XFlow - An xml-based document-centric workflow. How to stress XML technologies in a web application

A. Marchetti, M. Tesconi, S. Minutoli

IIT TR-16/2005

Technical report

Settembre 2005
XFlow

An xml-based document-centric workflow

How to stress XML technologies in a web application

Andrea Marchetti, Maurizio Tesconi, Salvatore Minutoli

CNR, IIT Department, Via Moruzzi 1, I-56124, Pisa, Italy
(andrea.marchetti, salvatore.minutoli, maurizio.tesconi)@iit.cnr.it

keywords: document-centric workflow, cooperative authoring, web application, xml technologies
Abstract

To ensure the efficiency of document circulation and administration in generic enterprise day-to-day operations, appropriate workflow management systems (WFMS) are necessary. This paper aims at investigating on an appropriate framework that allows the definition of workflows for collaborative document procedures, to be used in a multiplatform and decentralized enterprise environment.

This framework, called XFlow, is based on a complete independence between workflow definition and engine, and supports the simultaneous collaborative document processing. Workflows are described by means of a new XML application called XFlowML (XFlow Markup Language) largely based on XSLT Processing Model. XFlowML describes the document workflow using an agent-based approach. Each agent can participate to the workflow with one or more roles defined as XPath expressions based on a hierarchical role chart. An XFlowML document contains as many templates as agent roles participating to the workflow. The selection of the templates will establish the order with which the agents will receive the document.

The document workflow engine constitutes the run-time execution support for the document processing by implementing the XFlowML constructs. A prototype of XFlow has been implemented with an extensive use of XML technologies (XSLT, XPath, XForms, SVG) and open-source tools (Cocoon, Tomcat, mySQL). It is a web-based application where human agents interact with the system through an XForms browser that displays the document to process as a web form whereas software agents interact with the system via web services. In the next future XFlow will be extended in order to develop a distributed framework based on mobile agents.
Index

1 Introduction .................................................................................................................. 4

2 Overview: definitions, concepts, terminology .......................................................... 5

3 Document Workflow Framework ................................................................................. 7

3.1 Document Workflow Description ............................................................................ 8
   3.1.1 Document Schema (XML Schema) ................................................................. 9
   3.1.2 Document Interface (XForms, Web Service) .................................................. 9
   3.1.3 Role Chart: Agent Role Declaration (RCML) .................................................. 9
   3.1.4 XFlow: Document workflow definition (XFlowML) ..................................... 10
   3.1.5 Metadata and Log Components .................................................................... 13

3.2 The Document Workflow Engine (DWE) ............................................................... 14
   3.2.1 Agent’s activities support .............................................................................. 15
   3.2.2 Internal Agents .............................................................................................. 17
   3.2.3 System modules ............................................................................................. 17

4 Case study: a travel request ......................................................................................... 18

4.1 Travel Request Flow Description ............................................................................ 20
   4.1.1 The rolechart document .............................................................................. 20
   4.1.2 The XFlow document .................................................................................. 21

5 Implementation Overview ............................................................................................ 23

5.1 The client side: external agent interaction ............................................................. 23
5.2 The server side ....................................................................................................... 25

6 Conclusion .................................................................................................................. 28

7 References .................................................................................................................... 29
1 Introduction

The problem of processing documents has long been recognized to be a critical aspect in the enterprise productivity ([3], [6], [7], [8], [9]). The management of documents becomes more difficult when it involves different actors, possibly in a decentralized working environment, with different tasks, roles and responsibilities in different document sections.

Many enterprise day-to-day operations can be viewed as a series of steps involving the filling out of appropriate forms by different actors, sometimes with a concurrent processing. We can expect that implementing an effective system that supports these forms flows will result in considerable cost savings for enterprises. In addition, a more efficient management of documents can help in organizing and carrying out other activities, thus improving the overall productivity.

Enterprises need to better track and manage their document functions such as task assignment and workflow. For this purpose, an appropriate document Workflow Management System (WFMS), where documents and actors are organized in such a way to capture the relationships existing among them and introduces effective mechanisms to manage and automate this kind of processes, could be a solution to this problem.

Furthermore multiplatform environments require an interoperable and portable support to guarantee a correct workflow management and the possibility to define and manage multiple document workflows simultaneously is a prerequisite to efficient solutions for enterprise document management.

We propose a solution based on a complete independence between workflow definition and workflow engine which supports the management of simultaneous document workflows.

The remainder of this paper is organized as follows. Section 2 gives an overview of the terminology, presenting the main characteristics of document workflow policies. Section 3 presents the relevant figures of the document workflow model. In section 4, a case study is set up, on which our system effort is exercised to illustrate the practical utility of our approach. In section 5 an overview of the implementation is illustrated. Final remarks and indications for future developments can be found in section 6.
2 Overview: definitions, concepts, terminology

Before illustrating our approach, we give some remarks on the terminology used.

A workflow is “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules”.

As 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 ([1],[3],[9]). 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.

Through a DW a document life-cycle is tracked and supervised, continually providing document compilation actions control. In this environment a document travels among agents who essentially carry out the pipeline receive-process-send activity. There are two types of agents: external agents are human or software actors which perform activities dependent from the particular DW, and internal agents are software actors providing general-purpose activities useful for any DW and, for this reason, implemented directly into the system. An external agent executes some processing using the document content and eventually other data, updates the document inserting the results of the preceding processing, signs the updating and finally sends the document to the next agent(s).

Internal agents perform general functionalities such as creating a document belonging to a particular DW, populating it with some initial data, duplicating a document to send to multiple agents, splitting a document to send partitions to different agents, merging duplicated documents coming from multiple agents, aggregating document fragments, terminating operations on the document.

Figure 1 illustrates a generic document workflow diagram where external and internal agents cooperate exchanging documents according to some procedural rules
Fig. 1. A generic document workflow
3 Document Workflow Framework

Our document workflow framework is based on document-centric model where all the activities, made by the agents, turn out to documents compilation.

During its life the document passes through several phases, from its creation to the end of its processing. The state diagram in figure 2 describes the different states of the document instances. At the starting point of document instance life cycle there is a creation phase, in which the system raises a new instance of a document with several information attached (such as the requester agent data). The document instance goes into pending state. When an agent gets the document, it goes into processing state in which the agent compiles the parts of his competence. If the agent, for some reason, doesn’t complete the instance elaboration, he can save the work performed until that moment and the document instance goes into freezing state. If the elaboration is completed (submit), or cancelled, the instance goes back into pending state, waiting for a new elaboration.

In order to automate DWs, an engine with the task of managing all the functionalities to support agent activities is necessary. Our goal has been to design a DW engine which is independent from the single DW. This allows adding new DWs without having to modify the engine. For this reason it’s necessary to isolate the information of each DW separating it from the engine. The DW engine will have some parser to interpret these descriptions. We will see in detail the essential components necessary to describe a DW.
3.1 Document Workflow Description

The description of a DW can be seen as an extension of the XML document class. A class of documents, created in a DW, share the schema of their structure, as well as the definition of the procedural rules driving the DW and the list of the agents attending to the DW. Therefore in order to describe a DW we need four components:

- a *schema* of the documents involved in the DW;
- the agent roles chart, called *role chart*, i.e. the set of the external and internal agents, operating on the document flow. Inside the role chart these agents are organized in roles and groups in order to control who accesses the document. This component constitutes the DW environment;
- a *document interface description* used by external agents to access the documents. This component also allows to check the access to the document resource;
- a *document workflow description* defining all the paths that a document can follow in its life-cycle, the activities and policies for each role.

Furthermore the system for keeping track of document instances history (including the agents that have manipulated the document) and document state during its whole flow path needs respectively a *Log* and *Metadata* component. The Metadata component represents the document current state. Every time a document changes its state (see Figure 1) the Metadata is first saved into Log component and then updated. These last two documents are produced automatically by the DW engine. For each component we define a declarative language, using XML. Hence, each document belonging to a DW will have associated six documents that take for all its life-cycle as indicated in figure 4.
3.1.1 Document Schema (XML Schema)

*Document schema* describes the structure and the data-types of the documents participating to the flow. *Document schema* will be described using XML Schema.

3.1.2 Document Interface (XForms, Web Service)

This document describes for each agent role the interface toward the document. The *document interface* for *external human agents* relies upon Web Modules technologies [2], and *external software agents* make use of Web Services technologies (WSDL, SOAP). In the first solutions we adopted XForms technology, promoted by W3C [18].

3.1.3 Role Chart: Agent Role Declaration (RCML)

The *role chart* is a XML document containing the description of all actors (agents) that participate to the workflow. Each actor has a *role* and a *unique identifier*. Roles are organized in the roles chart hierarchically. Each agent can participate to the workflow with one or more roles, therefore it can appear in one or more roles chart points. The roles chart schema is depicted in figure 5.
Finally the document workflow description is a document based on a new XML application (XFlowML Xml document workflow Markup Language) suitably defined for this purpose.

### 3.1.4 XFlow: Document workflow definition (XFlowML)

For the definition of a language to describe complex document flows we analyzed several syntaxes and approaches. A possible solution was to use a notation similar to concurrent languages, using statements like fork and join to describe flows. Another choice was to describe the document flow from the point of view of the agents. To describe a document flow it is sufficient to accurately describe all the agents and all the operations any agent can perform on the document instance. This way of describing the flow resembles XSL syntax [22], where actions performed by various agents are similar to the templates to apply to the elements of an XML document. Our basic decision to represent flows as XML documents, led us to choose the second option, since with XML, due to its intrinsic hierarchical notation, it is more straightforward to represent lists rather than graphs (other approaches which emphasize the role of XML can be found in [4], [5], [13], [26], [27]). Taking as a simple example the generic flow depicted in Figure 1, we will have to supply as many descriptions as the agents roles involved in the process. For instance, in the description of the external agent with role1 (Ag. Role1), we must specify that it can receive documents from Creator or Ag.Role4, and send it to Ag.Role2 and Ag.Role3.

Figure 6 shows both the graphical representation and the XML notation of the Agent Role1.
To describe document flow we adopted a XML dialect, called XFlowML, largely based on XSL-Syntax, whose schema is represented in Figure 7.

A XFlowML document is composed of a list of internal or external agent. Each agent has a mandatory attribute role, containing a XPath expression ([21]) referring to the rolechart. Other optional attributes specify if the agent has to sign the document (sign), and the maximum time the agent is allowed to keep the document (timeout).
When an agent requests a document, the DW Engine matches the agent’s role on XFlow document and processes the three section: receive, action and send.

In the receive section the from elements identify from which agent roles the document can be received. The roles of the agents are coded as XPath expressions. The receive section is optional because it’s necessary only to verify if the agent can really receive the current document.

In the action section there are one or more permission elements defining the access policies to the document fields.

The send section contains all the possible receivers of the document. The document can be sent simultaneously to several agents by using a sequence of to elements.

In order to increase the flexibility and power of the language, thus allowing for an easy definition of more complex DWs, we introduced the conditional <i>if</i> and <choose> statements which adopt the XSLT syntax ([22]) and can be specified in any agent section (i.e. receive, action, send). Test attributes can contain any XPath expression which returns a Boolean, and it is possible to refer document Metadata or document instance. For distinguishing the referred document the test XPath expression will begin with two different prefixes: respectively $Metadata and $Instance.

Fig. 8 Inside XFlowML document we can have Xpath expression referring elements or attribute of Rolechart, Document, and Metadata. To distinguish them, they have the prefixes $instance e $metadata.
A typical use of these conditional statements is in send element, when we have to send the document to two different agent depending on the value of a document field previously filled out (see Figure 9).

Fig. 9 The send section with XSLT conditional statements

### 3.1.5 Metadata and Log Components

For each document instance that participate to a specific DW, additional information (that we have called metadata) is stored together with information to reconstruct the document history (Log) that consists of all the document transitions during its processing, including also information about actors involved. Log permits to undo actions.

The metadata associated with each document is described in the following example ..

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn</td>
<td>Univocal document’s name. It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>urn</td>
<td>Univocal document’s name. It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>flowId</td>
<td>Identifiers of the flow who the instance belong to. It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>docTitle</td>
<td>Document’s title. It’s depended from flow</td>
</tr>
<tr>
<td>docFileName</td>
<td>Instance file’s name. It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Date of the creation of the tuple. It corresponds to the state change of the document.</td>
</tr>
<tr>
<td>docDeadline</td>
<td>Deadline relevant to the completion of a document iter. It should be the deadline before the agent receives the document. As deadline it can be absolute (before 31 December 2003) or relative (before 30 days from date of the creation of the instance). It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>recDeadline</td>
<td>Deadline relevant to the completion of the elaboration of the received agent. It can be relative or absolute.</td>
</tr>
<tr>
<td>creator</td>
<td>RolePathId of the agent who has instanced the document. It doesn’t change from its creation until its registration.</td>
</tr>
<tr>
<td>sender</td>
<td>RolePathId of the agent who has done a submit on instance.</td>
</tr>
<tr>
<td>receiver</td>
<td>rolePathId of the agent who have to receive the instance. It can be a rolePathId to identify exactly an agent or a rolePath to identify a group of agents that can receive without distinct the document.</td>
</tr>
<tr>
<td>handler</td>
<td>RolePathId of the agent who is elaborating the instance.</td>
</tr>
<tr>
<td>status</td>
<td>State of the instance (processing, frozen, pending archived)</td>
</tr>
</tbody>
</table>

### 3.2 The Document Workflow Engine (DWE)

The document workflow engine constitutes the run-time support for the DW, it implements the *internal agents*, the support for *agent’s activities*, and some *system modules* that the external agents have to use to interact with the DW system. Also, the engine is responsible for two kinds of documents useful for each document flow: the *documents system logs* and the *document system metadata*.

In addition the DWE implements modules *Document Interface Manager*, *DW Interpreter*, *Document Base Manager*, *Digital Signature Manager* everyone responsible for a specific task.
3.2.1 Agent’s activities support

These are the two modules, called Sender and Receiver, supporting the activities of sending to, and receiving from the current agent1.

The Sender has to prepare and send the document (identified by an URN) requested by an external agent. It checks the agent rights, verifies if the document instance is still available, analyzes/interprets the workflow description to generate an adapted document, using the agent’s role and access rights to determine which are the parts of the stored document to be included. (using XForms for a human agent and SOAP for a software agent).

The Receiver gets the document from the handling agent in consequence of a submit, freeze or cancel command. It determines the roles of the next agents to whom the document must be sent.

Both modules use the DW Interpreter which transforms the XFlow document into a XSLT stylesheet. The generated stylesheet is applied to the Rolechart document producing the role agent’s activities.

3.2.2 Document Workflow Interpreter

The DW Interpreter is based on two XSLT processing steps.

---

1 The name of these modules are given with the point of view of DWE
At the first step it takes in input the XPath expression, corresponding to the current agent, that is the agent that received or sent the document. This expression selects a leaf node of the role chart, therefore it identifies precisely both the agent and the role used to make the request. In addition the DW definition (XFlowML document) related to the current document is recovered. From these two data the XSLT processor driven by a stylesheet creates a new stylesheet. This stylesheet contains as many templates as the agent roles described in the flow and a root template including an apply-templates instruction that selects the template corresponding to the current agent.

At the second step the stylesheet, prepared at the previous step, is executed in order to identify the XFlow template relative to the current agent and resolve any conditional statement contained in this fragment.

This XSLT transformation is quite atypical because it contains only a matching operation and the xml input file (rolechart document) isn’t suffered any transformation but it is necessary only like support to select the template.

Summarizing the DW Interpreter identify the XFlow fragment relative to the current agent. With this information the sender module will prepare the interface to the document fragment for the agent while the receiver module will know to which agent the document have to be sent, information that the receiver will insert into metadata document.
3.2.2 Internal Agents

Three agents execute generic operations useful for each documents flow. They are referred inside the documents workflow description (XWDL).

The Merging agent deals with the fusion of multiple documents, especially when the document is doomed to be forked in different parts that have to be manipulated from different agents.

The Creator has to produce a new document instance (belonging to a predefined workflow), pre-filling with some agent personal data recovered from some data base.

The Terminator -- x Maurizio e Salva --

3.2.3 System modules

Generic functions are performed by Authenticator, Workload Sender and Notifier.

Authenticator is the first module activated when an agent involved in the DW accesses the system. It consists in checking (through user name and password or certificate) that the user is properly registered.

Workload Sender is activated after agent authentication. It receives the name of the agent involved and generates for this agent a list of documents the agent can act upon.

Notifier has the task of alerting the agent about some important deadlines. Until now the Notifier consists in sending an e-mail or text to alert a human agent that has to process the document.
4 Case study: a travel request

To illustrate the usefulness of our system, a case study is set up and briefly outlined. One typical activity in a research institute is the participation in conferences or seminars. In this case, the employee (researcher) must obtain the proper authorization, involving the approval of the office manager, administrative verification, and final approval by the director.

In terms of workflow, this consists of filling out several mandatory fields (such as purpose, destination, duration and dates of the trip and estimated daily traveling allowance) by the employee. Some employee’s data, such as name and division can be pre-filled by the application.

Figure 11. Travel Request Graph

Figure 12 Travel Request Form for the employee
Then the document must be approved by the office manager and by the administration.

These two activities are independent and can be performed concurrently, therefore the document is duplicated in different copies, each one sent to the appropriate actor.

![Figure 13 Travel Request Form for Division Head](image1.png)

![Figure 14 Travel Request Form for Administration](image2.png)

Afterwards the document is properly recomposed from the merging agent and sent to the director for authorization.
Here each actor involved fills out a different part of the form, as shown by Figures 12, 13, 14 and 15. The method used to visualize the document consists of an adaptive user interface that shows the fields to be filled out.

Note that the Administration is actually a role, designating a set of individuals who can perform the task, while each of the other actors is a specific person.

### 4.1 Travel Request Flow Description

In order to describe this flow it’s necessary to create a schema as well as a rolechart and a xflow document.

#### 4.1.1 The rolechart document

A rolechart document contains all the agents organized by groups and roles. The roles are: employee, division head, administrator and manager.
Let’s suppose that in our firm ACME there is only the division “Foo” where George is the division head and Bob is the only employee. Two employees work in administration. The manager of ACME’s firm is Alice. Figure 16 describes this situation.

### 4.1.2 The XFlow document

The travel request graph is codified in xflowML document. It’s organized in 5 sections corresponding on the 5 agents roles: employee, division head, administration employee, merging (internal agent necessary for merging the document coming from division head and administration, see Figure 11) and the manager (see Figure 17).

In the figure 18 we can see the send section of the “Employee” agent. In this section it is declared that when the Employee submits the document this can follow two paths. If the document has just been created, it will be duplicated and sent to an administration employee and to the division head of the employee, otherwise the document will be archived because the process is ended.
Figure 18 XFlow document
5 Implementation Overview

5.1 The client side: external agent interaction

Our system is currently implemented as a web-based application where the human external agents interact with system through a web browser. All the human external agents attending the different document workflows are the users of system. In Figure 18 and 19 the use cases and the state diagram of user activities are shown.

![Fig. 19. The use cases diagram](image-url)
Once authenticated through user/psw (Fig. 20A) the user accesses his/her workload area (Fig. 20B) where the system lists all his/her pending documents sorted by flow. The system shows only the flows to which the user has access.

From the workload area the user can browse his/her documents and select some operations such as:

- select and process a pending document (Fig. 20C)
- create a new document partly filled with his/her data.
- display a graph representing a DW of a previously created document, highlighting the current position of the document (Fig. 20D). This information is rendered as an SVG image.
The form used to process the documents is rendered with XForms [19] (See Fig. 21C). XForms can communicate with the server by means of XML documents and is capable of displaying the document with a user interface that can be defined for each type of document. XForms is a recommendation of the W3C for the specification of Web forms. In XForms the description of how the form is displayed is separated from the description of what the form must do, so it is easy to use different type of views depending on the platform and on the document. A browser with XForms capabilities will receive an XML document that will be displayed according to the specified template, then it will let the user edit the document and finally it will send the modified document to the server.

5.2 The server side
The server-side is implemented with Apache Tomcat, Apache Cocoon and MySql. Tomcat is used as the web server, authentication module (when the communication between the server and the client needs to be encrypted) and servlet container. Cocoon is a publishing framework that uses the power of XML. The entire functioning of Cocoon is based on one key concept: component pipelines. The pipeline connotes a series of events, which consists of taking a request as input, processing and transforming it, and then giving the desired response. The pipeline components are generators, transformers, and serializers. A Generator is used to create an XML structure from an input source (file, directory, stream ...). A Transformer is used to map an input XML structure into another XML structure (the most used is XSLT transformer). A Serializer is used to render an input XML structure into some other format (not necessarily XML).

MySql is used for storing and retrieving the documents and the status of the documents.

There are some modules that allow the interaction with the user agents.

The Authenticator and WorkloadSender modules use XHTML to display data. The Receive and Sender Document modules use XForms to exchange XML document with human agents and the SOAP protocol to exchange documents with software agents. (fig. 22).

Each software agent is implemented as a web-service and the WSDL language is used to define its interface. The Notifier is used to alert an agent when some document needs
to be processed in a short time (deadline) and optionally when a new document is inserted in his work list.
6 Conclusion

In this paper we have described a framework to define a workflow for collaborative document procedures. It is based upon a complete independence between workflow definition and engine, and supports the simultaneous processing of collaborative documents.

The benefits arising from the usage of this system include:

- interoperability/portability deriving from using XML technologies;
- concurrent documents workflows managing;
- reduction of the cost of documents processes, through the e-documents processing and distribution;

XML is a suitable technology for representing not only data/documents but also to describe the document workflow logic. XFlowML is the XML application defined to describe a document workflow.

We have defined a model to describe DW based on three XML documents (schema, rolechart and xflow) that allows an easy description of many DW.

We have implemented a DW engine interpreting XFlowML documents, by using Cocoon, a very powerful middleware, to develop XML prototypes.

The DW engine implemented is heavily based on XML technologies (XSLT, XPath, XForms, SVG) and open-source tools (Cocoon, Tomcat, mySQL).

We have used Xforms technology to create dynamic user interfaces.

Our future work will focus on extension of the DW engine with new internal agents and development of a distributed framework based on mobile agents.
7 References


27. R. Tolksdorf Workspaces: A Web-Based Workflow Management SystemIEEE Internet Computing September 2002 v.6 n.5 p.18-26