

Experience paper: Using XML to implement a workflow tool

Alf Inge Wang*
NTNU, Trondheim, Norway.

April 4, 2001

Abstract

This paper presents experiences we had from building a workflow tool from scratch using XML technology. We will present some strengths found using XML-technology, but also some weaknesses. Although we had to create a simple process modelling language for this workflow tool, the focus of this paper is on experiences on using XML technology to build workflow tools. The experiences we have achieved, should be applicable for all kinds for process modelling languages. The paper consists of three main parts. First, the requirements for the workflow tool is outlined. Then XML technology is explained with some simple examples. The last part of the paper describes experiences we achieved from the experiment and the conclusions we drew from this.

Keywords: *Process modelling language representation, workflow tools, XML*

1 Introduction

Spring 1998, the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) was asked by a project called Renaissance, to create a simple workflow tool to demonstrate the *Renaissance process*. The Renaissance project was a partially founded project by the European Commission under the Framework Initiative (ESPRIT 22010). The main objective of the Renaissance project was to develop a systematic method to support the re-engineering of legacy systems. Among the results of the Renaissance project was the Renaissance method described in the Renaissance method book [3]. This book describes a step-by-step process for re-engineering legacy systems in an informal graphical process language. Our assignment was to create a simple graph-

ical web-based workflow tool that made it possible to go through the whole process in an interactive manner.

Our research interest for this assignment was not to create yet another Process Modelling Language (PML), but rather to see what technology to use to build a simple workflow tool over a short period of time and with scarce resources. The PML we chose, represents a process as a activity network interconnected with artifacts. Activities are activated through pre-conditions constrained by the states of their input artifacts. Although this papers will outline how the PML is represented in XML, the focus of this paper is on experience using XML to build a workflow tool (not the PML itself).

The rest of this paper is organised as following. Section 2 outlines the requirements for the workflow tool we built. Section 3 explains what XML is and give some simple examples of how to use XML. Section 4 describes the experiment of creating a workflow tool using XML. Section 5 presents the experiences we have achieved from using XML as a basis of a workflow tool. Section 6 concludes this paper.

2 The Renaissance process model

This section will outline the process model elements found in the Renaissance method description (the Renaissance process model) and how these elements are inter-connected. The description of the Renaissance model gave us the requirements for building the workflow tool and put constraints on what elements we should include. One problem with this description was that the process was described in an informal way. This caused that we had to add some elements to the PML to make the process executable.

*Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), 7035 Trondheim, Norway. Phone: +4773594485, Fax: +4773594466, Email: alfw@idi.ntnu.no

2.1 The basic process model elements

The Renaissance process model focuses on activities and documents needed or produced by the activities. In addition roles are used to assign persons to specific tasks. The following basic constructs were a part of the model:

- **Activity** An activity is decomposable, and consists of sub-activities or sub-tasks. An activity is described by a *name*, *description*, *inputs* and *outputs*. The *pre-* and *post-condition* of an activity is depending on sub-activities/tasks as described in section 2.3.
- **Task A** Task is an atomic unit, and cannot be decomposed. A task is also described by a *name*, *description*, *inputs*, *outputs*, *pre-conditions* and *post-conditions*. In addition a task description contains a list of roles *responsible* for the task. A task can be executed in parallel as well as in a sequential manner. Both for activities and tasks, the state of the inputs decides when to execute.
- **Input/Output** Inputs and Outputs refers to documents or collections of document that are involved in the process, and have a *name* as well as a *state*.
- **Roles** Roles define a generalised description of someone responsible for a task (e.g., project leader, secretary etc).
- **User** A user is a named human resource that can play several *roles* in a process.

2.2 Relations between process model elements

As indicated in previous section, activities and tasks have relationships to inputs and outputs (documents). In addition activities and tasks can be related in four different ways as illustrated in figure 1 and described below:

1. **Consist of relation** describes the relation between an activity and its children (the children can both be activities and tasks).
2. **Sequential activity flow relation** describes that two activities/tasks are executed sequentially.
3. **Concurrent activities relation** describes that two or more activities/tasks are executed in parallel.

4. **Concurrent iterative activities relation** describes that two activities/tasks are executed in a loop until a specified condition is fulfilled.

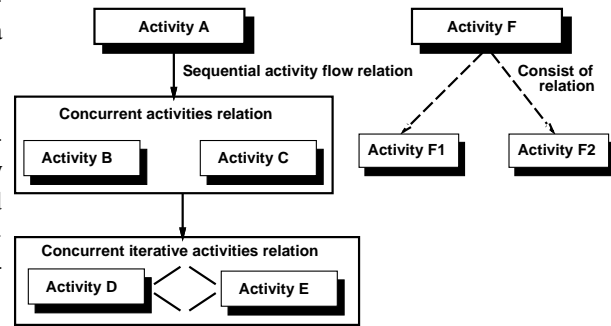


Figure 1: Relation types between activities/tasks

Since the Renaissance process did not have any conditional process flows (e.g., IF condition1=true THEN do activity1 ELSE do activity2), conditional flow is not a part of the PML.

2.3 Process state representation

The description above outlines the PML to be used to implement a workflow tool to support the Renaissance process. More detailed information for how to make this process representation executable was not available from the Renaissance project documentation. We had to decide how to represent process states and find the mechanisms to make the process model executable. We choose to use a state database to cope with the dynamic aspects of the process model. The state database was divided into three parts:

1. **Activity data** Keeps the state information about each activity and task in the process model. Whenever a pre-condition (input) or post-condition (output) is fulfilled, the activity/task state may change. An activity/task can have four states, Not ready (0), Ready (1), Started (2), and Finished (3).
2. **Condition data** Keeps the state information about each condition (pre- or post-conditions). A condition state changes whenever a user has changed the state of a document (read a document, produced a document, coded a document etc). A condition can have three states as Not finished (0), Iterating (1), and Finished (2).
3. **Concurrent data** Keeps track of concurrent activities/tasks (both concurrent and concurrent iterative activities/tasks). Two states are allowed

for concurrent state: Concurrent (0) and Not concurrent (1).

A more detailed description of the modelling language and the state database can be found in [8].

3 eXtensible Markup Language (XML)

XML is very similar to Hyper Text Markup Language (HTML) in many ways, which is the most popular Web markup language today. HTML has revolutionised the Web, by making it possible for everyone to create hyper-link related document consisting of text, tables, sound and graphics. HTML is very well suited for creating web-pages, but it lacks the capability for specialisation. HTML formats *how* the web-page data will look like, rather than *what* that data represents. HTML is has also only predefined *commands* (tags), and it can therefore not be tailored for specific needs.

XML is more flexible, because you can define your own markup elements. This means that XML makes it possible to tailor the XML documents for different needs, and makes it possible to use XML to represent all kind of data for different purposes. The rest of this section will explain what XML is and how to use it.

3.1 What is XML?

Extensible Markup Language (XML) is a specially design subset of *Standard Generalised Markup Language* (SGML), originally simplified and targeted at the WEB. You can use it to format and transfer data in an easy an consistent way. The syntax of XML is similar to HTML, further explained in next subsection.

3.2 Markup Tags

Tags are used as directives to applications reading XML-text and are enclosed text strings in angle brackets for example `< TAG >`. In HTML, these tags are used to tell the web-browser what colours to use, the size of font, to include images etc. In XML, it is up to the application reading the XML-file, what different tags mean. A small example will show how it can be used:

```
<?XML version = "1.0" ?>
```

```
<DOCUMENT>
<CUSTOMER>
  <NAME>
    <LASTNAME>Smith</LASTNAME>
    <FIRSTNAME>John</FIRSTNAME>
  </NAME>
  <PROFESSION>Student</PROFESSION>
  <PHONENUMBER>1-800-5412</PHONENUMBER>
</CUSTOMER>
<CUSTOMER>
. . .
</CUSTOMER>
</DOCUMENT>
```

The first tag shown in the example above is a processing instruction that tells the application that this document is an XML document and uses XML version 1.0. The rest of the tags in the example are tags that are defined for this example only. The XML-file above structures the data in a document consisting of one or more customers. To create hierarchical structures, a *start tag*, like `< DOCUMENT >`, and an *end tag*, like `< /DOCUMENT >` is used. Start and end tags are used to put data in context and to group data. Between a start tag and an end tag you can either put data or you can put more tags to define a multi-level hierarchy.

3.3 Document Type Declarations (DTD)

Since there are no restrictions for what tags you can define and how to structure these tags, it is usable to define a Document Type Declaration (DTD). The DTD is not strictly necessary in many XML documents, but to make sure that a document is written correctly a DTD is used. An XML processor can first read a DTD, then uses this DTD to check if the XML documents follow the structure define in the DTD. To define a DTD you need to define what tags are valid, what order should the tags go in and what tags can contain other tags. You can say that a DTD defines the syntax for XML-files and the XML pre-processor uses this information to find syntax errors in the XML file. To make it easier to under stand what a DTD is, we will present the DTD for the example presented in section 3.2.

```
<?XML version = "1.0" ?>
<!DOCTYPE document [
  <!ELEMENT document (customer)+>
  <!ELEMENT customer (name, profession, phonenumber)>
  <!ELEMENT name (lastname, firstname)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT firstname (#PCDATA)>
```

```
<!ELEMENT profession (#PCDATA)?>
<!ELEMENT phonenumber (#PCDATA)*>
]>
```

As we can see, the DTD defines what tags are valid, and how the tags are structured. Note that the + symbol means that the item can be repeated one or more times, the * symbol means that the item it refers to can be repeated 0 or more times, and finally the symbol ? means that you can have 0 or 1 element. The example above shows that one *DOCUMENT* can consist of several *CUSTOMER* items, and that a *CUSTOMER* can have several *PHONENUMBERS*.

3.4 Tool support

There are several XML tools available on the market today. You can download most of them from a Website for free. The functionality these tools provide varies from syntax checkers to full-fledged XML parsers that builds up the document structure for example as Java data-structures. Most XML parsers has support for creating unique identifiers within a XML-file, which makes it easier to refer to elements in the XML document.

Also web-browsers have support for XML. Currently, Internet Explorer 4.0 from Microsoft has support for parsing XML documents and generating a hierarchically structured tree representation. In version 5.0 of Internet Explorer and version 5.0 of Netscape, the support for XML has been expanded.

All the major software companies like IBM, Microsoft, Sun, Adobe, Netscape, AT& T are developing XML tools for creating and parsing XML files. For a list of over fifty XML tool implementation, take a look at this web-page [9]. To get an introduction to XML, Steven Holzner's book *XML Complete*[5] is recommended.

4 The experiment

Autumn 1998, three 4th grade students at the Department of Computer and Information Science, at the Norwegian University of Science and Technology (NTNU) started to work on a workflow tool that can guide a user through the Renaissance process step-by-step.

4.1 Workflow tool implementation

These three students assign the the Renaissance workflow project, worked on the prototype for four

months using about 1000 man-hours of work for implementing the system. In this time, they have created:

- A graphical workflow tool, that produces a XML representation of the model [4] as shown in figure 2. This tool was created as a Java-applet using standard Java-classes to draw the graphics and generate XML code.
- A workflow engine, that validates the XML-file, parses through the XML-document and read and changes states of the process model. The workflow engine offers a CGI-interface and was implemented in Perl. A C++ XML parser was used to validate and parse through the XML document. The workflow engine supports also cyclic loops in the process [8].
- A web-based graphical workflow client, that guides the users interactively through the process [7]. This tool was implemented as Java applet communicating with the workflow engine through a CGI-interface.

Figure 2 shows a screen capture of the graphical workflow tool we made. It is a screen capture of two different windows. The main window named *Applet for renMS project* is the modelling tool consisting of several buttons and a screen-area to draw the model (partly covered by another window). To the right in the screen-area we can see an example of the Renaissance process represented as five activities (the boxes). The other window, named *XML for the Renaissance Method* is the result of pressing on the *Show XML* button and is the process model represented in XML generated by the tool.

Although the prototype is not very stable and advanced yet, we were very pleased that we could do so much in this short period of time and with little resources.

4.2 The Renaissance process model represented in XML

We choose to use XML to represent the process model in our workflow prototype. XML was initially chosen, because we wanted to see how well XML was for this purpose and to get an evaluation of practical use of XML.

First we would like to present the Document Type Declaration (DTD) for the description of a task in XML. The DTD was used to define the grammar for our PML as well as making it possible to check syntax and grammar of the process model. Note that the

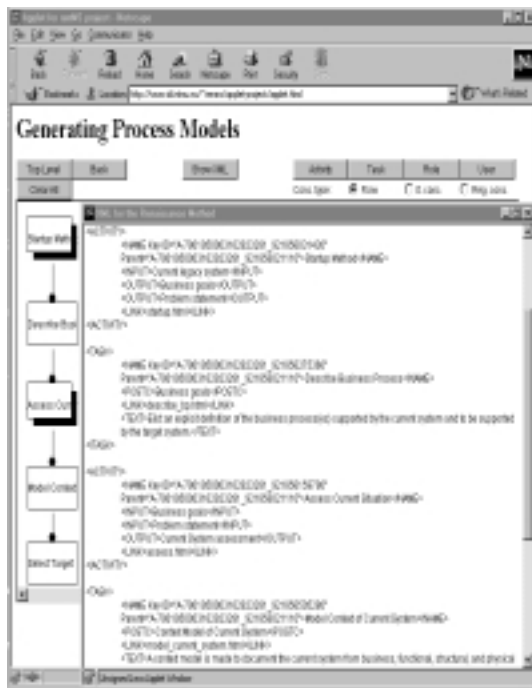


Figure 2: Screen capture from the graphical workflow tool

ID listed in the DTD listing generates a unique ID for the whole XML file.

```
<?XML encoding='UTF-8'>
<!ELEMENT database (activity|task)*>
<!ELEMENT activity (name,
                    (input)*,
                    (output)*,
                    (concurrent)?,
                    (description)?)>
<!ELEMENT task (name,
                (pre-condition)+,
                (post-condition)+,
                (concurrent)?,
                (role)+,
                (user)*,
                (description)?)>
<!ATTLIST activity
  key ID #REQUIRED
  parent IDREF #IMPLIED>
<!ATTLIST task
  key ID #REQUIRED
  parent IDREF #REQUIRED>
...
```

A similar DTD file was also created for *inputs/outputs* as well as for *roles* and *users*. As mentioned above the DTD-files were used to check the grammar and syntax of the XML-files. Another use is to use the DTD file as input for a graphical process modeller tool. What information you can enter

the workflow tool is depended on the definition of the PML found in the DTD-files. In this way, it is possible to change the modelling language without changing the tool.

Now it is time to see how the process is represented in an XML-file. Below, you can see how an activity is described in XML.

```
<database>
<activity key='A0'>
<name>Renaissance method</name>
<input>Current legacy system</input>
<output>Operational target system</output>
<output>New business goals</output>
<output>Revised business process</output>
</activity>
...
</database>
```

5 Experiences

In our project of developing a workflow tool using XML technology, we wanted to see how well suited XML was for representing process models. The last PSEE we built in our research group, EPOS [1, 2, 6], we used Prolog syntax to represent the process model. Actually, Prolog is not very far from XML when it comes to representation of information, but the syntax of XML is simpler for unexperienced users. We found that the main benefits from using XML were:

1. **XML makes it easier for unexperienced users to model their own process models.** This is mainly because XML syntax is similar to HTML, and that XML is rather readable and easy to understand.
2. **XML makes it easier to create workflow engines, since many XML-parsers are already available.** We found that we could build a workflow engine in a relatively short time, because tool support for parsing the XML and building data-structures from XML-data already were available.
3. **XML makes it easier to make the workflow tool available on the web.** This is mainly because XML-tools are implemented in Java or many XML-tools are easy to integrate with a CGI-server. HTML code is also easy to include into XML documents.
4. **XML makes it easier to create graphical modelling tools.** Since Java is an excellent choice for creating graphical modelling

tools (through good graphical support and user-interface support), a Java-based XML processor make the transmission from the Java graphical representation of the model to a XML document easy.

5. **XML makes it easy to change the process model language.** The DTD makes it possible to change the language without changing the whole code for the applications using the XML document. This is possible, since the XML-processor will read the DTD first and then check and build the data-structure for the XML-document.

Although, we were very pleased using XML to represent process models, XML has also one major disadvantages. We found that it was very hard to represent dynamic data using XML. The main reason for this, is that it can be hard and slow to frequently update specific parts of the XML-document (often stored as files in directories). For instance, we did not use XML to represent the states of the process (states of tasks and activities etc.). To do this with XML, we had to read the whole XML-document into a data structure, change some parts of the data structure, and translate the data-structure back to an XML-file. We chose to use small Unix-databases to represent process state, since these databases required small overhead to change its content.

Our experiences with XML indicate that it is suitable for storing and representing the static parts of process models. By static parts, we don't mean parts that will never be changed, but parts that will only be changed once in a while. The DTD will define the syntax of the PML and can be used to check syntax of the XML files. Template process models can also be made by using an almost empty XML-document. It is rather simple to implement tools that create template process models based on process model instances, by removing specific data from a XML document. This can be used later on as a starting point for modelling similar processes.

6 Conclusion

In this paper we have looked at how XML can be used to make it simpler to create workflow tools. First we introduced the background for the experiment than founded the requirements for the prototype we built. Although our mission was to create a workflow tool for a specific process using a specific PML, the approaches we used can be used for other similar projects.

XML is really just a way of organising data. What makes XML so useful is that the XML language itself is not fixed and can be tailored to serve different purposes. It is really up to the one building the application to define *what* tags to put in and *how* to organise these tags. In addition you can use Document Type Declarations to define what tags and how tags must be organised if XML files should be used by an application. Most XML parsers has built in syntax and grammar checker, which is a very useful feature when building a program that uses a textual model as input. Another good reason for using XML is that XML documents are relatively easy to understand for software programs as well as for humans. Since many people are familiar with the syntax of HTML, XML requires little extra effort to understand. All lot of tools are available to make the transition between a XML document and a data-structure as easy as possible.

Generally, we think that XML technology is a good help for researchers in general to make it easier to create prototypes need some kind of model representation. At last for our future prototypes, XML will be used.

7 Acknowledgement

We will give a big thank to Morten Simonsen, Jørgen Andre Brecke and Hans Kristian Fuglenes who have executed experiments using XML to build a workflow tool. In addition, we wish to thank the Renaissance project for giving us an challenging exercise and Mark Greenwood at Information Process Group, University of Manchester, for giving me useful feedback on this paper.

References

- [1] Reidar Conradi, Marianne Hagaseth, Jens-Otto Larsen, and Minh Nguyen. Object-Oriented and Cooperative Process Modeling in EPOS. Technical report, PROMOTER Book 1994, ed. Bashard A. Nuseibeh, Imperial College, 1994.
- [2] Reidar Conradi, Espen Osjord, Per H. Westby, and Chunnian Liu. Initial Software Process Management in EPOS. *Software Engineering Journal (Special Issue on Software process and its support)*, 6(5):275–284, September 1991.
- [3] Renaissance Consortium. *The RENAISSANCE Method handbook*. Published by web, 1998. <http://www.comp.lancs.ac.uk/computing/research/cweg/projects/renaissance>.
- [4] Hans Kristian Fuglenes. WWW technologies for handling process models. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1999.

- [5] Steven Holzner. *XML Complete*. McGraw-Hill, 1998. ISBN 0-07-913702-4.
- [6] Minh Nguyen, Alf Inge Wang, and Reidar Conradi. Total Software Process Model Evolution in EPOS. In *Proceedings ICSE'97*, Boston, USA, May 1997.
- [7] Jørgen Andre Brecke. Design and implementation of the Renaissance process system client. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1999.
- [8] Morten Simonsen. Diploma thesis: Renaissance Multimedia System. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1998.
- [9] XML.COM. XML.COM - XML Implementations. web: http://www.xml.com/xml/pub/Guide/XML_Implementations, 1999. (C) Seybold Publications and O'Reilly and Associates, Inc.