be analyzed in which cases this is feasible and necessary, and in which cases it is best to let this control flow be handled directly by the XSLT renderer.

9 Conclusion

Interactive Previews are a possibility to provide a proof-view of generated documents of a Document Workflow. Since there are many possible Document workflow implementations as well as possibilities for creating such a preview, careful evaluation is necessary to find just the right technique for a given workflow.

Although there are various workflow engines featuring different workflow definitions, the most supported workflow definition language at the moment is BPEL, followed by XPDL. Some workflow engines even support both like JBoss jBPM. Both feature a similar set of capabilities for defining workflows, which will in most cases be sufficient for describing a given business workflow. An interesting evolving workflow language is YAWL, since it is an approach based on the analysis of already existing workflow languages. It tries to learn from weaknesses from former languages and is supported by Van der Aalst which has done a lot of research in methods of comparing workflow languages. However, the language is not supported widely at the moment and the author of the thesis doubts that this will change in the near future.

While choosing the right workflow language, it is important to look at the given business workflow that needs to be implemented: Does it require a lot of interaction with users, or is it mostly automated ? Does the user interact in a way with the workflow that requires synchronous invocation of webservices, having the user wait for the result? How responsive has the workflow to be and how important is performance during the execution of the workflow ? Workflow definitions are a great tool for modeling a business workflow, but still there are cases in which the overhead of using a series of webservices with a workflow engine creates such an amount of overhead that the result is not worth the effort. It has also to be kept in mind that some workflow languages support user interaction more than others. The example implementation of the author has shown that it may generate a lot of work if this aspect is not considered during the design of the workflow.

In the case of BPEL, there is already an extention suggested that is called BPEL4People. It integrates a separate activity for interaction with users called people activity. Since there are already efforts to implement this enhancement for existing workflow engines although this extention is not completely specified yet, shows that there is a need for workflow solutions including such activities.

Choosing the right workflow engine for a given workflow language is not easy, especially for BPEL, since there already exist some good implementations from big software companies as well as less cost intensive solutions from Open Source organizations. This thesis only points out some popular engines that already find use in some serious projects as well as engines that are not used so often but implement one of the more scientific driven approaches on workflow definition languages.

The evaluation of solutions for displaying an Interactive Document Preview shows, that there are many viable solutions for creating such documents in a browser-driven environment. Simple solutions like basic XHTML pages enhanced with either XForms and/or SVG may lead to solutions that come very close to the resulting document and fit very well into an existing web-based infrastructure. However, XHTML has several limitations concerning its capabilities to display arbitrary layouts. More sophisticated solutions feature Flash, SVG or XUL implementations, that have the possibility to come very close to the resulting document while being able to support all levels of interactivity. The difficulty concerning these techniques is that it is hard to automatize the creation of such template documents based on, e.g., an XSL-FO document.

The result could be worth the effort, however. Open Laszlo is an interesting project since it is based on the idea of being an abstraction over technologies like Flash or XHTML and offers therefore a lot of options and perspective.

In comparison to the capabilities of this “Thin-Clients”, “Fat-Clients” do not offer much more abilities. The main advantage of Microsoft’s Infopath is, that it can be integrated in the Sharepoint environment. In cases where InfoPath is already in use, this might be an options, since mainly Infopath documents are a composition of HTML and XML documents which can easily be customized outside manual correction in the Infopath editor.

Generally this means that, although there are some possibilities for creating interactive previews to be viewed within a Fat client environment, there is no good reason to do so with existing workflow engines. Most of them are XML based and can easily be integrated within a Thin Client solution, making it more independent on a computer’s software configuration.

The example implementation of a workflow based on the research before shows that some effort has to be made to find a good transformation process from the document format that is used for printing a document to a document format that allows to create a preview with the level of interactivity that is shall be reached. This implementation tries to capture a basic level of interactivity by allowing to change single values in a document as well as add elements to a list of already existing set of element. This is normally enough for most scenarios like customer accounts or letters.

The approach shown in this implementation is build on synergies and similarities between the two languages used, XSL-FO and XHTML with embedded Xforms. For the layout, both languages are basically build on the information model that is CSS, even though XSL-FO is a page-based media which XHTML is not. Although XSL-FO does not feature external stylesheets, through the use of inline style information it is easy to bring this layout information to XHTML. There more difficult part is simulating elements that have no identical equivalent in XHTML, like lists: although there are list definitions in XHTML, they cannot be customized enough to be used for simulating XSL-FO lists, even with the use of CSS. Although there are differences between the two primary supported browsers, Internet Explorer 6.0 and Firefox, the differences are not significant enough to consider them for this implementation.

Another difficult part of the implementation was the underlying framework and workflow engine used. Especially the interaction between browser action and workflow response can be tricky. A lesson learned from this implementation is that when having to deal with user interaction within a workflow definition, a focus has to layed on the handling of this interaction to integrate it into the workflow defintion as seamlessly as possible.

It is also important to evaluate all participating components if they really fit into the planed architecture. For example, although the author of this thesis wanted to use the Release Candidate from Chiba 2.0.0 in its final implementation, some serious bugs including the inability to redirect to arbitrary Servlets after having previewed a document made it impossible to use it in the final implementation.

10 Bibliography

References

[1] Adobe. Macromedia Flash. http://www.adobe.com/products/flash/flashpro/.

[2] Apache. Batik. http://en.wikipedia.org/wiki/Batik_%28software%29.

[3] Uwe Assmann. Architectural styles for active documents. *Sci. Comput. Program.*, 56(1-2):79–98, 2005.

[4] Christophe Coenraets, Vincent Mendicino, Natalia Shmoilova, and Dirk Wodtke. Creating next generation sap analytics applications with SAP NetWeaver and Macromedia Flex. http://www.adobe.com/products/flex/whitepapers/pdf/sap_flex.pdf.

[5] WWW Consortium. Resource description framework. http://www.w3.org/RDF/.

[6] WWW Consortium. Simple object access protocol (SOAP). http://www.w3.org/TR/soap/.

[7] WWW Consortium. Web Service Definition Language (WSDL). http://www.w3.org/TR/wsdl.

[8] WWW Consortium. Xml schema. http://www.w3.org/XML/Schema.

[9] WWW Consortium. Xml stylesheet language (XSL). http://www.w3.org/Style/XSL/.

[10] Schahram Dustdar, Harald Gall, and Manfred Hauswirth. *Software-Architekturen fr Verteilte Systeme*. Springer Verlag, 2003.

[11] Apache Foundation. Axis. http://ws.apache.org/axis/.

[12] Apache Foundation. Tomcat servlet engine. http://tomcat.apache.org/.

[13] Juha Haataja and Renne Tergujeff. Using BPEL4WS for Supply-Chain Integration - Experiences and Evaluation. Technical Report C-2003-74, Department of Computer Science, University of Helsinki and VTT, 2004.

[14] Matthew J. Horn and Mike Peterson. Getting started with Flex. http://download.macromedia.com/pub/documentation/en/flex/15/flex_getting_started.pdf.

[15] IBM. BPEL reference. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[16] IBM. BPWS4J. http://www.alphaworks.ibm.com/tech/bpws4j.

[17] IBM. Domino workflow white paper automating real-world business processes. ftp://ftp.lotus.com/pub/lotusweb/eibu/dominoworkflow.pdf.

[18] Konstantin Ignatyev. Xflow2 process management system. http://kgionline.com/xflow2/.

[19] JBoss. jbpm. http://www.jboss.com/products/jbpm.

[20] Matjaz B. Juric, Bostjan Kezmah, Marjan Hericko, Ivan Rozman, and Ivan Vezocnik. Java rmi, rmi tunneling and web services comparison and performance analysis. *SIGPLAN Not.*, 39(5):58–65, 2004.

[21] Michael Kay. Saxon. http://saxon.sourceforge.net/.

[22] James C. King. A format design case study: Pdf. In *HYPERTEXT '04: Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, pages 95–97, New York, NY, USA, 2004. ACM Press.

[23] Kloppmann, Koenig, Leymann, Pfau, Rickayzen, Riegen, Schmidt, and Trickovic. WS-BPEL Extension for People – BPEL4People. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.

[24] Laszlo Systems, Inc. Open Laszlo. http://www.openlaszlo.org.

[25] Andrea Marchetti, Maurizio Tesconi, and Salvatore Minutoli. Xflow: An xml-based document-centric workflow. *Lecture Notes in Computer Science, Volume 3806, Oct 2005*, pages 290 – 303, 2005.

[26] Microsoft. Rich text format (RTF). http://www.microsoft.com/downloads/details.aspx?FamilyID=ac57de32-17f0-4b46-9e4e-467ef9bc5540&displaylang=en.

[27] Microsoft. Vector markup language (VML). http://msdn.microsoft.com/workshop/author/vml/default.asp.

[28] Microsoft. Windows vista. http://www.microsoft.com/windowsvista/default.aspx.

[29] Jinyoung Moon, Daeha Lee, Chankyu Park, and Hyunkyu Cho. Transformation algorithms between BPEL4WS and BPML for the executable business process. In *13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 135– 140. WET ICE, 2004.

[30] Mozilla. Mozilla organisation. http://www.mozilla.org/.

[31] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. Ws-security. Technical report, OASIS, Mar. 2004.

[32] OASIS. Open document format for office applications (ODF). www.oasis-open.org/committees/office/.

[33] Queensland University of Technology. Yawl: Yet another workflow language. http://www.yawl.fit.qut.edu.au/.

[34] Object Management Group (OMG). Bpmi.org. www.bpmi.org.

[35] Oracle. BPEL process manager. http://www.oracle.com/technology/products/ias/bpel/index.html.

[36] Steve Probets, Julius Mong, David Evans, and David Brailsford. Vector graphics: from postscript and Flash to SVG. In *DocEng '01: Proceedings of the 2001 ACM Symposium on Document engineering*, pages 135–143, New York, NY, USA, 2001. ACM Press.

[37] Open Source Project. Chiba. http://chiba.sourceforge.net/.

[38] Robert M. Shapiro. Xpdl 2.0: Integrating process interchange and bpmn. *2006 Workflow Handbook including Business Process Management*, 2006.

[39] Open Source. The Flame Project. http://www.flameproject.org/index.php/Main_Page.

[40] Rick Strahl. Past the AJAX hype - some things to think about. http://www.west-wind.com/weblog/posts/2725.aspx.

[41] SUN. Java runtime environment. java.sun.com.

[42] Jacques Surveyer. JavaScript and Flash ActionScript. http://theopensourcery.com/jsactscript.htm.

[43] Informatik Universitt Hamburg. Petri nets. http://www.informatik.uni-hamburg.de/TGI/PetriNets/.

[44] Wil van der Aalst. Workflow patterns. http://is.tm.tue.nl/research/patterns/products.htm.

[45] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and implementation of the YAWL system. *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE04)*, 3084, 2004.

[46] WMP van der Aalst and AHM ter Hofstede. YAWL: Yet Another Workflow Language (Revised Version). Technical report, Technical Report FIT-TR-2003-04, QUT, Queensland University of Technology, Brisbane, April. Accessed from http://www. citi. qut. edu. au/pubs/technical/yawlrevtech. pdf. 14, 2003.

[47] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5 – 51, 2003.

[48] Wil M.P. van der Aalst1, Marlon Dumas2, Arthur H.M. ter Hofstede, and Petia Wohed. Pattern based analysis of bpml (and wsci). *FIT Technical Report FIT-TR-2002-05*, 2002.

[49] Helmut Vonhoegen. Galileo computing. http://www.galileocomputing.de/artikel/gp/artikelID-164.

[50] W3C. Synchronized multimedia integration language (SMIL). http://www.w3.org/AudioVideo/.

[51] www.enhydra.org. Enhydra.org community. http://www.enhydra.org/index.html.

[52] www.gnu.org. Lesser Gnu Public License. http://www.gnu.org/copyleft/lesser.html.

[53] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, and Kerard Hogg. Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 301–312, New York, NY, USA, 2005. ACM Press.

11 Figures and Tables

List of Figures

1 A simple Workflow Chart 14
2 A simple XPDL workflow chart with Activities and Events [38] 20
3 YAWL Architecture[45] 26
4 Domino Workflow Architect[17] 29
5 An example Web Application Architecture with Flex 36
6 A Flash Application 37
7 A SVG application 41
8 An example Open Laszlo Application 42
9 Class Chart of the Extension of the editor 51
10 Simple Flowchart of the Workflow 54
11 BPMN chart of the Workflow 56
12 Layer Model 57
13 Overview Conversion XSL-FO to XHTML: page construction and fo:block/fo:block-container 60
14 Overview Conversion XSL-FO to XHTML: fo:table and fo:list-block 61
15 Overview Conversion XSL-FO to XHTML: inline elements 62

List of Tables

1 Formatting Attributes — supported: 63
2 Formatting Attributes — not supported: 64
3 Not supported attributes in specific elements 65