

A COMPONENT BASED WORKFLOW MANAGEMENT SYSTEM FOR
ENACTING PROCESSES DEFINED IN XML

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YALIN YARIMAĞAN

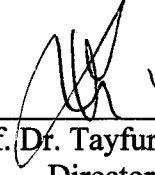
92888

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

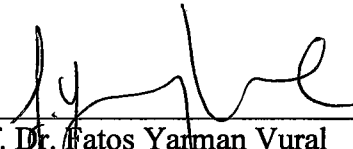
JANUARY 2000

**T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ**

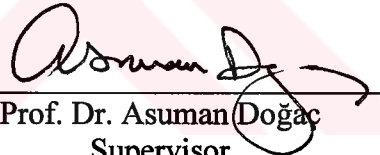
Approval of the Graduate School of Natural and Applied Sciences.


Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master Of Science.

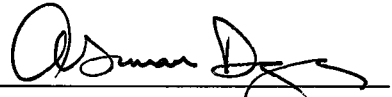
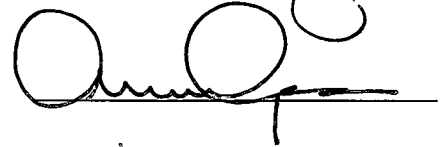

Prof. Dr. Fatoş Yanman Vural
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.


Prof. Dr. Asuman Doğaç
Supervisor

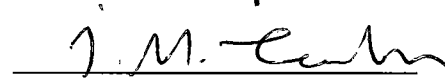
Examining Committee Members

Prof. Dr. Asuman Doğaç

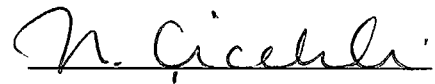



Assoc. Prof. Dr. Aral Ege

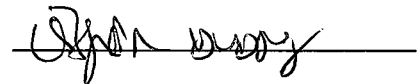
Assoc. Prof. Dr. İsmail Hakkı Toroslu



Assoc. Prof. Dr. Nihan Kesim Çiçekli



Assoc. Prof. Dr. Özgür Ulusoy



ABSTRACT

A COMPONENT BASED WORKFLOW MANAGEMENT SYSTEM FOR ENACTING PROCESSES DEFINED IN XML

Yarımağan, Yalın

M.S., Department of Computer Engineering

Supervisor: Prof. Dr. Asuman Doğaç

January 2000, 81 Pages

In this thesis, a component-based workflow enactment service is developed for executing workflow systems defined in the Extensible Markup Language (XML), conforming to a workflow Document Type Definition (DTD). A workflow DTD is also provided which can be replaced with a standardized one when such a DTD becomes available as a result of the current standardization efforts.

Defining a workflow process in XML provides for the interoperability of applications since XML is both machine processable and human readable. Defining workflow processes in XML and coding a workflow engine in Java to execute those definitions over the Internet provides for high interoperability. This approach proves to be useful especially for electronic commerce applications as in the case of the supply chain automation and integration project currently progressing at the Software Research and Development Center of the Middle East

Technical University. The work described in this thesis is realized as a part of this automation project.

The workflow system is designed to consist of components and thus provides for scalability that is, components are activated only when they are necessary resulting in a small system footprint.

The system is developed in Java to provide for platform independence and thus can be loaded over the Internet and executed. The clients of the system are coded as network-transportable Java applets so that there is no need for the end users of the system to pre-install any software on their computers. This promotes user mobility further as well as easy maintenance of the system components which can be upgraded transparently on the server side.

Keywords: Workflow Management System, XML, DTD, Java, Internet, Electronic Commerce, Supply Chain, Electronic Catalog

ÖZ

XML İLE TANIMLANMIŞ SÜREÇLERİ HAREKETE GEÇİRMEK İÇİN PARÇA ESASLI İŞ-AKIŞI YÖNETİM SİSTEMİ

Yarımağan, Yalın

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Asuman Doğaç

Aralık 1999, 81 Sayfa

Bu tezde, bir iş-akış Döküman Türü Tanımı (DTD)'na uygun olarak Genişletilebilir İşaretleme Dili (XML) ile tanımlanmış iş akışlarını çalıştırmak için parça-tabanlı bir iş-akışı harekete geçirme servisi geliştirilmiştir. Tez kapsamında bir iş-akışı DTD'si de sağlanmıştır. Sağlanan bu DTD, devam etmekte olan standardizasyon çalışmaları sonucunda standart bir DTD ortaya çıktığında onunla değiştirilecektir.

XML, hem makinalar hem de insanlar tarafından anlaşılabilirdiği için, bir iş-akışı sürecini XML ile tanımlamak uygulamaların beraber çalışabilirliğini arttırmak bakımından faydalı olacaktır. İş-akışı süreçleri için XML'de yazılmış tanımları ve bu tanımları İnternet üzerinden çalıştıracak Java tabanlı bir iş-akış motoru sağlamak, beraber çalışabilirliği arttıracaktır. Halen Orta Doğu Teknik Üniversitesi, Yazılım Araştırma ve Geliştirme Merkezi'nde yürütölmekte olan tedarik zinciri otomasyonu ve entegrasyonu projesi gibi elektronik ticaret uygulamaları

buna güzel bir örnektir. Bu tez çalışması kapsamında gerçekleştirilen çalışmalar sözü edilen bu tedarik zinciri otomasyon projesinin bir parçasıdır.

Ölçeklenebilirliği arttırmak amacıyla iş-akış sistemi parçalardan oluşacak şekilde tasarlanmıştır. Sistemi oluşturan her bir parçanın yalnızca gerekli olduğu zaman aktive edilmesi sayesinde o anda kullanılmayan parçaların sistem kaynaklarını gereksiz yere işgal etmelerinin önüne geçilmiştir.

Sistem, platformdan bağımsızlık kazanması için Java kullanılarak geliştirilmiştir. Bu sayede Internet üzerinden yüklenerek çalıştırılabilmesi mümkün olmaktadır. Sistemin istemcileri, ağ üzerinden transfer edilebilir Java-programcıları "Java applets" olarak tasarlandıkları için kullanıcılar makinalarına herhangi bir yazılım kurmadan sisteme erişebilmektedirler. Bu sayede kullanıcıların farklı makinalardan da sistemi aynı şekilde kullanabilmelerine olanak sağlandığı gibi sistemi oluşturan parçaların bakımı sunucu tarafında kullanıcılara hissetirilmeden yapılabilecektir.

Anahtar Sözcükler: İş-Akışı Yönetim Sistemi, XML, DTD, Java, Internet, Elektronik Ticaret, Tedarik Zinciri, Elektronik Katalog

TABLE OF CONTENTS

ABSTRACT	iii
ÖZ.....	v
TABLE OF CONTENTS.....	vii
TABLE OF FIGURES	ix
CHAPTER	
1 INTRODUCTION.....	1
2 RELATED WORK	4
2.1 XML	4
2.1.1 XML Entities.....	11
2.1.2 XML-Link.....	14
2.1.3 XML-Style Sheets.....	17
2.1.4 Namespaces in XML.....	19
2.1.5 XML APIs	21
2.2 WORKFLOW MANAGEMENT SYSTEMS.....	22
2.2.1 ConTract.....	25
2.2.2 METEOR	28
2.2.3 Exotica And FlowMark.....	30
3 MESCHAIN ARCHITECTURE	33
3.1 BASIC CONCEPTS.....	33
3.2 ARCHITECTURE OF THE SYSTEM	38
3.2.1 An open architecture supporting different standards	40
3.2.2 The architecture of catalog agents	41
4 WORKFLOW MANAGEMENT SYSTEM ARCHITECTURE.....	43
4.1 OVERVIEW OF THE PROPOSED ARCHITECTURE.....	43
4.2 COMPONENTS OF THE SYSTEM.....	47

4.2.1	<i>Workflow Domain Manager</i>	48
4.2.2	<i>Workflow Process Object (WPO)</i>	48
4.2.3	<i>Component-Server Repository</i>	49
4.2.4	<i>Workflow Process Definitions Library</i>	50
4.2.5	<i>History Manager</i>	50
4.2.6	<i>Task Manager</i>	50
5	IMPLEMENTATION	52
5.1	TOOLS AND FACILITIES	52
5.2	THE PROGRAM STRUCTURE	53
5.2.1	<i>Loading Process Definitions</i>	54
5.2.2	<i>Executing Workflow Processes</i>	57
5.3	THE DATA STRUCTURES	61
6	CONCLUSION AND FUTURE WORK	65
	REFERENCES	67
	APPENDICES	
A	DTD FOR WORKFLOW PROCESS DEFINITIONS	70

TABLE OF FIGURES

FIGURE

2-1 Weather Report Encoded in XML	7
2-2 XML Document Using Namespace Scoping.....	21
2-3 WfMC's Workflow Reference Model	23
3-1 The Proposed Supply Chain Architecture.....	39
4-1 Workflow Management System Architecture.....	47
5-1 Member Variable Block Type Matchings	62



CHAPTER 1

INTRODUCTION

In this thesis, a workflow enactment service is designed and implemented to execute workflow systems defined in XML conforming to the document type definition that we provide.

The design of the workflow system is original in the sense that it has a component based architecture. The components like basic enactment server, worklist manager are activated only when they are necessary thus resulting in a small system footprint. The system is developed in Java to provide for platform independence and thus can be loaded over the Internet and executed. The clients of the system are coded as network-transportable Java applets so that there is no need for the end users of the system to pre-install any software on their computers. This promotes user mobility further as well as easy maintenance of the system components which can be upgraded transparently on the server side.

A workflow can be defined as a collection of processing steps (also termed as tasks or activities) organized to accomplish some business process. A task can be performed by one or more software systems, by a person or team, or a combination of these. In addition to collection of tasks, a workflow defines the order of task invocation or condition(s) under which tasks must be invoked, i.e. control flow, and data flow between these tasks.

Workflow management is the automated coordination, control and communication of work as is required to satisfy workflow processes.

Workflow Management System (WFMS) is a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic [1]. Workflow Management Systems aim at automating business processes to provide flexibility to cope with on going business changes. Furthermore, WFMSs coordinate and streamline complex business changes within large organizations to achieve improvements in critical, contemporary measures of performances, such as cost, quality, service and speed.

XML [2] has gained a great momentum and is emerging as the standard for self-describing data exchange on the Internet. Its power lies in its extensibility and ubiquity. Anyone can invent new tags for particular subject areas and they define what they mean in document type definitions (DTDs). Content oriented tagging enables a computer to understand the meaning of data. But if every business uses its own XML definition for describing its data, it is not possible to achieve interoperability. The tags need to be semantically consistent across merchant boundaries. One of the efforts in this respect is Common Business Library (CBL) [3][4] by CommerceNet. The CBL consists of information models for various concepts including:

- Business forms, such as catalogs, purchase orders and invoices,
- Standard measurements, such as date and time, location and classification codes.

CommerceNet also intends to provide a workflow DTD. In fact, in many of the business forms defined in CBL, there is a need to invoke a workflow management system. Within the scope of this thesis, a workflow DTD is defined which should be replaced by the standard one when it becomes available.

This thesis is realized as a part of the Electronic Catalog Integration and Supply Chain Automation Project (MESChain) that is currently going on at the Software Research and Development Center, METU. MESChain

uses CBL as its interoperability infrastructure. When it becomes necessary to invoke workflows within the MESChain project, the workflow management system developed within the scope of this thesis should be used.

The thesis is organized as follows; Chapter 2 summarizes the related work about both workflow management systems and the XML technologies. The architecture of the MESChain project is provided in Chapter 3. In Chapter 4, design issues for the Workflow Management System is discussed. Implementation is explained in the Chapter 5 and Chapter 6 concludes the thesis discussing the future work.



CHAPTER 2

RELATED WORK

Our primary goal in developing a workflow enactment system for executing workflow processes defined in XML is to handle the problems of the current workflow management systems. Most obvious problems of such systems are lack of mechanisms for providing heterogeneity and interoperability. By employing XML as the process definition language and by providing a workflow engine coded in Java for executing these definitions makes it possible for enterprises to pass their workflow definitions to each other and execute them without problems.

In this chapter, the technologies and standards that we exploited to realize our goals are summarized. In Section 2.1, Extensible Markup Language (XML) is presented. Later on, general concepts about workflow management systems are discussed and a number of existing workflow systems are briefly summarized in Section 2.2.

2.1 XML

Internet protocol (IP), the Hypertext Markup Language (HTML) and the Hypertext Transport Protocol (HTTP) have revolutionized the manner in which information is distributed, displayed and searched for. Organizations rapidly embraced browsers and search engines with the creation of corporate intranets, then extended these to customers, suppliers, and business partners with extranets [8].

Extensible Markup Language (XML), which complements HTML, promises to increase the benefits that can be derived from the wealth of information found today on IP networks around the world. This is because XML provides a uniform method for describing and exchanging structured data. The ability to describe structured data in an open text-based format and deliver this data using standard HTTP protocol is significant for two reasons. XML will facilitate more precise declarations of content and more meaningful search results across multiple platforms. And once the data is located it will enable a new generation of viewing and manipulating the data.

HTML can be used to tag a word to be displayed in bold or italic; XML provides a framework for tagging structured data. An XML element can declare its associated data to be a retail price, a sales tax, a book title, the amount of precipitation, or any other desired data element. As XML tags are adopted throughout an organization's intranet, and by others across the Internet, there will be a corresponding ability to search for and manipulate data regardless of the applications within which it is found. Once data has been located, it can be delivered over the wire and presented in browsers in any number of ways, or it can be handed off to other applications for further processing and viewing.

XML is a subset of Standard Generalized Markup Language (SGML) that is optimized for delivery over the Web; it is defined by the World Wide Web Consortium (W3C), ensuring that structured data will be uniform and independent of applications or vendors. This resulting interoperability enables a new generation of business and electronic-commerce Web applications.

W3C has created a powerful standard definition of XML to significantly advance how people can take advantage of Web-based data. HTML enables a universal method for displaying data; XML provides a universal method for describing data. There is a rapidly growing need to go beyond simply viewing data, as is the case with Web browsers

today. People need to be able to work with and manipulate the data. And this is what XML allows.

Software corporations are increasingly moving from classic client/server two-tier application models to three-tier models, in which a browser front end interacts with a middle-tier Web server, which in turn communicates with a back-end database server for storage. This three-tier architecture has several benefits over client/server models, including easier scalability and better security. And XML will make possible richer implementation of such models through structured data exchange over HTTP.

The power of XML is that it maintains the separation of the user interface from structured data, allowing the seamless integration of data from diverse sources. Customer information, purchase orders, research results, bill payments, medical records, catalog data and other information can be converted to XML on the middle tier, allowing data to be exchanged online as easily as HTML pages display data today. Data encoded in XML can then be delivered over the Web to the desktop. No retrofitting is necessary for legacy information stored in mainframe databases or documents, and because HTTP is used to deliver XML over the wire, no changes are required for this function.

Once the data is on the client desktop, it can be manipulated, edited, and presented in multiple views, without return trips to the server. Servers now become more scalable, due to lower computational and bandwidth loads. Also, since data is exchanged in the XML format, it can be easily merged from different sources.

XML is valuable to the Internet as well as large corporate intranet environments because it provides interoperability using a flexible, open, standards-based format, with new ways of accessing legacy databases and delivering data to Web clients. Applications can be built more quickly, are easier to maintain, and can easily provide multiple views on the structured data.

XML is a text-based format that lets developers describe, deliver and exchange structured data between a range of applications to clients for local display and manipulation. XML also facilitates the transfer of structured data between servers themselves. Vast stores of legacy information exist today, distributed across disparate, incompatible databases. XML allows the identification, exchange and processing of this data in a manner that is mutually understood, using custom formats for particular applications if needed.

XML resembles and complements HTML. XML describes data, such as city name, temperature and barometric pressure, and HTML defines tags that describe how the data should be displayed, such as with a bulleted list or a table. XML, however, allows developers to define an unlimited set of tags, bringing great flexibility to authors, who can decide which data to use and determine its appropriate standard or custom tags.

In Figure 2-1, an example XML document is presented which describes a weather report.:

```
<weather-report>
  <date>March 25, 1998</date>
  <time>08:00</time>
  <area>
    <city>Seattle</city>
    <state>WA</state>
    <region>West Coast</region>
    <country>USA</country>
  </area>
  <measurements>
    <skies>partly cloudy</skies>
    <temperature>46</temperature>
    <wind>
      <direction>SW</direction>
      <windspeed>6</windspeed>
    </wind>
    <h-index>51</h-index>
    <humidity>87</humidity>
    <visibility>10</visibility>
    <uv-index>1</uv-index>
  </measurements>
</weather-report>
```

Figure 2-1 Weather Report Encoded in XML

This data could be displayed in many different ways, or it could be handed off to other applications for further processing. Style sheets help by transforming structured data into different HTML views for display in a browser, or even to other display formats on other platforms running other applications.

Today, Document Type Definitions (DTDs) may accompany an XML document, essentially defining the rules of the document, such as which elements are present and the structural relationship between the elements. DTDs help to validate the data when the receiving application does not have a built-in description of the incoming data. With XML, however, DTDs are optional.

Data sent along with a DTD is known as "valid" XML. In this case, an XML parser could check incoming data against the rules defined in the DTD to make sure data was structured correctly. Data sent without a DTD is known as "well-formed" XML. Here an XML-based document instance, such as the hierarchically structured weather data above, can be used to implicitly describe itself.

With both valid and well-formed XML, XML encoded data is self-describing. The open and flexible format used by XML allows it to be employed anywhere a need exists for the exchange and transfer of information. This makes it extremely powerful.

For instance, XML can be used to describe information about HTML pages, or it can be used to describe data contained in business rules or objects in an electronic-commerce transaction, such as invoices, purchase orders and order forms. Since XML is separate from HTML, XML could also be added inside HTML documents. W3C is working to define a format by which XML-based data, or "XML islands," can be encapsulated in HTML pages. By embedding XML data inside an HTML page, multiple views could be generated from the delivered data, using the semantic information contained in the XML. Moreover, XML can be

used for such compelling applications as distributed printing, database searches, and others.

XML brings so much power and flexibility to Web-based applications that it provides a number of compelling benefits to developers and users:

- More meaningful searches
- Development of flexible Web applications
 - Data integration from disparate sources
 - Data from multiple applications
 - Local computation and manipulation of data
 - Multiple views on the data
 - Granular updates
- Open standards
- Format for Web delivery
 - Enhances scalability
 - Facilitates compression

The opportunities will expand further as more vertical market data formats are created for key markets such as advanced database searching, online banking, medical, legal, electronic commerce, and other fields.

Customer services are now migrating to Web sites from call centers and physical locations and will therefore benefit from the robust functionality of XML. And, because most of these business applications involve manipulation or transfer of data and database records, such as purchase orders, invoices, customer information, appointments, maps and such, XML will revolutionize end-user possibilities on the Internet by allowing a rich array of business applications to be implemented. In

addition, information already on Web sites, whether stored in documents or databases, can be marked up using XML-based, intranet-oriented vocabularies. These vocabularies also help small and medium-sized corporations that need to exchange information between customers and suppliers.

A vital untapped market is development tools that make it easy for end users to build their own collaborative Web sites, including tools for generating XML data from legacy database information and from existing user interfaces. In addition, standard schemas could be developed for describing portfolios or other data, for example, which could use the layout, graphs and other functions of spreadsheet programs. Declarative and visual tools for describing XML generated from legacy databases are a powerful opportunity.

XML will require powerful new tools for presenting rich, complex XML data within a document; this is done by mapping a user-friendly display layer on top of a complex set of hierarchical data that can change dynamically. Possible layouts to use for XML data include collapsing outlines, PivotTable dynamic views, and a simple sheet for each portfolio.

Web sites offer stock quotes or news articles or real-time traffic data, which can be obtained by filtering from Web broadcasts or by intelligent polling of a tree of servers replicating these sites. Information overload can be avoided with XML by writing custom rules for aging of information, for example, as is done with e-mail. XML-based tools for users to construct these rules and server and client software to execute them are a huge opportunity. A Standard Object Model could enable these functions, typically written in script, to filter incoming messages, examine stored messages, create outgoing messages, access databases and such. These agents can be written to run anywhere automatically.

XML has several issues such as entities, style sheets and namespaces that enable it to accomplish all that is explained up to this point. In the following sub-sections, such major aspects of the XML will be introduced.

2.1.1 XML Entities

XML supports some practical aspects of document management with its concept of *entities*. Entities are sequences of characters or bit patterns, held within an XML document or external to it.

Entities can be used to represent external resources, such as illustrations, with an added notation to indicate their type. XML supports this approach, and it calls such resources *external binary entities*.

In addition, XML offers two types of entities that can be used to help manage XML documents.

External text entities are simply resources (usually, but not necessarily, files) containing XML markup. External text entities allow building XML documents in chunks. For example, a book can be being written one chapter at a time, and each chapter is a separate text entity. When the time comes to make the whole book into a single XML document, all that is to be done is to declare each chapter as an external text entity and make reference to them all in the correct order.

In addition to allowing documents to be written by a number of different people, external text entities allow information reuse. A standard warning, for example, could be marked up in XML and held in a file, ready to be pasted into all new customer documentation.

The ability to build up a document in chunks can also be useful when parts of it are in a different character set. Each part can be isolated and placed in a separate entity with its own encoding rules.

XML also supports the idea of *internal entities*, which are shorthand for a piece of XML markup that is held within the document. Internal

entities can be used for standard phrases, expansions of abbreviations, and even for single characters (such as the trademark symbol, TM).

Internal entities have a value that is given directly in the entity declaration:

```
<!ENTITY XML "Extensible Markup Language">
```

This entity declaration maps the name XML to the content "Extensible Markup Language". No separate storage unit is involved.

External entities, by contrast, refer in their declarations to a storage unit by means of a SYSTEM or PUBLIC identifier:

```
<!ENTITY chapter1 SYSTEM "chap1.xml">
```

This associates the name `chapter1` with the URL "chap1.xml". An XML processor must read the file referenced by that URL in order to find out the content of that entity.

The format for PUBLIC identifiers includes a PUBLIC name before the SYSTEM name:

```
<!ENTITY image1 PUBLIC "-//RBL//NONSGML  
Illustration 1//EN" "images/illus1.gif" NDATA gif>
```

PUBLIC identifiers provide a mechanism for describing an entity, but do not specify any particular storage location. This can be very useful when a particular entity is already widely available. An XML processor may be able to determine, by matching the PUBLIC identifier against a catalog of resources available locally, that it does not need to fetch a new copy of this entity because it is already available.

External entities can be either *text* or *binary*.

A *text entity* contains text data, which is considered to form part of the XML document. When you insert a reference to a text entity, it is treated as though its content actually appears in the text of the document at that point. For example, the entity XML declared as

```
<!ENTITY XML "Extensible Markup Language">
```

means that references to the entity `XML` should be replaced by the phrase “Extensible Markup Language”. Then the markup

```
<p>physical structures in &XML;.</p>
```

is equivalent to the following:

```
<p> physical structures in Extensible Markup Language.</p>
```

A *binary entity* is basically anything that isn't meant to be treated as though it is XML-encoded. In that sense, the name is somehow misleading: A binary entity might be a plain text file, for example.

Each binary entity has to have an associated *notation*. A notation describes a type of resource (which is usually just a file type, such as BMP, GIF or JPG). In fact, having a notation is the one thing that distinguishes a binary entity from a text entity. Notation information forms part of the entity declaration in the DTD;

```
<!ENTITY my.picture SYSTEM "mypic.bmp" NDATA BMP>
```

This line declares that the entity `my.picture` is in the BMP notation; that is, it is a BMP file.

The notation information tells an XML processor the type of the entity, and therefore, allows the processor to decide what to do with it. For example, if an entity contains mathematical equations using the `Tex` notation, an XML browser might call a helper application to format the equations and then display the result inline within the document.

An entity is inserted into XML documents by using `&`, followed by the Name of the entity, followed by a `;`, as in the following example:

```
&pub-status;
```

XML entities might be useful in a large number of situations. Following are some examples;

- Represent non-standard characters within XML documents.
- Function as shorthand for frequently used phrases.

- Hold chunks of markup that might appear in more than one XML document.
- Hold sections or chapters from a large XML document.
- Organize DTD into logical units.
- Represent non-XML resources.

2.1.2 XML-Link

XML provides built-in linking facilities that aim to be powerful enough to cope with most Web-based applications. Three standards, in particular, have influenced the design of hyperlinks in XML;

- *HTML* provides a linking mechanism that is already familiar to everyone who has ever encoded a Web page.
- *HyTime* is an International Standard that contains useful background concepts.
- *Text Encoding Initiative (TEI) Guidelines* provide a concise syntax for specifying complex links.

XML allows the user to state which elements in XML documents are to be interpreted as linking elements. This is done by specifying the special `XML-LINK` attribute. This both asserts that an element is to be treated as a link and specifies what type of link it is. For example, the following code states that this is a simple link;

```
<A XML-LINK="SIMPLE" HREF="http://www.w3c.org/XML/Activity">
```

Every link can have a machine-processible *role*. The role is the meaning of the link, and it tends to be specific to a particular XML application. It can also have a human-readable label.

Finally, the default behaviors can be specified for a link. The `SHOW` attribute indicates whether the resource pointed to by the link is designed to be embedded in the current context, to replace it, or to start

its own new context. The `ACTUATE` attribute indicates whether the user has to take action before anything is done with the link.

HTML-style links are one example of what XML calls *simple links*. Simple links are links that sit in an XML document and point to a single target, or *resource* (which might be anything that can be addressed by a link).

Extended links are a lot less obvious. They can point to any number of targets, and they can live anywhere. In fact, extended links don't need to live inside any of the resources they point to. This might not sound helpful, but in fact it gives the ability to set up bidirectional and even multidirectional links. Also, taking links out of documents should make it easier to manage them (for example, by checking that their target resource continues to exist).

XML lets declaring one or more XML documents as a *link group*, which makes it easier to manage a set of documents that contain a network of links.

The XML-Link specification extends HREF links in the following ways;

- Allowing any element type to indicate the existence of a link
- Defining the precise meaning of the *fragment identifier* (the part of the URL that follows the # or |) in cases when the target of the link is an XML document
- Providing links with human-readable labels
- Providing links with machine-readable labels
- Specifying policies for the context in which links are displayed and processed
- Specifying policies for when links are traversed
- Supporting extended linking groups
- Supporting multi-ended links

The XML-Link specification talks about several concepts. These are briefly introduced below;

- *Resource*: This is any addressable object that is taking part in a link. Examples include documents, parts of (and points within) documents, images, programs, and query results.
- *Linking element*: This is a special element within an XML document that asserts the existence of a link and contains a description of the link's characteristics.
- *Locator*: A character string appearing in a linking element that is the address of a resource and can be used to locate that resource.
- *Title*: This is a caption associated with a resource, suitable for explaining to human users the significance of the part played in the link by that resource.
- *Traversal*: This is the action of using a link to access a resource. Traversal can be initiated by a user action (for example, clicking on a displayed portion of a linking element), or it can occur under program control.
- *Multidirectional link*: This link can be traversed starting at more than one of its resources. Note that being able to go back after following a one-directional link does not make the link multidirectional.
- *Inline link*: This link is indicated by a linking element that serves as one of its own resources. HTML <A> elements are examples of inline links.
- *Out-of line link*: This link is indicated by a linking element that does not serve as one of its own resources (except perhaps by chance). It need not even occur in the same document as any of the resources it serves to connect.

2.1.3 XML-Style Sheets

Although XML is a data oriented structure, it still needs proper ways for displaying its contents, such as the style sheet mechanisms that are widely used for formatting HTML documents.

In principle, the existing CSS1 (Cascading Style Sheets, level 1) standard could be applied to give simple control over the on-screen display of XML documents. CSS1 states its rules in terms of element names, IDs, and so on, which are clearly features of XML documents as well as HTML documents.

CSS1 was specifically designed around HTML, and some of its features rely on HTML-specific encoding practices. For example, the *a linking element* is treated specially so that visited links can look different from unvisited ones. Similarly, the class attribute is treated specially by CSS1: that attribute has to be added to every element in XML DTDs to take advantage of that feature of CSS1. However, this is likely to change. The CSS1 specification states an expectation that it will become more generic: "CSS1 has some HTML-specific parts (e.g., the special status of the `CLASS` and `ID` attributes) but should easily be extended to apply to other DTDs as well." Such changes will clearly make CSS a more attractive style sheet mechanism for XML documents.

XML provides its own means of specifying style sheets (termed XS). Apart from having the advantage of being designed to work with any XML document structure, XS is massively more powerful than CSS1.

XS is designed to control the output of an XML document to the screen, the printed page; or any other two-dimensional display device.

It is essentially a data-driven style mechanism. When an XML document is to be displayed, one or more XS style sheets will be called into action. These might be specified in the XML document or selected by the user.

The processing of the XML document is determined by scanning the XML document's structure and merging it with the *formatting specification* derived from the active style sheets. These instructions are then used to create the flow *objects*, such as paragraphs and tables, which determine the layout of the document. This merging process produces a tree structure of flow objects: the *flow object tree*.

Each flow object has *characteristics*, such as `font-name`, `font-size`, and `font-posture`, that can be specified explicitly or inherited from flow objects further up the flow object tree.

XS has an extremely flexible *core expression language*, based on Scheme. This is a complete programming language, with facilities for doing calculations, testing conditions, and so on. It can be used to build up complex instructions for the processing of individual elements or even characters within an XML document.

At a higher level, XS supports *construction rules* that declare, in effect, what to do with an element. More precisely, they state what flow objects are to be created, and what characteristics each flow object is to have. For example, an element construction rule for the `p` element might specify that a paragraph flow object is to be added to the flow object tree, with these characteristics:

```
font-size: 12pt
first-line-start-indent: 20pt
quadding: left
```

This will cause the characters in the paragraph to be 12-point font size, with a 20-point indent at the start of the first line. The paragraph will be left-justified.

Typically, the low-level formatting instructions will be declared as functions. This means that the top-level instructions can be made much clearer to a reader, as in this example:

```
(element NOTE (STANDARD-NOTE))
```

```
(element EG (MONOSPACED-TEXT))
```

```
(element CODE (UNDERLINED-PHRASE))
```

Because the core expression language is so powerful, it is possible to use nearly any aspect of an XML document's structure to control how it appears. Most construction rules are pitched at the element level, but their behavior can easily be refined by testing properties of the element, such as its attributes and their values or its ancestor elements.

Although by default the whole document will be displayed in its original order, the core expression language actually gives access to the full document structure at any point. Therefore, a related piece of text from elsewhere in the document can be grabbed beforehand. For example, a virtual table of contents can be generated from the chapter headings, and placed at the start of the document while displaying. Also, parts of a document can be suppressed at certain times.

The mathematical functions in the core expression language make it easy to express font sizes, spacing, and so on in relative rather than absolute terms. This means that style sheets can be written in which the user can blow up the whole display in a consistent manner by overriding the base font size.

2.1.4 Namespaces in XML

There are applications of XML where a single document may contain elements and attributes (referred to as a "markup vocabulary") that are defined for and used by multiple software modules. One motivation for this is modularity; if such a markup vocabulary exists which is well-understood and for which there is useful software available, it is better to re-use this markup rather than re-invent it.

Such documents, containing multiple markup vocabularies, pose problems of recognition and collision. Software modules need to be able to recognize the tags and attributes which they are designed to process, even in the face of "collisions" occurring when markup

intended for some other software package uses the same element type or attribute name.

These considerations require that document constructs should have universal names, whose scope extends beyond their containing document. This specification describes a mechanism, *XML namespaces*, which accomplishes this.

An **XML namespace** is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.

URI references which identify namespaces are considered **identical** when they are exactly the same character-for-character. URI references which are not identical in this sense may in fact be functionally equivalent. Examples include URI references which differ only in case, or which are in external entities which have different effective base URIs.

Names from XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. Mechanisms are provided for prefix scoping and defaulting.

URI references can contain characters not allowed in names, so cannot be used directly as namespace prefixes. Therefore, the namespace prefix serves as a proxy for a URI reference. An attribute-based syntax described below is used to declare the association of the namespace prefix with a URI reference; software which supports this namespace proposal must recognize and act on these declarations and prefixes.

A default namespace is considered to apply to the element where it is declared (if that element has no namespace prefix), and to all elements with no prefix within the content of that element. If the URI reference in a default namespace declaration is empty, then unprefixed elements in the scope of the declaration are not considered to be in any namespace. Note that default namespaces do not apply directly to attributes.

An example XML document using namespace scoping is given in Figure 2-2

```
<?xml version="1.0"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
  xmlns:isbn='urn:ISBN:0-395-36341-6'>
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace -->
    <p xmlns='urn:w3-org-ns:HTML'>
      This is a book
    </p>
  </notes>
</book>
```

Figure 2-2 XML Document Using Namespace Scoping

2.1.5 XML APIs

XML documents are parsed or created with XML processors that provide a standard set of APIs that allow access to all parts (elements, attributes, entities, processing instructions, comments and DTD) of an XML document.

There are two major types of XML APIs:

- **Tree-based API:** A tree based API compiles an XML document into an internal tree structure and allows an application to navigate the tree. The Document Object Model (DOM) is such an API.
- **Event based API:** An event based API, on the other hand, reports parsing events (such as the start and end of an element) to an application through call backs. The application implements handlers to deal with various events. Simple API for XML (SAX) is such an API.

2.2 Workflow Management Systems

Workflows are composite activities that typically involve a variety of computational and human tasks and span multiple systems. Workflows arise naturally in heterogeneous environments consisting of a variety of databases and information systems. In a heterogeneous environment, applications autonomously developed at different sites in different languages on different hardware and software platforms need to share information and invoke each other's services. It is common that workflow systems access heterogeneous resources and interoperate with other workflow systems. If a workflow system tries to overcome heterogeneity problem, which can take the form of communication level, platform level or semantic level heterogeneity, within its own architecture, the system becomes very complex and inflexible. Therefore, it is more efficient to base a Workflow Management System (WFMS) framework on a standard middleware that hides some levels of heterogeneity in the environment. The object technology, the distributed computing technology and XML are the enabling technologies to provide necessary communication infrastructure for this purpose.

A partial solution to semantic interoperability problem specific to WFMSs can be obtained by complying to the standards being developed by the Workflow Management Coalition (WfMC). WfMC,

which aims at standardizing the terminology and interoperability between workflow products, defines a Workflow Reference Model.

Most of the available workflow management products have some common functionalities and characteristics because they aim at the same functional target. Until recently there has been no common standard for these products to make them interoperate with each other. In order to provide the communication and interoperation of workflows across different vendor products, a standard workflow specification is necessary. Workflow Management Coalition (WfMC), founded in 1993, is a non-profit, international organization of workflow vendors and analysts. Their objectives include standardization of the terminology and interoperability between workflow products.

WfMC defines a Workflow Reference Model [1] and interface descriptions between the basic components of the Reference Model. Figure 2-3 illustrates the major components and interfaces within workflow architecture of the model.

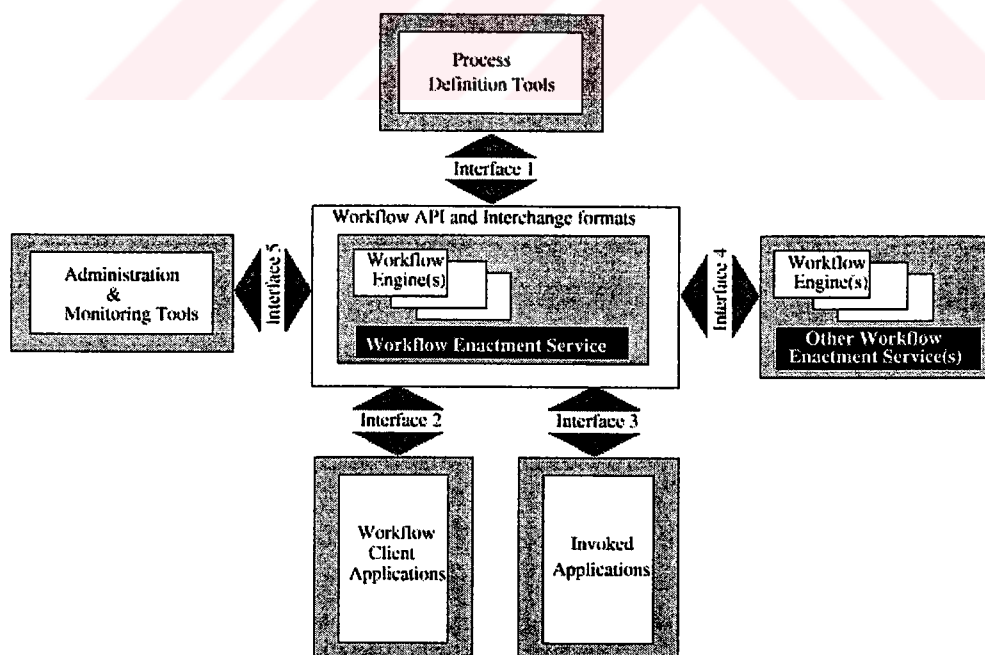


Figure 2-3 WfMC's Workflow Reference Model

The core component in the Reference Model is the Workflow Enactment Service which is responsible for creation, management and execution of workflow process instances according to process definition produced by process definition tools. The workflow enactment software consists of one or more workflow engines, which are responsible for managing all, or part of the execution of individual process instances. It interprets a process definition coming from the definition tool and coordinates the execution of workflow client applications for human tasks and invoked applications for computerized tasks. This may be set up as a centralized system with a single workflow engine responsible for managing all process execution or as a distributed system in which several engines cooperate, each managing part of the overall execution.

In order to achieve interoperability among various WFMS implementations, WfMC has defined five standard interfaces between the components. These interfaces are designated as Workflow APIs and interchange formats. These interfaces are;

Interface 1: Process Definition Tools Interface

The purpose of this interface is to integrate process definitions generated by different process definition tools, or to use a definition generated for a workflow system in another system.

Interface 2: Workflow Client Applications Interface

Users of a workflow system utilize several different types of client applications such as editors, CAD/CAM tools, WWW browsers. This interface provides integration of these applications in order to participate in a workflow system.

Interface 3: Invoked Applications Interface

WFMSs are expected to work with already existing software components such as legacy systems and DBMS applications.

Interface 4: Other Workflow Enactment Services Interface

It may be necessary for different enactment services to interoperate because a process in one enactment service may invoke a process in another.

Interface 5: Administration and Monitoring Tools Interface

An administration and monitoring tool may be a separately developed system, or it may be necessary to centrally monitor different workflow systems.

Process Definition Tools are used to analyse, model, describe and document a business process. The outcome of this process modeling and design activity is a process definition which can be interpreted at run time by the workflow engine(s) within the enactment service. Workflow Client Applications involve the activities which require interaction with the human resources. In the workflow model, interaction occurs between the client application and a particular workflow engine through a well-defined interface embracing the concept of a worklist - the queue of work items assigned to a particular user by the workflow engine. Invoked Applications are the programs invoked by the workflow management system. Administration and Monitoring Tools are used to track workflow process events during workflow process execution.

In the following sub-sections, some of the existing workflow systems are reviewed.

2.2.1 ConTract

One of the first and most advanced workflow research projects is the ConTract project [9]. The focus of this project has been to extend transaction-oriented run-time mechanisms for fault tolerant workflow execution in a distributed environment.

The ConTract model tries to provide the formal basis for defining and controlling long-lived, complex computations, just like transactions control short computations. It is inspired by the mechanisms for

managing flow that are provided by some TP-monitors, like queues and context-databases [10].

The basic idea of the ConTract model is to build large applications from short ACID transactions and to provide an application independent system service, which exercises control over them. As a main contribution, ConTracts provide the computation as a whole with the reliability and correctness properties. The ConTract Model extends the traditional transaction concepts to form a generalized control mechanism for long-lived activities. A large distributed application is being divided into a set of related processing steps which have appropriate consistency and reliability qualities for their execution.

A ConTract can be defined as consistent and fault tolerant execution of an arbitrary sequence of predefined actions (called steps) according to an explicitly specified control flow description (called script). In other words, a ConTract is a program that has control flow like any programming environment, has persistent local variables, accesses shared objects with application oriented synchronization mechanisms and has precise error semantics.

In ConTract programming model, the coding of steps is separated from defining an application's control flow script. Steps are coded without considering implementation details like managing asynchronous or parallel computations, communication, resource distribution, synchronization and failure recovery.

Control flow between related steps can be modeled by the usual elements: sequence, branch, loop, and parallel constructors. *PARFOREACH(query)* statement executes its query in parallel. A sample ConTract *script* can be like:

```
CONTRACT Business_Trip_Reservations
CONTROL_FLOW_SCRIPT
  S1: Travel_Data_Input(in_context: ;out_context: date, from,
to);
  PARFOREACH (airline: EXECSQL select airline from ... ENDSQL)
  S2: Check_Flight_Schedule(in_context: airline, date, from,
to; out_context: flight_no, ticket_price);
```

```

END_PARFOREACH
...
END_CONTROL_FLOW_SCRIPT
...
END_CONTRACT

```

Each of the S_n states is a *step* whose implementation can be coded in a programming language.

Each ConTract step is implemented by embedding it into a traditional ACID transaction, preserving only local consistency for the manipulated objects. Due to not being a transaction, a whole ConTract is not an ACID unit of work.

The ConTract script programmer can define atomic units of work consisting of more than one step by grouping them into sets. Furthermore, the transaction programmer may specify events depending on the result of steps and/or transactions. Grouping of a set of steps into one atomic unit of work is modeled by the following example:

```

TRANSACTIONS
  T1 ( S1, S2 )
  T2 ( S3, S4 )
END_TRANSACTIONS

```

Dependency between the execution results of different steps can also be modeled:

```

DEPENDENCY (T1 abort -> begin T1)

```

ConTract introduces concept of *context elements*, which are ConTract-local variables . These variables are kept in stable transactional storage and are only accessible from the ConTract in which they are defined. Only a step of a ConTract can read or modify the value of context elements. Since the steps are running under the protection of a transaction, their modifications of the context are protected by this transaction. To support the examination of history, context elements are

stored in an update-free way, in which a new version is generated upon a change to the variable.

APRICOTS (A PRototype Implementation of a COnTract System) [11] is an implementation of the ConTract model to show feasibility of its mechanism.

2.2.2 METEOR

METEOR (Managing End To End ORganization) [12] is a workflow language and model based on an extension of the Bellcore workflow model [13]. Its workflow execution model is driven by intertask dependency rules that are expressed in specifically designed script language. METEOR allows workflow definition at two levels of abstraction, by using two different languages: the Workflow Specification Language (WFSL), describing workflow structure and data exchange among tasks, and the Task Specification Language (TSL), describing the details of the tasks.

WFSL is used to specify the workflows, including all task types and classes in a workflow, all intertask dependencies, application level failure recovery and error handling issues. TSL is a declarative rule-based language that mainly deals with task structure, typed input and outputs for each task, and preconditions for every controllable transition. The designer may define *task types*, describing task structures, and *task classes*, that are of a specific type and have typed input and outputs. Three task structures in this model are *transactional*, *non-transactional*, and *open 2PC* task structures. Each task has got *controllable* transitions that can be enabled by the workflow controller, and *non-controllable* transitions that are enabled by the task's processing entity. There are also *compound tasks* which can be composed of *simple tasks* and/or other compound tasks.

One of the key objectives of TSL is the minimal rewriting of existing tasks. TSL provides a wrapper for code describing interaction with an

interface to processing entity and essentially comprises a set of macros that can be embedded in a host language. The main functionality of the TSL macros is to indicate points in the task execution at which the workflow controller can be informed about the current logical state of the task. This functionality allows the workflow to deal with legacy applications without changing their code. Description of a simple transactional task can be :

```
simpleTaskType SIMPLE_TRANSACTIONAL
{
    CONTROLLABLE start(initial, executing) input ;
    NON_CONTROLLABLE abort (executing, aborted) output ;
    NON_CONTROLLABLE commit (executing, committed) output ;
}
```

simpleTaskType denotes that the definition does not relate a compound task; and also two non-controllable transitions produce output.

Each WFSL rule has two components: a control part and an optional data transfer part. Every next step of the workflow is determined by an evaluation of the relevant rules when an event occurs.

Intertask dependencies [14] determine how the tasks in the workflow are coordinated for execution. Two general types of dependencies in METEOR are *state dependencies* and *value dependencies*. State dependencies describe how a controllable transition of a task depends on the current observable states of other tasks. A *state dependency* example takes the following form:

[L1, done] ENABLES [L2, start]

This indicates that the start transition of L2 can be enabled only after L1 has entered the done state. This approach is similar to the transitions in ECA (Event-Condition-Action) rules, however ECA rules use events on the left hand in contrast to *states* in this case. Each state can be the result of one or more events being happened. An example of a rule including state and value dependencies is:

[L1, done] & (success(L1.output) = TRUE) ENABLES [L2, start]

success is a filter function which determines whether L1 has logically completed successfully.

WFSL allows to associate output of a task with input of others. The specification:

L1.output → L2.input;

indicates that output of L1 must be used as input of L2.

2.2.3 Exotica And FlowMark

Exotica Research Project [15], [16] being carried out at the IBM Almaden Research Center brings together industrial trends and research issues in the workflow area. It has got focus on a commercial product called *FlowMark* [17].

Exotica project has got six major research areas: failure resilience in distributed WFMSs, compensation and navigation in workflow networks, high availability through replication, mobile computing, distributed coordination and advanced transaction models.

The system, *Exotica/FMQM* [16], FlowMark on Message Queue Manager, is a distributed workflow system in which a set of autonomous nodes cooperate to complete the execution of a process. Each node functions independently of the rest of the system, the only interaction between nodes is through persistent messages informing that the execution of a step of the process has been completed. In this system, the sequence of events is as follows: a user first creates a process. The process is compiled in the process definition node. After compilation, the process is divided in several parts and each part is distributed to an appropriate node. The division of the process into parts will be based on the users associated with the different nodes and the roles associated with the different activities in the process. Upon receiving its part of the process, a node creates a process table to store this information and starts a process thread to handle the execution of instances of such

process. Finally, the process thread creates a queue for communicating with other nodes all information relevant to instances of the process.

The FlowMark workflow model is a representation of a process, comprising a process diagram and the settings that define the logic behind the components of the diagram. Listed below are the main components of a FlowMark workflow model:

Process: A sequence of activities that must be completed to accomplish a task.

Activity: A step within a process that represents a piece of work that the assigned person can complete by starting a program or another process.

Block: Grouping of several activities and nested blocks to reduce complexity and looping through a series of activities.

Control flow: Determines the sequence in which activities are executed.

Connector: Links activities in a workflow model to define the sequence of activities and the transmission of data between activities.

Data container: Allocated storage for the input and output data of the process and of the activities and blocks within it.

Data structure: An ordered list of variables with a name and a data type.

Condition: The means by which the flow of control in a process can be specified.

Program: A computer-based application program that supports the work to be done in an activity.

Server: A server can be specified for each subprocess, so that a process can be distributed among several servers.

Staff: Each activity in a process is assigned to one or more staff members.

FlowMark provides a graphical definition tool that can be used to model a workflow, by including the above components. There are symbols for activities and blocks, and a process can contain many of these blocks and activities.

The information given in a graphical form can be exported to a textual form. This definition contains declarations for data structures, programs, servers and staff. The process definition gives some information about the process and how the workflow is supposed to execute.



CHAPTER 3

MESCHAIN ARCHITECTURE

The work described in this thesis is part of the METU Supply Chain Automation (MESChain) Project, which is being developed at the Software Research and Development Center of Middle East Technical University. The purpose is to develop a system for "Electronic Catalog Integration and Supply Chain Automation on the Internet".

This chapter is dedicated to the discussion of the MESChain [29] project to be able to put the work described in this thesis into perspective. Section 3.1 will review the basic concepts about the system. The next section will discuss the architecture of MESChain.

3.1 Basic Concepts

Electronic commerce is happening at a very fast pace and business-to-business ecommerce seems to be taking the lead, a very important part of which is the supply chain integration and automation. There is a high demand for well accepted interoperability standards which need to be fitted together for supply chain integration to meet the business demands such as being able to integrate catalogs from different companies. This will facilitate product comparisons and producing customized catalogs. Given an anchor product anywhere on the supply chain, it should be possible to obtain information on related products that complement or add value to this anchor product. Yet another key issue is the full automation of the supply chain processes. However since a single dominant electronic commerce standard is unlikely, the supply chain integration and automation should be able to accommodate

different standards like Open Trading Protocol (OTP) or Open Buying on the Internet (OBI). This will make it possible for users to conform to the standards of their choice.

Another important fact is that rigid supply chains can co-exist with supply chains formed on the fly where participants can transact business spontaneously since the Web is able to make the information instantly available to all trading partners. Facilitating resource discovery that is finding out about the information on possible partners and their catalogs on the Internet and transacting business automatically also becomes an important issue.

The electronic catalogs will much better serve the needs of the business today if they can achieve the following functionalities through seamless interoperation of the resources on the whole supply chain of retailers, distributors and manufacturers [18]:

1. *Facilitating product comparisons and customized catalogs:* Buyers should be able to query multiple catalogs concurrently and then assemble the accumulated information in any format they specify to be able to compare competitive products. In other words, it should be possible to integrate data from a number of different resources to create catalogs that are not only timely and information rich, but also tailored according to the customer's needs and preferences. Also for large organizations that have negotiated special discounts with certain suppliers, interoperable catalog technology should make it possible to create internally distributed product listings that describe approved items and the prices set for them through master purchase agreements.
2. *Locating complementary products:* A buyer may be willing to purchase related products or a product may need several additional components before a complete system is deployed. This necessitates being able to locate compatible products in other catalogs through standardized queries. That is, once a buyer

establishes a core (or "anchor") product anywhere on the supply chain, s/he should be able to readily obtain information from other catalogs describing items and services that complement or add value to this anchor product. For many suppliers this approach is also preferable to the alternative of redirecting customers to complementary catalogs since many of these catalogs may contain pointers to competitive products.

3. *Bidirectional catalog integration*: Catalog integration should be possible not only down the supply chain, that is from retailer to manufacturer but also up the supply chain that is from manufacturer to the retailer. As an example, a distributor's catalog should not only be able to obtain information about products from the original suppliers, but should also acquire information about the retail outlets where products in the catalog are offered. This will make it possible for the user to access all the information available on a particular product, regardless of where s/he has chosen to establish an anchor on the supply chain.
4. *Automation of processes on the supply chain*: Whenever a product is bought, this information should propagate down and up the supply chain automatically triggering a series of distribution, manufacturing and logistics events. As an example, the items collected in a shopping cart should automatically trigger the issuing, approval and delivery of related purchase orders electronically to the appropriate vendor organization. In response an electronic message should be sent to the buyer confirming the acceptance of the transaction, providing tracking number of the transaction and summarizing the status of the order. At the seller site, on the other hand, the related subprocesses like shipment and payment need to be automatically activated. Assuming that the buyer is a customer who contacted a retailer, it is necessary to automatically trigger the processes down the supply chain alerting necessary processes in distributors and manufacturers.

In order to address these needs, MESChain project is addressing the following issues;

1. *Catalog interoperability.*

In MESChain the new catalogs are created in Extensible Markup Language (XML) and already existing catalogs are transformed dynamically into XML, all conforming to the "catalog.dtd" of the Common Business Library (CBL) [3]. XML is a simplified metalanguage, derived from SGML, emerging as the standard for self-describing data exchange in Internet applications. The CBL contains business documents and message formats as XML Document Type Definitions (DTDs). CBL is not a single standard but a collection of common business elements underlying all EDI and Internet Commerce protocols and in this respect is not limiting the architecture proposed; rather CBL provides the much needed interoperability basis among XML documents in the system.

It should be noted that the major database vendors are in an effort to fully support XML and therefore dynamic XML will soon be readily available [19]. However, as a proof of concept, the software is implemented to be used until then, that generates an XML document conforming to "catalog.dtd" of CBL from a query sent to an Oracle database. There is a manually built dictionary to transform the attribute names in the database to the tag names conforming to the ones in "catalog.dtd".

2. *Given an anchor product on the supply chain, discovery of the items and services that add value to this product.*

To realize this feature on the supply chain, the "property" feature of the RDF is used.

Resource Description Framework (RDF) is a foundation for processing metadata for providing interoperability between applications that exchange machine-understandable information. It can be used in a variety of application areas, including resource

discovery to provide better search engine capabilities, cataloging for describing the content and content relationships [20], [21].

The basic data model consists of three object types: *resources* which are the things being described by RDF, *properties* which are specific aspects, attributes or relations describing a resource and *statements* that assign a value to a property of a resource.

Through the "property" feature of RDF, it is possible to state which resource (e.g. a printer) is an addon product for which other resource (e.g. a computer).

3. *Bi-directional traversal on the supply chain.*

The relationships among the catalogs on the supply chain, in other words participant roles, are described through RDF properties which makes it possible for bi-directional catalog search and integration.

4. *An open architecture being able to support different electronic commerce standards.*

It is widely believed that a single dominant ecommerce standard is unlikely. Rather there will be many standards. An ecommerce architecture should be open in the sense that it should be able to support more than one standard at a time. As an example both OBI and OTP are different but similar standards. When the base is XML, since XML is machine processable, it naturally follows that the task of supporting more than one standard should be delegated to the agents and this is the approach taken in MESChain by associating catalog agents to catalogs.

Associating catalog agents with catalogs makes it possible to conduct business according to the different standards. A different workflow process template is defined for each standard, and the catalog agents are informed about messages specific to each of these process templates. Since the agents are able to differentiate between different types of messages and act accordingly, the

proposed architecture provides its users the flexibility to conform to any of the procurement standards they prefer.

5. *Resource discovery on the Web.*

Resource discovery on the Web has been an important yet difficult problem. On the other hand finding out about catalogs is necessary on the supply chain for forming on the fly partnerships.

The architecture proposed helps the users in the following ways in this respect:

- The metadata of the catalogs is expressed in RDF using the Dublin Core element set and hence it becomes possible for resource discovery agents to find out about these catalogs.
- Catalogs are also associated with catalog agents that advertise their capabilities to the facilitators which further help with match making.

6. *Automating the whole process on the supply chain.*

When a catalog agent receives a customer order, it identifies the associated protocol and initiates the corresponding workflow template inside the company's firewall. The companies on the chain can define their workflow templates in XML conforming to workflow.dtd and a workflow engine in Java is provided to execute these definitions.

It is clear from the discussion above that to be able to support different standards and for automating the whole process on the supply chain, there is a need to invoke workflow systems that are highly interoperable and this constitutes the scope of this thesis.

3.2 Architecture Of the System

In the classical supply chain model, the distinctions between retailers, distributors and manufacturers are not clear cut in that manufacturers may also need to buy raw material and distributors or retailers may be

assembling (i.e., producing) products. Therefore in order to come up with a more generic model we assume that all the participants on the supply chain may purchase or produce products that they sell. This removes any differences between a manufacturer and a distributor or a retailer in terms of supply chain functionality.

Figure 3-1 shows the proposed architecture. In a business-to-consumer (B2C) model, a customer either contacts a predetermined supply chain or uses a search agent to find out the potential catalogs. The search agent can dynamically create a catalog for comparative shopping purposes. Catalogs are associated with catalog agents which activate the related workflow template instances. Workflow process instances obtain the necessary information from product descriptions such as stock availability.

In the following, we provide details about those functionalities of the system that make direct use of the workflow management system that is developed within the scope of this thesis.

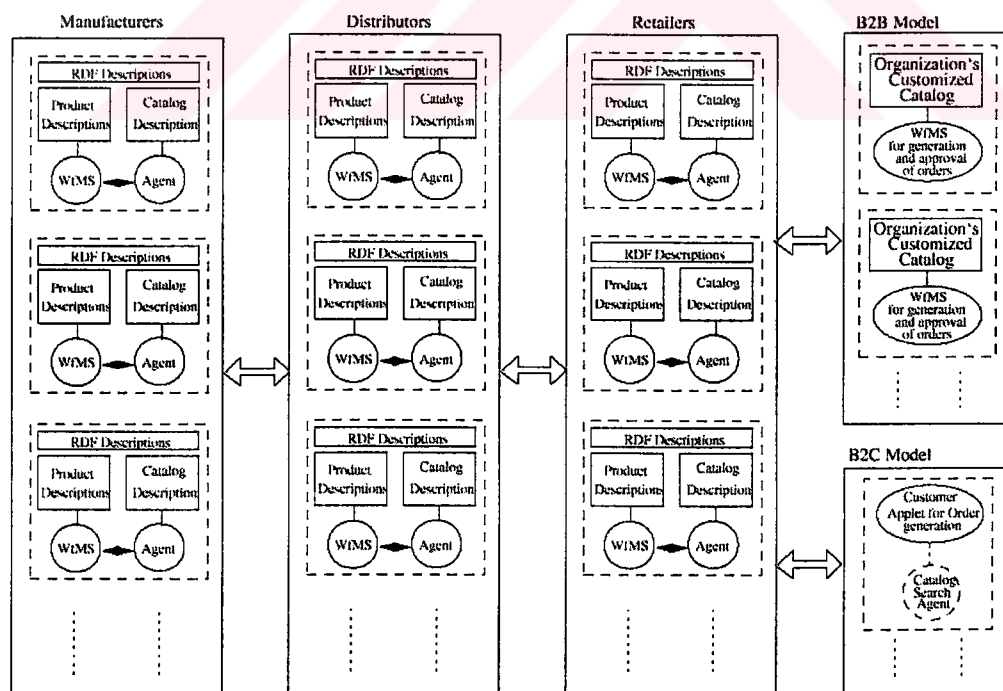


Figure 3-1 The Proposed Supply Chain Architecture

3.2.1 An open architecture supporting different standards

Catalogs have associated catalog agents which initiate a workflow process to handle incoming orders.

There can be different workflow process template definitions conforming to different standards like OTP or OBI. A catalog agent initiates the related workflow template according to the incoming message type. It is possible to accommodate new standards by introducing new workflow templates and by informing the catalog agents about the message formats and the related workflow templates.

The workflow processes are defined in XML conforming to "workflow.dtd" as provided in Appendix A. The organizations on the supply chain can download the engine and execute any workflow process defined in XML conforming to this DTD, which gives an enormous flexibility in terms of interoperability.

The workflow process instances are invoked through the catalog agent which queries the XML order message it receives, and according to the type of the message invokes the related workflow template.

An important point to note here is the need to invoke applications also from an XML document. For example a workflow process invoked by a catalog agent may need to invoke some subprocesses. Currently XML does not interact with external applications. The "NOTATION" facility in XML provides one-way interface, i.e., invoking and sending some information to an external application, but not the other way around, i.e. getting the results back is not supported.

With the current XML technology the following two alternatives can be used to integrate the results of an external database application into an XML document:

- If an autonomous inventory control database is supporting XML, then it can be queried through XML-QL queries that are placed in the XML document [23].

- The XML-QL queries can be used in combination with XML "NOTATION" facility. For example, an external application that accesses the company's autonomous database may be initiated by the NOTATION facility. The external application can store its results in XML to a pre-defined location which can be accessed then by an XML-QL query.

These issues are being addressed within the scope of a related MS Thesis [26].

3.2.2 The architecture of catalog agents

Agents are distinguished from other types of software because they are independent entities capable of completing complex assignments without intervention, unlike tools that must be explicitly manipulated by human users.

In MESChain each catalog is associated with a catalog agent which handles incoming orders or any other XML messages sent to the catalog. The catalog agents have the following properties:

- They accept messages rather than being invoked by other programs.
- They evaluate the messages that they receive and act in response rather than being told what to do, i.e., they are autonomous. Specifically, they are able to handle order requests in different formats to provide for the flexibility of conducting the commerce according to different standards that the user may prefer. For example, a catalog agent can handle the customer orders to the catalog when the requests to the catalog come in diverse formats such as OTP or OBI messages. It should be noted that the catalog and the catalog agent can be outside the company firewall and if the order is accepted, the agent needs to initiate a workflow process inside the firewall to process the customer order. In MESChain, the

workflows invoked within firewalls make use of the system developed in this thesis.

- They find out about the other catalog agents that they need to communicate.
- They keep their state while communicating with other agents.
- The catalog agents can advertise the services offered by the catalog through facilitators (possibly to the marketplaces) to extend their commerce activities as much as possible. Currently catalog agents communicate with each other through an Agent Communication Language, which is specific to MESChain agents. To communicate with other agents in the outside world, a standard agent communication language like KQML should be used [24].

MESChain Agent Communication Language is implemented on TCP/IP. The usage of TCP/IP is preferred over HTTP and FTP for its efficiency. For each catalog agent one predefined socket is assigned where it listens to the incoming messages.

Concurrent access to a single port is handled by Java's Net Package by assigning dynamic ports for each request. Simple message buffering and queuing are also provided by this package. These queuing facilities have been extended to persistent queue implementation and transactional agent communication for safe and consistent transmission.

CHAPTER 4

WORKFLOW MANAGEMENT SYSTEM ARCHITECTURE

As described in the previous chapter, MESChain project requires a Workflow Management System to fully automate the supply chain process. This chapter proposes such a workflow architecture that not only fulfills the requirements of MESChain project but can also be used as a general purpose Workflow Management System.

The system that is proposed will be able to run any workflow process that is defined in XML conforming to the workflow.dtd. The workflow.dtd is given in Appendix A. It should be noted that, for the sake of standardization and common ontology, the provided workflow.dtd will be replaced by a standardized one when it becomes available within the scope of CommerceNet's electronic commerce infrastructure.

4.1 Overview of the Proposed Architecture

There are many business models used in electronic commerce (EC) like e-shop, e-procurement, e-mall, electronic marketplace, virtual communities, value chain integrators, collaboration platforms, and information brokerage [27]. In all of these models, the business processes can be modeled as a set of steps that are ordered according to the control and data flow dependencies among them. This corresponds to a workflow process, where the coordination, control and communication of activities are automated, although the activities themselves can either be automated or performed by humans.

Yet the workflow systems to be used in electronic commerce should have specific features that are of critical importance for electronic commerce applications [28]:

- The mass-business characteristics of EC workflows require a high-throughput workflow execution engine. Thus, load distribution across multiple workflow servers is necessary to ensure this kind of scalability. The EC workflow systems must also quickly adopt to network changes due to failed sites or due to load balancing.
- The EC workflows should easily adapt to different and changing requirements of the customers. So a workflow model is more likely to be a template that is dynamically enriched by introducing additional activities along with their control and data flow, and also possibly skipping the parts of the pre-specified workflow template. Also there could be changes in EC process execution flow triggered by collaborative decision points, or context-sensitive information updates or other internal or external events which necessitate dynamic modification of the workflow instance (e.g., cancelation of an order by the customer).
- Electronic commerce processes should be ubiquitous. To achieve this, they should be able to run in environments with scarce resources, and they should also have an open architecture. That is, the functionality of a workflow system should be tailorable to the needs and available resources of a customer and the system should run on the Internet and should be based on an open and interoperable infrastructure.
- Frequent failures and unavailability of EC workflow servers would immediately weaken the market position of the merchant. This requires efficient replication of workflow servers.

The current monolithic workflow engines on the other hand can not fulfill these needs. In the current systems even for very simple workflow processes, the full-scale engine runs, consuming resources

unnecessarily. These systems also suffer from difficulties in load balancing, migration of workflow process instances, and efficient replication of servers.

Within the context of this thesis, a workflow system architecture based on Internet, XML and Java has been designed to address these issues:

- Each process instance is an object that contains all the necessary data and control information as well as its execution history. This feature makes it possible to migrate the object in the network to provide load balancing. Furthermore, it is possible to dynamically modify the process definition on the instance basis at run time. It should be noted that with this architecture, a site failure affects only the process instances running at that site.
- The system is designed to consist of functional components containing but not restricted to: Basic Enactment Service, UserWorklistManager, Workflow Monitoring Tool, Workflow History Manager, Dynamic Modification Tool, Process Definition Library Manager, Reliable Message Queue Manager, Workflow Domain Manager and Distributed Transaction Manager. This componentwise architecture makes it possible to incorporate the functionality and thus the complexity only when it is actually needed at run time by a process instance by downloading only the necessary components which results in effective usage of system and network resources. It is also possible to add new components or maintain and upgrade the existing components of the system incrementally without affecting the other parts of the system.
- The componentwise architecture facilitates the replication to a great extent. Each site can download its own copy of a component server; also the Domain Manager can be replicated at each site as a Site Manager. This provides for availability and prevents network overhead.

- The clients of the system are coded as network transportable applets written in Java so that the end users can acquire workflow components from the Workflow Domain Manager over the network. Thus it is not necessary to have any pre-installed software on the user computer. This promotes user mobility further as well as easy maintenance of the system components which can be upgraded transparently on the server side.

Another important design issue is the usage of XML for the system infrastructure. Currently there are a lot of commercial systems and research prototypes that are quite successful within themselves. However, the lack of standards about WFMSs prevents them from communicating properly, which is a vital requirement for operations running on such heterogeneous environments.

As indicated in Chapter 2, it is so common for a Workflow Management System that it has to deal with several applications, information systems and other Workflow Management Systems that operate on different hardware and software platforms. The heterogeneity problem might take the form of communication level, platform level or semantic level heterogeneity.

Even though there has been efforts to set the standard to overcome these heterogeneity issues, the outgoing implementations are not satisfactory enough. Most vital example of this issue shows up when it comes to the process definition stage. Every Workflow Management System has its own language for defining workflow processes which makes it cumbersome to adopt a definition written for a specific WFMS to another one, if not impossible. This is a major issue preventing the usage of WFMSs to the extent it should be.

It should be stated at this point that the efforts of WFMC, which are mentioned in Chapter 2 has not been fully successful. The main reason

for this is the fact that the consortium could not agree on the interface standards.

XML is a promising candidate for providing interoperability among different workflow systems. If all workflow processes were defined in XML, conforming to a universally accepted workflow DTD, integration of process definitions among different WFMSs would require no extra effort.

Actually, since any two XML documents conforming to a particular DTD become inherently interoperable, XML might even be used for defining the whole set of interfaces proposed by the WFMC.

4.2 Components of the System

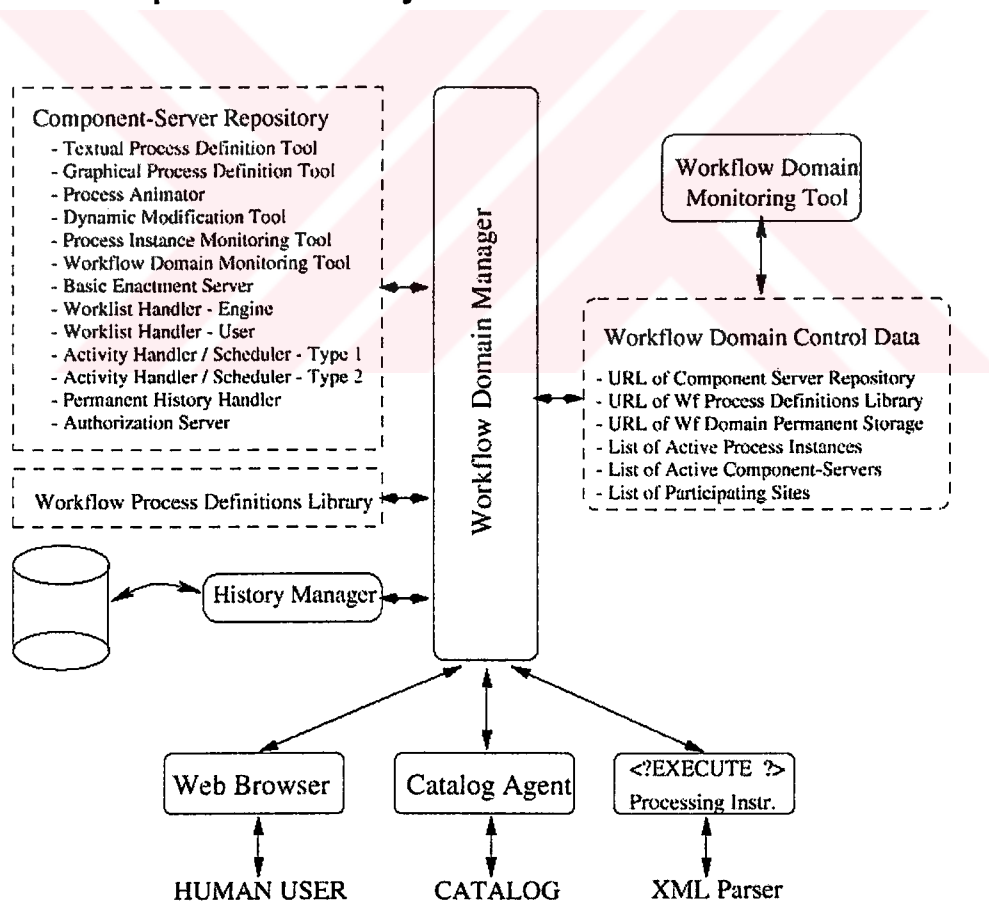


Figure 4-1 Workflow Management System Architecture

The architecture of the workflow system is component based as shown in Figure 4-1. This architecture provides for a complete general purpose adaptive workflow system that participants can use not only for the supply chain automation but for any kind of workflow processes that their business may require. Each participant in the supply chain has a Workflow Domain Manager that runs in close contact with the catalog agent. The Workflow Domain Manager and other components of the workflow system are implemented as Java objects and they communicate with each other through XML messages.

The basic components of this workflow system are presented in the following subsections.

4.2.1 Workflow Domain Manager

The domain manager is the Web server of the system. It communicates via XML messages. For example, a catalog agent sends an XML message indicating which workflow process should be initiated when a customer order is accepted. On the other hand, human clients access to the domain manager via a Web browser that can communicate in XML, and in response to their authorized service requests the domain manager downloads appropriate Java applets to the client which then handles subsequent requests of the same client for that particular service which is provided by a component server. If a client needs a different WfMS service, the domain manager is then accessed again via the Web browser and another Java applet is downloaded. The domain manager keeps runtime information such as list of active process instances, active component servers, list of participating sites, etc. for domain monitoring purposes.

4.2.2 Workflow Process Object (WPO)

When the catalog agent or an authorized user wants to initiate a new instance of a pre-specified workflow process, the Domain Manager

creates a new "Workflow Process Object". The main method of this object is the "Basic Enactment Method" which is activated by the Domain Manager on behalf of the client. The WPO contains all the data (such as workflow process definition, workflow relevant data, enactment history of that instance upto the current execution point, etc.) required to complete the execution of the process instance, or to migrate the process instance from one site to another, or to rescue an instance in case of failures. Since a WPO contains its own copy of the workflow process definition and all the run-time information about its own process instance, the dynamic modification of the workflow process definition on instance basis is simply enabled by dynamically modifying the WPO.

WPOs exist only for active process instances. When a process instance terminates, its WPO is destroyed after its history is permanently saved by the History Manager.

4.2.3 Component-Server Repository

The components of the system are implemented as Java objects and are activated by the domain manager as requested by the executing process instances. The human interaction components like Dynamic Modification tool, on the other hand, are accessed by the authorized users through Java applets. The Component-Server Repository includes the following components for human interaction:

- **Workflow Process Definition Tool:** Authorized users are allowed to define new workflow processes or to delete previously defined processes. Graphical or textual specification interfaces are available. The process definition is syntactically verified and permanently stored in the Workflow Process Definition Library in XML.
- **Dynamic Modification Tool:** Authorized users are allowed to modify a particular workflow process instance at run time to respond to external changes that cause variations in the pre-specified process

definition. In such a case, the modifications can be applied to executing instances selectively or to all instances of the same workflow process if required. The modifications can also be reflected to the template definitions in the Workflow Process Definition Library if needed.

- **Process Instance Monitoring Tool:** Users are allowed to trace workflow process instances they have initiated and extract run-time information about the current execution status of an instance. Collecting and measuring process enactment data are needed to improve subsequent process enactment iterations as well as documenting what process actions actually occurred in what order. This feature provides data to improve optimization and evaluation of processes. Note that an authorized user can monitor any process instance.

4.2.4 Workflow Process Definitions Library

Workflow definitions (i.e., the process templates), organizational role definitions, and participant-role assignments are durably stored into this library. Only workflow process definition tool and dynamic modification tool may insert or update workflow process templates in this library.

4.2.5 History Manager

The History Manager handles the database that stores the information about workflow process instances which have been enacted to completion to provide history related information to its clients (e.g. for data mining purposes). It should be noted that the history of active process instances are stored within the WPO itself.

4.2.6 Task Manager

Another important component of the system is the Task Manager. Even though it is not a direct component of the Workflow Management

System, it has a vital part for the success of operations of the system. Every WFMS needs to execute several tasks which might be of arbitrary complexities ranging from as simple as sending confirmation mails informing the acceptance of purchase orders to logging on to Database Management Systems and querying stock quantities of particular products. Making the issue more complex, most of these tasks are required to execute on remote machines rather than the host running WFMS.

Task Manager is the component responsible from executing tasks. It operates together with a Name Server. On its initializatin phase, Task Manager registers itself to the Name Server, informing the tasks that it is capable of executing. Whenever a Workflow Process Object of the WFMS requires to execute a particular task, it queries the Name Server through XML messages, gathering address information about the most suitable Task Manager that is capable of executing that specific task. Then, using that information, it contacts with that particular Task Manager and submits the task request. The Task Manager interprets the request, extracts the task to execute and the parameters provided. It executes the task and returns any results generated to the WPO.

CHAPTER 5

IMPLEMENTATION

In this chapter, implementation details about the Enactment Service module of the proposed Workflow Management System architecture are provided. First, the tools and facilities that are used during the implementation are briefly introduced. Then, the program structure is overviewed. Finally, data structures that are being employed are discussed.

5.1 Tools and Facilities

The system is developed using the Java language. Java 2 SDK, v1.2.1 from Sun Microsystems is used as the Java platform. Java is chosen as the development platform mainly because of its ability to run on heterogeneous platforms.

An external XML parser is required for the verification process of workflow process definitions written in XML. XMLParser from

5.2 The Program Structure

The main module that is accessible from outside is the Domain Manager, which is basically a process listening to a specific port for incoming requests. An incoming request may either run an existing workflow process from the process library or may execute a new workflow process whose definition is provided in the request.

During the initialization phase, Domain Manager creates an instance of a `processLibrary` object and calls its *loadSignatures* method, which reads the process definition files that reside in a specific directory, extracts process signatures and permanently stores that information in the memory. A procedure signature consists of the name of the procedure, its formal parameters and the return type.

Having successfully created an instance of a `processLibrary` object, Domain Manager becomes ready to accept incoming requests. The rest of the operation executes in a multi-threaded fashion. That is; the Domain Manager creates a server socket which is connected to a specific port of the host it is running on and invokes *listener* objects each of which are responsible for serving one client. Any incoming request from that specific port is handled by one of the free *listener* objects which makes it possible to serve multiple clients concurrently.

Whenever a listener object catches an incoming request, it contacts with the process library and decides whether the request can be accepted or not. If it accepts the request, creates an instance of a Workflow Process Instance object (WPO) and calls its *loadProcessDefinition* method. After successfully loading the process definition, calls the *basicEnactmentService* method of the WPO and returns the generated results to the requester.

In the following sub-sections, details about these two methods are discussed.

5.2.1 Loading Process Definitions

Loading process definitions is a two-stage process. The first part consists of generating the Document Object Model (DOM) from a given XML document and thus making sure that the given definition is verified against the workflow DTD. This guarantees that the program will execute a valid process definition which greatly simplifies the rest of the code by saving a lot of post-controls. The second part consists of traversing that DOM and generating a corresponding structure which will store the necessary program and hierarchy information that will be used during the execution of the workflow process.

5.2.1.1 Creating the DOM

In order to verify a process definition against the workflow DTD, XJParser is employed. As described in the previous section, XJParser is a Java based XML parser which is capable of verifying XML documents against DTDs. XJParser has several methods that can be called in order to generate the Document Object Model (DOM) of a given XML document. Any application that needs to parse an XML document starts with an instance of a *Document* object. Below is a brief summary of the most important properties of the Document object. By using appropriate get/set methods on these properties, XML documents can be parsed;

- *URL*; the URL or the local file path of the XML document
- *validateOnParse*; if the value is false the XML is not validated against any data definition. If the value is true, the appropriate validating parser is set and the XML is validated against available data definition resources such as DTDs.
- *parserClassName*; there are three parser classes available in XJParser for parsing an XML document to create the DOM. These are;

- *Basic Parser* is a high-performance non-validating XML parser. It does not resolve any external entities and does not read default attributes from any DTD.
- *DTDValidatingParser* is a class derived from the *BasicParser* class. It is a full DTD-validating parser. Regardless of the value of the "validateOnParse" property, external entities are resolved, and default attributes are read and parsed as if they were part of the XML. If the "validateOnParse" property is `true`, the XML is validated against the DTD while parsing, and the process stops at any validation error.
- *XMLDOMParser* is the full-featured parser class in *XJParser*. It is derived from the *DTDValidatingParser* class. In addition to the features provided in the *DTDValidatingParser* and *BasicParser* classes, it validates the namespace of the elements and attributes, resolves the data types of the elements and attributes in the original XML document, resolves the schema through the namespace URL of the schema document, and validates the XML against the schema.

For the purposes of our project, whenever a new process definition is to be added to the library, *validateOnParse* and *parserClassName* properties are set to "true" and "XMLDOMParser" respectively, which makes it possible to catch up any possible errors before registering the definition in the library. From then on, unless modified, the definitions are loaded with "false" and "BasicParser" parameters.

5.2.1.2 Creating the Structure

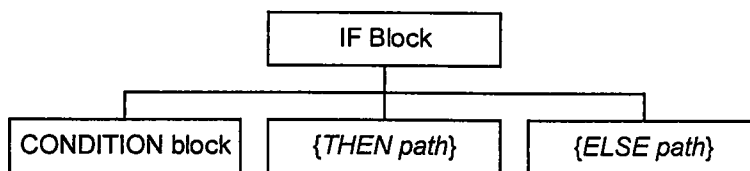
After creating the DOM for the workflow process, the rest of the loading process consists of traversing the DOM and creating a corresponding structure for representing the process definition. This structure will store all the necessary information that will be required during the execution of the workflow process. These include;

- Formal parameter declarations
- Variable declarations
- Process body
- Return values

Type and value information of formal parameters and variable are stored in a *symbolTable* object whose details are provided in section 5.3. Later on, while loading the process body, each referenced variable will be checked against the entries in the symbol table. Non-declared variables and variables referred with non-compatible types will result in compile-time errors.

Process Body stores the rules of the workflow process; that is the order, hierarchy and conditions about executing tasks. A process body tree composed of nodes called *controlBlocks* is created for this purpose. The structure of a *controlBlock* object is discussed in section 5.3.

For each node of the DOM that is a child of the Process Body tagged node, there will be one or more *controlBlock* nodes in the process body tree. For example, for an IF-THEN-ELSE tagged DOM node; following sub-tree will be added to the process body tree;



The interpretation rules for each type of the controlBlock objects is given in the following section. Child nodes labelled *THEN path* and *ELSE path* might consist of arbitrary depth sub trees. For demonstration purposes, suppose the XML document contains the following (simplified) sub part in the process body;

```

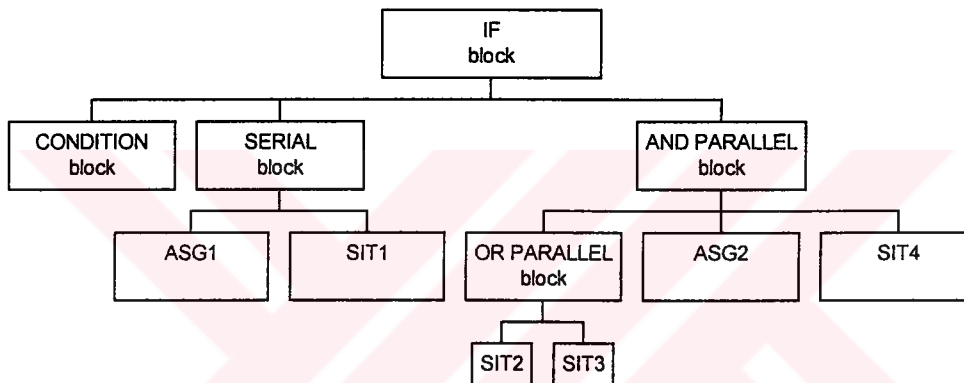
<IF>
  <CONDITION>
  <SERIAL>
    <ASSIGNMENT (ASG1)>
  
```

```

<SERIAL>
  <ASSIGNMENT (ASG1)>
  <SIMPLE TASK (SIT1)>
</SERIAL>
<AND PARALLEL>
  <OR PARALLEL>
    <SIMPLE TASK (SIT2)>
    <SIMPLE TASK (SIT3)>
  </OR PARALLEL>
  <ASSIGNMENT (ASG2)>
  <SIMPLE TASK (SIT4)>
</AND PARALLEL>
</IF>

```

The corresponding structure in the process body tree will be as below;



5.2.2 Executing Workflow Processes

After successfully loading the process definition into the memory, the domain manager calls the Basic Enactment Service (BES) of the WPO which handles the execution of the workflow process.

Before starting the execution, BES loads the actual parameters into the symbol table by calling the *loadActualParameters* method. It then creates an *execTable* object, which stores information about the execution status of the process. Finally, it creates an *executionThread* object and assigns the process body to that thread. Execution thread is capable of executing one node of the process body tree. If the node is a leaf, i.e. has no children node, it simply executes, writes its execution status to the *execTable* and terminates. If the node is not a leaf, it creates threads for its children and starts to wait for the outcome of

those threads. According to that information, it determines whether it is successful or not and informs its own parent.

By this way, the process body is traversed in a multi-threaded fashion. For each level of the process body tree, parallel jobs are carried-out by different threads.

As stated before, each node of the process body tree consists of one *controlBlock* object. There are several different *controlBlock* types, and for each of those, there is a well-defined task to be done. Below is a summary of those tasks;

- *Process Body Block* : The root node of the process body tree is a Process Body block. Process Body blocks are executed similar to serial blocks.
- *Serial Block* : In order to execute a Serial Block, a new execution thread is created for the first child block. If it succeeds, another thread is created for the next child. All child blocks are executed sequentially in this fashion. If any one of the child blocks fail, the whole block fails, otherwise succeeds.
- *AND Parallel Block*: In order to execute an AND Parallel Block, a new execution thread is created for each one of the child blocks. If any one of the child blocks fail, the whole block fails, otherwise succeeds.
- *OR Parallel Block*: In order to execute an OR Parallel Block, a new execution thread is created for each one of the child blocks. If any one of the child blocks succeeds, the whole block succeeds, otherwise fails.
- *XOR Parallel Block*: In order to execute an XOR Parallel Block, a new execution thread is created for each one of the child blocks. If only one of the child blocks succeed, the whole block succeeds, otherwise fails.

- *IF Block*: In order to execute an IF Block, the condition of the block is evaluated. If it evaluates to *true*, then a new thread is created and is assigned to the first child block. If the condition is *false* and the block has a second child, then a new thread is created and is assigned to the second child block. If child block succeeds, the block succeeds, otherwise fails.
- *Iterative Block*: In order to execute an Iterative Block, the condition of the block is iteratively evaluated and each time it evaluates to *true*, a new thread is created and assigned to the child block. If any execution of the child blocks fail, the whole block fails, otherwise succeeds.
- *FOR EACH Block*: In order to execute a FOR EACH Block, a simple block is iterated for every element of a list variable. The simple block can either be a Serial Block or any one of the Parallel Blocks. If any execution of the simple blocks fail, the whole block fails, otherwise succeeds.
- *Task Block*: In order to execute a Task Block, a suitable Task Manager is queried via the Name Server and the task is submitted to that Task Manager. Whether the task succeeds, the block succeeds; the task fails, the block fails.
- *Assignment Block*: In order to execute an Assignment Block, the expression of the block is evaluated. The calculated value of the expression is replaced with the current value of the simple variable pointed to by the *leftHSBlock* pointer of the block.
- *List Assignment Block*: Execution of the List Assignment Block is similar to an Assignment Block. The only difference is that more than one expression needs to be evaluated and each calculated value is appended to a list variable.

As indicated in the previous chapter, one of the essential requirements for a workflow management system is the ability to communicate with external applications. For example, for the purposes of the MESChain Project, suppose there is a workflow process which processes incoming purchase orders. After the acceptance of each valid order, there is a need to contact the stock system and check whether the remaining stock level has fallen below a certain predefined amount or not. If so, the system automatically fires certain actions to get the stock quantity back to a desired level. In order to be able to accomplish this, the workflow management system must be capable of communicating with an external application, a Database Management System in this case.

The proposed architecture makes use of a component called Task Manager for this purpose. The design principles of the Task Manager are similar to those of the Domain Manager. There is a library object called *Task Library* which is responsible for storing information about tasks that the manager is capable of executing. During the initialization, Task Manager creates an instance of a *Task Library* object and registers the types of tasks that it can execute. Task Library stores these information in the memory and registers the Task Manager to a Name Server.

In order to execute tasks, the Task Manager uses objects called drivers. Currently, there are two drivers;

- *Database Driver*; as its name implies, this is a specialized object for executing database related tasks such as running SQL queries and stored procedures.
- *Shell Driver*; this is a specialized object for invoking applications via shell commands such as sending mails and running maintenance programs.

Similar to the Domain Manager, there are *listener* objects for incoming requests. Each *listener* object can handle one client at a time and after verifying from the *Task Library* that the request is valid, it creates an

instance of a *Task* object and calls its *execute* method which makes use of the appropriate driver object to execute the requested task.

5.3 The Data Structures

As indicated in the previous section, a tree structure is constructed in order to store the workflow process definition, which consists of nodes called *controlBlocks*. These blocks are the most basic data structure of the system. A *controlBlock* object has these member variables;

```
private int                controlBlockType;
private controlBlock      parentPtr;
private WPO                owner;
protected boolean        vital;
protected Vector          leftHSVector;
protected String          taskDefinition;
protected condition       conditionPtr;
protected Vector          expressionVector;
protected String          listName;
protected Vector          childControlBlockVector;
protected processCallBlock processCallBlockPtr;
protected String          iterationVar;
protected controlBlock    compensationBlock;
protected controlBlock    contingencyBlock;
```

Only first four of the variables are used in all of the control blocks. The rest of the variables and pointers are used according to the type of the block, which is determined by the *controlBlockType* variable. *parentPtr* is a pointer to the parent block of the current block and is null for the root block. The *owner* is a pointer to the WPO that the current block belongs to. *vital* stores information about whether or not the block has to be completed in order for the workflow process to be successful. The possible values that the *controlBlockType* variable can take are;

```
public static final int CB_PROCESS_BODY          = 1;
public static final int CB_ASSIGNMENT           = 2;
public static final int CB_SIMPLE_TASK          = 3;
public static final int CB_IF_BLOCK             = 4;
public static final int CB_ITERATIVE_BLOCK      = 5;
public static final int CB_LIST_ASSIGNMENT      = 6;
public static final int CB_FOR_BLOCK            = 7;
public static final int CB_COMPENSATION         = 8;
public static final int CB_CONTINGENCY         = 9;
public static final int CB_PROCESS_CALL        = 10;
public static final int CB_SERIAL_BLOCK        = 41;
```

```

public static final int CB_AND_PARALLEL      = 42;
public static final int CB_OR_PARALLEL      = 43;
public static final int CB_XOR_PARALLEL     = 44;

```

Matchings for the rest of the member variables and the block types that make use of them are presented in Figure 5-1;

Variable Name	Block Types
<i>LeftHSVector</i>	Assignment, Process Call
<i>TaskDefinition</i>	Simple Task
<i>ConditionPtr</i>	IF block, Iterative Block
<i>ExpressionVector</i>	Assignment, List Assignment
<i>ListName</i>	List Assignment
<i>ChildControlBlockVector</i>	Process Body, Serial, AND Parallel, OR Parallel, XOR Parallel, IF Block, Iterative, For Each
<i>ProcessCallBlockPtr</i>	Process Call
<i>IterationVar</i>	For Each

Figure 5-1 Member Variable Block Type Matchings

compensationBlock and *contingencyBlock* are two reserved variables which are not being used in the current implementation. They will store information that will be useful for rolling-back the execution in case of failures.

Besides the *controlBlock* object, there are a number of different objects that are used within the system among which *symbolTable*, *execTable*, *expression*, and *condition* are the most important ones. Below is an explanation of the member variables of those objects and how they are used;

- *symbolTable* is a vector which contains objects called *symbolTableEntry*. Each *symbolTableEntry* contains the following member variables;

```
protected String    varName;
protected int      varType;
protected String    varValue;
protected list      listPtr;
protected int       kind;
```

The *kind* variable determines whether the entry is a simple variable or a list variable. If it is a simple variable; *varName*, *varType* and *varValue* are used for storing the name, type and value of it respectively. If it is a list variable, a *list* object, pointed to by *listPtr* stores such information. A list is a simple object with the following member variables;

```
private String      name;
private int         type;
private Vector      value;
```

- *execTable* is a vector which contains objects called *execTableEntry*. Each *execTableEntry* has the following member variables;

```
private int status;
private execTable parentTable;
```

The *parentTable* is a pointer to the *execTable* object which this entry belongs to. The *status* is a flag for determining the current status of the execution of an *executionThread* object. The possible values it can take are;

```
public static final int ETE_NOT_STARTED      = 1;
public static final int ETE_EXECUTING       = 2;
public static final int ETE_COMPLETED       = 3;
public static final int ETE_TERMINATED      = 4;
public static final int ETE_COMPENSATED     = 5;
```

- *expression* is a collection of two vectors, one for storing the operands and one for storing the operators. Possible operators are; addition, subtraction, multiplication and division. An operand is an object with the following member variables;

```
protected int          Kind;
protected String       Name;
protected literal      lPtr;
protected expression   ePtr;
```



```
protected processCallBlock      pcBlockPtr;
```

The *kind* variable determines the type of the operand. An operand can be a variable reference, a literal, an expression, a list reference or a process call. The *name*, *lPtr*, *ePtr* and *pcBlockPtr* variables are used for variable reference, literal, expression and process call typed operands respectively. For list references, the *name* and *ePtr* variables are used together for denoting the name of the list and index to its referenced member.

A *literal* is a simple object for denoting simple literals. Its member variables are;

```
protected int      type;  
protected String  value;
```

Currently supported literal types are;

```
public final static int INTEGER      = 1;  
public final static int STRING      = 2;  
public final static int BOOLEAN     = 3;  
public final static int FLOAT       = 4;
```

- *condition* is a collection of two vectors, one for storing the logical operators and one for storing the operands. Possible logical operators are AND, OR, XOR and NOT. An operand is represented by an object called *simpleCondition* which has the following member variables;

```
private operand    leftOperand;  
private operand    rightOperand;  
private int        comparisionOpr;  
private boolean    negate;
```

If the condition is unary, only the *leftOperand* is used. If it is binary, *rightOperand* and *comparisionOpr* variables are also used. In both cases, the *negate* variable determines whether or not the resulting value should be negated. Possible comparison operators are;

```
public final static int CO_EQUAL     = 1; // equal to  
public final static int CO_GREATER  = 2; // greater than  
public final static int CO_LESS     = 3; // less than  
public final static int CO_GEQ      = 4; // greater or equal  
public final static int CO_LEQ      = 5; // less or equal  
public final static int CO_NEQ      = 6; // not equal
```

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, a component based workflow management system for enacting processes defined in XML conforming to a workflow DTD is described. The work done is realized as part of the Electronic Catalog Entegration and Supply Chain Automation Project that is being developed at the Software Research and Development Center of the Middle East Technical University.

The main contribution of the thesis is to improve the interoperability of a workflow system through the usage of XML as the basis for process definitions. In order to achieve interoperability, there is a need for a structure that can run on heterogeneous platforms, can easily be adopted for different applications and being supported by major software vendors. XML not only meets these requirements but also is an open and self-explaining standart which makes it machine-understandable. In addition, since XML can be transmitted over the HTTP protocol, any computer in the world can receive and send XML documents provided that it has access to the Internet, there is no need for an extra infrastructure. In a work that is currently progressing at the Software Research and Development Center, an XML parser is being developed which will be capable of invoking external applications from within XML documents. Thus, it will be possible to replace specific tags of XML documents with the results of the applications that are being called. By employing this powerful feature in our work, it will be possible to define dynamic workflow processes rather than static ones.

Another major contribution of the thesis is to propose a component based architecture for a workflow management system rather than a monolithic one which provides several benefits such as better utilization of system resources and better management and maintenance opportunities.

There are a number of modification and contributions that can be done to the current implementation of the system as future work;

- As indicated above, it is planned to incorporate the “external application invoking” capability to the system.
- The current implementation makes use of a DTD that is provided by the SRDC. However, in order to achieve world-wide interoperability, that DTD should be replaced with a standard one whenever one becomes available as a result of the ongoing standardization efforts carried-out by the international organizations like CommerceNet.
- Current implementation of the workflow system is not fault-tolerant. That is, whenever a failure occurs at any step during the execution of a process, the whole process fails. It is necessary to incorporate fault-tolerance into the system. For this purpose, the process definitions should be enriched by side-ways such as contingency and compensation blocks which should provide alternative paths to take for the program in case of failures.
- The system currently supports only primitive data types. It might be necessary to support the usage of complex data types such as user-defined structures and arrays in order to increase the efficiency of processes defined.

Furthermore, the system being a prototype to demonstrate the feasibility of a component based architecture lacks some further components of a full-fledged WFMS such as a history manager, worklist manager or monitoring tool which might be developed as future work.

REFERENCES

- [1] D. Hollingsworth, "The Workflow Reference Model", *Workflow Management Coalition Specification*, TC00-1003 (Draft 1.0), 1994.
- [2] "Extensible Markup Language (XML) 1.0". *W3C Recommendation*. <http://www.w3.org/TR/REC-xml-19980210>, 1998.
- [3] R.Glushko, J.M. Tenenbaum, and B. Meltzer, "An XML Framework for Agent-Based E-Commerce", *Communications of the ACM*, 42(3), 1999.
- [4] VEO Systems Inc., <http://www.veosystems.com>, 1998.
- [5] OBI, "Open Buying on the Internet", <http://www.openbuy.org/>, 1998.
- [6] OTP, "Open Trading Protocol", <http://www.otp.org/>, 1997.
- [7] RosettaNet, <http://www.rosettanet.org/general/finished-project/laptop.html>, 1998.
- [8] Microsoft, "XML: Enabling Next-Generation Web Applications", <http://msdn.microsoft.com/xml/articles/xmlwp2.asp>, 1999.
- [9] H. Watcher and A. Reuter, "The ConTract Model", In *Transaction Models For Advanced Database Applications*, Chapter 7, Morgan-Kaufmann, February 1992.
- [10] U. Dayal, M. Hsu and R. Ladin, "Organizing Long Running Activities with Triggers", In *Proc. of the ACM SIGMOD*, 1990.
- [11] F. Schwenkreis, "APRICOTS - A Prototype Implementation of a ConTract System: Management of the control flow and the communication system", In *Proc. of the 12th Symposium On Reliable Distributed Systems*, 1993.

- [12] N. Krishnakumar and A. Sheth, "Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations", In *Distributed And Parallel Databases*, Vol. 3, No. 2, April 1995.
- [13] M. Rusinkiewicz and A. Sheth, "Specification and Execution of Transactional Workflows", In *Modern Database Systems: The Object Model, Interoperability and Beyond*, W. Kim (ed). Addison-Wesley, 1994.
- [14] P. Attie, M. Singh, A. Sheth and M. Rusinkiewicz, "Specifying and Enforcing Intertask Dependencies", In *Proc. of the 19th International Conference on Very Large Databases (VLDB'93)*, 1993.
- [15] C. Mohan, G. Alonso, R. Gunthor and M. Kamath, "Exotica: A Research Perspective on Workflow Management Systems", In *Data Engineering*, Vol.18, No.1, March 1995.
- [16] G. Alonso, D. Agrawal, A. Abbadi, C. Mohan, M. Kamath and R. Guenthoer, "Exotica/fmqm: A Persistency Message-Based Architecture for Distributed Workflow Management ", In *Proc. of the IFIP Working Conference on Information Systems Development for Decentralized Organizations (pp 1-18)*, Trondheim, Norway.
- [17] "FlowMark: Programming Guide", IBM Document No. SH19-8240-01, February 1996.
- [18] CommerceNet, "Catalogs For the Digital Marketplace", Note 97-03, 1997.
- [19] Oracle, "XML Support in Oracle8i", http://www.oracle.com/xml/documents/xml_tmp/, 1999.
- [20] "Resource Description Framework (RDF) Model and Syntaz Specification", *W3C Working Draft*, <http://www.w3.org/TR/WD-rdf-syntax>, 1998.
- [21] "Resource Description Framework (RDF) Schema Specification", *W3C Working Draft*, <http://www.w3.org/TR/WD-rdf-schema>, 1998.

- [22] "Dublic Core Metadata Element Set", <http://purl.org/DC/>, 1998.
- [23] A. Deutsch, M. Fernandez, D.Florescu, A. Levy, and D. Suciu, "XML-QL: A query language for XML", *W3C Document*, <http://www.w3.org/TR/NOTE-xml-ql>, 1998.
- [24] Y. Labrou and T. Finin, "A Proposal for a New KQML Specification", *Technical Report TR-CS-97-03*, University of Maryland, 1997.
- [25] DataChannel, "XJParser Developer's Guide", <http://xdev.datachannel.com/downloads/xjparser/documentation>, 1999.
- [26] E. Sevinç, "Invoking Applications From XML", Department of Computer Engineering METU, MS Thesis, in preparation.
- [27] P.Timmers, "Internet Electronic Commerce Business Models", <http://www.ispo.cec.be/ecommerce/busimod.htm>
- [28] P. Muth, J. Weissenfels, and G. Weikum, "What Workflow Technology Can Do for Electronic Commerce", in *Current Trends in Database Technology*, A. Dogac, T. Ozsu, O. Ulusoy, editors, Idea Group Publishing, 1998.
- [29] İ. Cingil, "Electronic Catalog Integration and Supply Chain Automation on the Internet", Department of Computer Engineering METU, Ph.D Thesis, in preparation.

APPENDIX A

DTD FOR WORKFLOW PROCESS DEFINITIONS

In this appendix, the proposed DTD for the workflow processes is given. For any process to be executed by our system, it must be possible to verify its definition against this DTD.

```
<!ENTITY %remarks "#PCDATA">
<!ENTITY %req.name.attrib "name CDATA #REQUIRED">
<!ENTITY %opt.name.attrib "name CDATA #IMPLIED">
<!ENTITY %operand "var.ref | literal | expression | list.ref |
    process.call">
<!ENTITY %activity "assignment | list.assignment | simple.task
    | simple.block | if.block | iterative.block |
    for.each.block | process.call">

<!ELEMENT workflows (process.defn)>

<!ELEMENT process.defn (formal.parameters?, variables?,
    process.body, return.value?)>
<!ATTLIST process.defn
    %req.name.attrib;
    type (integer | string | boolean | void) "void">

<!ELEMENT formal.parameters (parm.defn)+>
<!ELEMENT parm.defn (%remarks;)>
<!ATTLIST parm.defn
    %req.name.attrib;
    mode (IN | OUT | INOUT) "IN"
    type (integer | string | boolean) "integer">

<!ELEMENT variables (var.defn | list.defn)+>

<!ELEMENT var.defn (%remarks;)>
<!ATTLIST var.defn
    %req.name.attrib;
    type (integer | string | boolean | float) "integer"
    initial.value CDATA #REQUIRED>

<!ELEMENT list.defn (%remarks;)>
<!ATTLIST list.defn
    %req.name.attrib;
```

```

    type (integer | string | boolean | float) "integer">
<!ELEMENT return.value (expression)>
<!ELEMENT process.body (%activity;)*>
<!ATTLIST process.body
    %opt.name.attrib;>
<!ELEMENT simple.task (execute.block, compensation?)>
<!ELEMENT execute.block (#PCDATA)>
<!ELEMENT compensation (%activity;)+>
<!ELEMENT simple.block ((%activity;)+, compensation?)>
<!ATTLIST simple.block
    %opt.name.attrib;
    type (SERIAL | AND_PAR | OR_PAR | XOR_PAR) "SERIAL">
<!ELEMENT list.assignment (expression)*>
<!ATTLIST list.assignment
    %req.name.attrib;>
<!ELEMENT assignment (leftHS, (expression)*)>
<!ELEMENT leftHS (var.ref | list.ref)>
<!ELEMENT var.ref (%remarks;)>
<!ATTLIST var.ref
    %req.name.attrib;>
<!ELEMENT list.ref (expression)>
<!ATTLIST list.ref
    %req.name.attrib;>
<!ELEMENT expression ((%operand;), (operator, (%operand;))*)>
<!ELEMENT operator EMPTY>
<!ATTLIST operator
    kind (+ | - | * | /) #REQUIRED>
<!ELEMENT literal (integer | string | boolean)>
<!ELEMENT integer (#PCDATA)>
<!ELEMENT string (#PCDATA)>
<!ELEMENT boolean EMPTY>
<!ATTLIST boolean
    value (TRUE | FALSE) #REQUIRED>
<!ELEMENT for.each.block (simple.block, compensation?)>
<!ATTLIST for.each.block
    varName      CDATA #REQUIRED
    listName     CDATA #REQUIRED>
<!ELEMENT if.block (condition, simple.block, simple.block?)>
<!ELEMENT iterative.block (condition, simple.block,
    compensation?)>
<!ELEMENT condition (simple.cond, (logical.opr, simple.cond)*)>
<!ELEMENT simple.cond ( (%operand;), comparision.opr,
    (%operand;))>
<!ATTLIST simple.cond

```



```
truth.value (NOT | SELF) "SELF">

<!ELEMENT comparision.opr    EMPTY>
<!ATTLIST comparision.opr
  type (EQUAL | GREATER | LESS | GEQ | LEQ | NEQ)
#REQUIRED>

<!ELEMENT logical.opr    EMPTY>
<!ATTLIST logical.opr
  type (AND | OR | XOR) #REQUIRED>

<!ELEMENT process.call (expression | leftHS)*>
<!ATTLIST process.call
  %req.name.attrib;>
```

