Fall 1-31-2000

Collaborative software agents support for the texpros document management system

Jrtian Lin
New Jersey Institute of Technology

ABSTRACT

COLLABORATIVE SOFTWARE AGENTS SUPPORT FOR THE TEXPROS DOCUMENT MANAGEMENT SYSTEM

by
Jrtian Lin

This dissertation investigates the use of active rules that are embedded in markup documents. Active rules are used in a markup representation by integrating Collaborative Software Agents with TEXPROS (abbreviation for TEXt PROcessing System) [Liu and Ng 1996] to create a powerful distributed document management system. Such markup documents with embedded active rules are called Active Documents. For fast retrieval purposes, when we need to generate a customized Internet folder organization, we first define the Folder Organization Query Language (FO-QL) to solve data categorization problems. FO-QL defines the folder organization query process that automatically retrieves links of documents deposited into folders and then constructs a folder organization in either a centralized document repository or multiple distributed document repositories. Traditional documents are stored as static data that do not provide any dynamic capabilities for accessing or interacting with the document environment. The dynamic and distributed nature of both markup data and markup rules do not merely respond to requests for information, but intelligently anticipate, adapt, and actively seek ways to support the computing processes. This outcome feature conquers the static nature of the traditional documents.

An Office Automation Definition Language (OADL) with active rules is defined for constructing the TEXPROS's dual modeling approach and workflow events representation. Active Documents are such agent-supported OADL documents. With

embedded rules and self-describing data features, Active Documents provide capability of collaborative interactions with software agents. Data transformation and data integration are both data processing problems but little research has focused on the markup documents to generate a versatile folder organization. Some of the research merely provides manual browsing in a document repository to find the right document. This browsing is time consuming and unrealistic, especially in multiple document repositories. With FO-QL, one can create a customized folder organization on demand.

COLLABORATIVE SOFTWARE AGENTS SUPPORT FOR THE TEXPROS DOCUMENT MANAGEMENT SYSTEM

by
Jrtian Lin

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

January 2000

APPROVAL PAGE

COLLABORATIVE SOFTWARE AGENTS SUPPORT FOR THE TEXPROS DOCUMENT MANAGEMENT SYSTEM

Jrtian Lin

Dr. Peter A. Ng, Dissertation Advisor — Date
Chair and Professor of Computer Science, University of Nebraska at Omaha

---

Dr. Gary Thomas, Dissertation Co-Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. D.C. Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Franz Kurfess, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. Taiming Chu, Committee Member — Date
Assistant Professor of Mechanical Engineering, NJIT

Dr. Ronald Curtis, Committee Member — Date
Assistant Professor of Computer Science, William Paterson University

BIOGRAPHICAL SKETCH

Author: Jrtian Lin

Degree: Doctor of Philosophy in Computer and Information Science

Date: January 2000

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer and Information Science
  New Jersey Institute of Technology, Newark, NJ, 2000

- Bachelor of Science in Electrical Engineering
  National Tsing-Hua University, Hsinchu, Taiwan, 1992

Major: Computer and Information Science

Presentations and Publications:

Lin J.T., Shen H., Chu T.M., Doong S., Curtis R., Hung D.C., and Ng P.A., 1999, *Folder Organization Query Language,* Proceedings on the Fourth World Conference on Integrated Design and Process Technology (IDPT), Kusadasi, Turkey, June 27-July 2, 1999.

Shen H., Lin, J.T., Chu T. M., Tanik M., Curtis R., Hung D.C., and Ng P.A., 1999, *Automatic Authoring in HyTEXOROS,* Proceedings on the Fourth World Conference on Integrated Design and Process Technology (IDPT), Kusadasi, Turkey, June 27-July 2, 1999.

Lin J.T., Chu, T.M., Doong S., and Ng P.A., 2000, *Active Documents: Active Rules Embedded in Makup Documents,* Proceedings on the Fifth World Conference on Integrated Design and Process Technology (IDPT), Dallas, Submitted for publication.

To Mother and Laurie

ACKNOWLEDGEMENT

First, I extend my deepest gratitude to my advisor, Professor Peter A. Ng. Dr. Ng is the most erudite and articulate person I have ever had the pleasure to work with. He is also a patient and nurturing mentor, who could always boost my confidence in times of self-doubt and frustration. I am honored to have his name on this work. I thank the members of my extraordinary committee, Professor Gary Thomas, Professor D.C. Hung, Professor Franz Kurfess, Professor Taiming Chu, and Professor Ronald Curtis. Professor Gary Thomas is my co-advisor in this work. I am honored to have the benefit of such a diverse, generous, and helpful panel.

I thank Ms. Miriam Senator for her help by editing my dissertation.

I thank my uncle Ben Lin for supporting my return to United States and graduate school. Without his support, I would never be able to study here.

I thank my mother and my sisters, for their life-long love of educating.

Finally, but most earnestly, I thank my wife, Chien-Jung Huang, to whom both this work and my future are dedicated.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF TABLES
(Continued)

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

A document management system is best suited for an Internet environment that integrates metadata markup technology, document knowledge discovery, data mining, Web technology, and heuristic decision-making processes. A document usually is structured in a certain format according to a user's need and from its structure information significant to the user can be extracted. This structured information, which can be used to represent its original corresponding document, can be organized to form a frame instance [Liu and Ng 1996] of a document type. Each document type can be characterized by a group of attributes to form a frame template, which corresponds to a document class. Various classes of document types can be organized as a structural hierarchy.

TEXPROS [Liu and Ng 1996] employs frame templates to describe the classes of office documents as document types. Upon the arrival of a document, the frame template of its document type is used as a basis for extracting information to form a frame instance, which is the synopsis of the document associated with its type (represented by a frame template). Based on the nature of their content, different frame instances can co-exist as a group (say a folder). Analogously, an approach to manage structured documents is to develop a generic object schema to represent a collection of documents in a class. The schema must model the basic hierarchical structure (classes are organized in a hierarchy) and offer extensibility of the structure through subclass overrides. Each schema has information contained in the document, and provides facilities for creating arbitrary links between this information and other documents. This object schema model

provides users with a framework to describe various document types. Assume that we want to treat each document as having active features such as those of a modern computer program. For example, developing such a facility to manage active features is of particular need for managing a document flow system. During a document flow process, however, each document can dynamically change its state as it interacts with its local environment. Such interaction comes from a software agent that provides collaborative interactions with this document.

Collaboration is a major feature of an active system, which is made possible by software agent technology. With software agent technology, we can embed active rules in documents to provide more dynamic features. Collaboration combines Event/Condition/Action to take the documents' behavior beyond traditional static actions. The asynchronous authoring and publishing process can be done by dynamic collaboration with the help of software agents. The early work of the Collaborative Software Agent was first proposed with the concurrent actor model [Hewitt 1977]. The Collaborative Software Agent [Nwana and Azarmi 1997] with artificial knowledge is used to trigger the events and generate the actions. In the concurrent actor model, the concept of a self-contained, interactive and concurrently active object is referred to as an "actor". This software entity encapsulates internal states and can respond to messages from the outside world.

1.1 Background Information

TEXPROS employs a dual model paradigm to handle documents. It consists of a Document Type Hierarchy and a Folder Organization [Wang and Ng 1992]. The

Document Type Hierarchy exploits structural commonalties between frame templates. Such a hierarchy helps classify documents into various document types. The Folder Organization mimics the user's real-world document filing system and provides the user with an intuitively clear view of his/her document structure. Such a view facilities document retrieval and filing activities [Liu and Ng 1996].

The frame template-based model uses frame templates as data formats for describing structured documents in terms of frame instances. These data formats can be described using a markup representation such as eXtensible Markup Language (XML)[Connolly and Bosak, 1998]. A frame template can be described by a Document Type Definition (DTD) and its corresponding frame instances, which is referred to as structured documents, and can be stored as markup documents. A markup document can contain a large amount of information or data embedded within two quoted tags, a start-tag and an end-tag. Any data enclosed by a start-tag and an end-tag is referred to as an element. The start-tag is delimited using the characters '<' and '>'. The end-tag is delimited using the characters '</' and '>'. For example,

```
<sender>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</sender>
```

An element is defined using the semantic meaning for the data, such as <sender>. It can be recognized easily as the author of the document. The markup text <sender> recursively contains two markup elements <firstname>John</firstname> and <lastname>Smith</lastname>. Such a well-defined notation for a markup document can

be used to distinguish between communication documents and users, or between documents and software entities. The semantic tag can be used as the metadata of the TEXPROS's frame instance. The semantic tag can also include links, which are used to specify the aggregation of documents. As an example, frame instances that share some predefined common characteristics are collected as groups (called folders). A folder organization [Fan and Ng 1998] can be viewed as a facility that provides links for connecting various groups of frame instances. The folder organization is a hierarchical view of groups of frame instances in the frame instance base.

In general, to model a document process electronically, office automation needs to access documents of various types for communication purposes and is associated with highly sophisticated and rule-based software agents. Consider the case of organizing a conference meeting. There are a large number of documents of different types used to exchange information. These document types include memos, letters, calls for papers, program announcements, advanced programs, registration forms, paper evaluation forms, business to business transactions, business to customer transactions, and so forth. A frame template represents each of these document types. However, to deal with the variety of documents, they should be defined precisely and consistently. As a result, CSAs can apply to all of them regardless of the different domains of application. To accomplish this, a meta-language system is needed to define all document types. In this dissertation, the Office Automation Definition Language (OADL) is proposed, which is a knowledge-embedded descriptive language. OADL provides techniques for describing structured documents such as frame instances, specifying predicates to form a folder organization, and specifying active rules to activate the documents. It is a meta-language

for implementing various frame templates, each of which represents a class of documents as a document type.

Therefore, metadata is embraced by using human readable semantic "data" to describe "data". Semantics can be broadly defined as "the scientific study of the relations between signs and symbols and what they denote or mean" [Wood 1985]. For example, a library catalog describes a book as a record. Records can be created in descriptive cataloging [Weibel et al. 1995]. A book record can be created by manually filling in each field with data. These fields can be author, title, publisher, ISBN, subject, and location identification. They are treated as a unit and are often called attributes or collectively, metadata. For both machines and humans, metadata is understandable information that is both unambiguous and meaningful. Metadata should have a precise, predictably interpreted meaning, which can be used by software agents for their well-defined semantic structures. This is one of the effective ways to provide unambiguous conventions for representing the embedded knowledge. Semantics of the metadata that address the users' particular needs in terms of the metadata's literal meanings, are not content-dependent [Winograd and Flores 1987]. The syntax of the metadata is the systematic structural arrangement of data elements for machine processing. The systematic structures of the metadata can be used as a formal constraint on the syntax for the consistent representation of document types. Building constraints on the structure can make the metadata more procurable and interchangeable.

For the purpose of exchanging documents in the Internet environment, the semantic markup can be used to describe the metadata and representation of the document. The objective of the markup notation is the reuse and self-description of the

contained data. Reuse means that we are able to formulate new uses for the information including uses that were not envisioned when the document was created. Self-description means that the documents retain their usefulness over time instead of becoming obsolete if the authoring tools have changed. The self-descriptive makeup data can be relocated, extracted, and manipulated as desired.

CSAs are software entities or software agents throughout this dissertation. They perform in four different ways. 1. They observe the user's behavior. 2. They perform actions based on events triggered. 3. They perform notification to other Collaborative Software Agents. 4. They provide callback based on the users' reactive interactions.

CHAPTER 2

OFFICE AUTOMATION DEFINITION LANGUAGE (OADL)

The TEXPROS's frame templates [Liu and Ng 1996] are portable and structured document formats that allow addressing both content and context of documents as pieces of data on the Web. A method to describe the frame templates can be done by the markup technology. The Office Automation Definition Language (OADL) is an application of the eXtensible Markup Language (XML) [Connolly and Bosak 1998], which allows TEXPROS's frame instances to be described as functional interfaces that can be accessed by remote systems over the data exchange control. OADL uses metadata to depict pieces of documents, which describe how the content and context of the information are presented and interpreted. The markup text is used to describe the attributes and is called metadata. The OADL markup can include active rules that define how a document should be processed and manipulated. These instructions can guide the dynamic interactions between software agents and documents. This procedure provides an efficient and practical means for TEXPROS to be rapidly integrated across the Intranet, the Extranet, and the Internet.

2.1 Evolution of Frame Templates

A Document specified in an *open* structure can easily be exchanged and manipulated. The benefit is that we can use ubiquitous software (such as a Web browser) to process and manipulate any open structured documents.

In the early 1960's, the Graphic Communication Association (GCA) created GenCode to develop generic typesetting codes for dealing with heterogeneous vendors with different data typesets. IBM introduced the Generalized Markup Language (GML) for generalizing the generic document types to solve problems that are machine and application dependent. GML is a high-level language for formatting documents of different types. GML uses the tags to specify the layout instructions. In the early 1980's, GenCode and GML were combined to form a standard for processing textual documents, in an attempt to unify methods for defining, specifying and using the document markup.

In 1986, the Standard Generalized Markup Language (abbreviated as SGML)[ISO 8879:1986] promised to make documents modular and interchangeable. The two key elements of SGML are its syntax and semantic rules. The syntax rules are based on the IBM's GML syntax styles. The semantic rules are derived from the type settings of GCA. In 1989, a language called Hypertext Markup Language (HTML) [Berners-Lee 1989, Raggett, Hors, and Jacobs 1998] was created based on a small set of SGML's markup concept. HTML has a common tag set for presenting data, but not the description of the semantic meanings of the data. It does not have certain features, such as the extensible feature, the semantic structure, and the data validation, that are provided by SGML. But even with limited features, HTML is still widely used on the Internet.

Major drawbacks of SGML specification are difficulty of processing and manipulation. In 1998, the World Wide Web Consortium (W3C) recommended the proposal of the eXtensible Markup Language (XML)[Connolly and Bosak, 1998], which is a more usable format for general purposes in comparison with SGML. XML is a standardized text format designed specifically for transmitting structured data for Web

applications. The data in XML is self-descriptive - that means each data has a self-schema to describe data. A markup language is a tagging system that can be used to break down a document's structure before transmitting it electronically from one location to another location, and then to reassemble it upon arrival at a new location. The tagging system is extensible in the sense that it provides users with a mechanism for creating their own ontology. That is, it is a common system for codifying any concepts that are meaningful to the software agents.

However, HTML is not used to describe data with semantic information and data structures efficiently and consistently over the network. The purpose of using markup technology is to simplify the variety of file formats which solve non-compatible file formats that can not be processed without additional compatible software. The markup language is designed to create structured documents so that they can be transferred easily across the network and processed consistently anywhere. The frame templates of TEXPROS specify uniform ways for annotating documents as information containers. The frame templates are represented as document types. Folders are referred to as documents of the same class. They are nicely integrated to form a dual-model; a Document Type Hierarchy (DTH) depicts the structural organization of the documents, and a Folder Organization (FO) represents the user's real-world view of the document filing system.

2.2 Office Automation Definition Language

The Office Automation Definition Language (OADL) is used to specify the frame instances of TEXPROS as functional interfaces. The frame instances can be accessed by

remote systems over data exchange control, specifying predicates to form the folder organization, and specifying active rules to activate the documents.

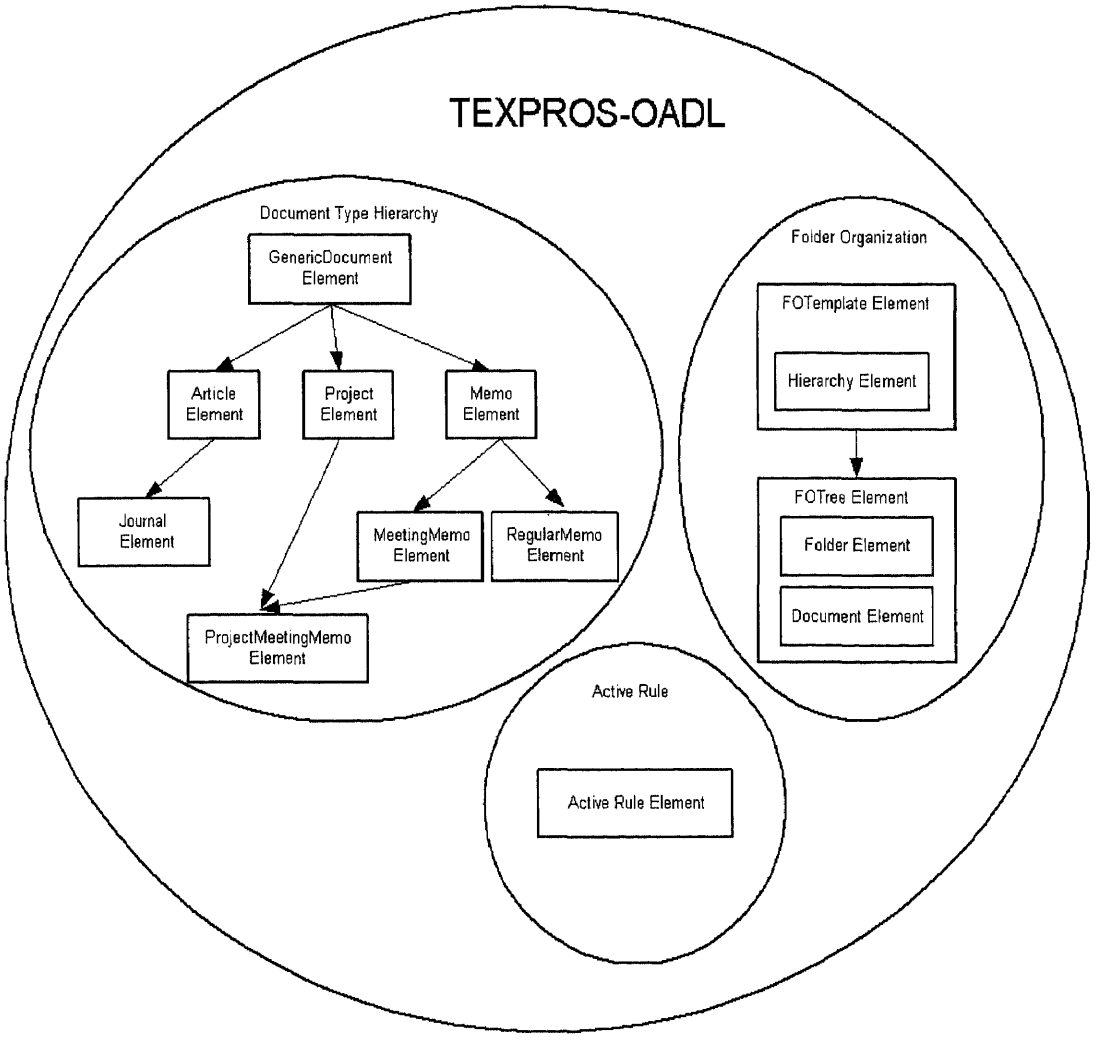

Figure 1 OADL overview

Figure 1 shows that TEXPROS-OADL comprises three components: Document Type Hierarchy, Folder Organization, and Active Rule. The Document Type Hierarchy depicts the structural organization of the documents. The Folder Organization represents the real-world view of document filing system. We propose Active Rule with the help of software

agents. The Active Rule is defined in OADL for event notification and workflow guideline.

2.2.1 Semistructured Data

The data in the computer system can be represented in different forms, either an unstructured form of data in a plain ASCII file system or a highly structured form of data in a relational database system. These data could be raw, such as the byte stream of sound or an image. They could be structured so that they can be stored in a relational database system with rigid types. Some structured data such as word documents or HTML documents are only used for presentation purposes.

OADL is designed for use in a distributed environment with multiple document repositories. The OADL documents differ from traditional relational or object-oriented data. However, the OADL documents are very similar to the semistructured data model. Semistructured data [Abiteboul 1997, Buneman 1997, Fernandez, Popa, and Suciu 1998, Fernandez, Florescu, Levy, and Suciu 1997] referred to data that are neither raw nor strictly typed. That is, they are neither table-oriented, as in a relational model, nor sorted-graph type, as in an object database. These data are generally described as a format that does not conform to a rigid schema. The data are represented by a collection of atomic or complex objects. The value of an atomic object can be some basic types, such as integer, string, floating point, or date. For example, "1999", "11/12/1999" and "John Smith" are atomic objects. We can define "1999" to be integer type, "11/12/1999" to be date type, and the " John Smith" to be an array of characters. The values of complex objects are sets of (attribute, value) pairs. For example, a book can be described as

Book: {

Author: "John",

Author: "Helen",

Title: "Fundamental of Database Systems",

Publisher: "ACME",

Date: {Month:"December",

Year: 1998},

ISBN: "0-8053-1748-1"}

This structure has some characteristics: repeatable attributes, composite attributes and dynamically changing attributes. A repeatable characteristic means an attribute can be defined to appear multiple times. For example, the book object can have multiple "Author" attributes. The composite attributes means that the data allow heterogeneous sets of attributes. For example, the book object's date attribute can contain month and year attributes. The dynamically changing attributes mean that we can add or delete any attributes. For example, the book object can add an additional (attribute, value) pair, such as Author: "Thomas" or add a non-existing (attribute, value) pair Edition: "5th edition", as needed.

Semistructured data are beyond the relational data barrier, since their specifications are quite flexible. For example, the semistructured data specification allows values without attributes in records, multiple occurrences of an attribute, composite attributes, the same attribute name of different types in different objects, etc. Therefore, it is important that OADL can represent TEXPROS's frame instances with

composite attributes and repetitive attributes, which are the semistructured data via the metadata markup. The specification of OADL data shares many features of the semistructured data. Its structure can be irregular; the structure is not always known ahead of time, and it may change frequently without notice. On the other hand, it is easy to convert data from documents of different types into one document type, which is specified via OADL, to make any document able to be manipulated for organizations managing their information sources.

2.2.2 Production Rules

The OADL grammar consists of a collection of production rules specified in terms of Extended Backus-Naur Form (EBNF). Each of the rules defines a symbol using this form. The entire structure of OADL, in fact, is expressed in the following production rule.

OADL-Document -> prolog element*

The production rule means: "Any OADL document must include a prolog, followed by zero or more elements." The purpose for prolog is to identify the document, which is an OADL document. All documents that begin with <OADL/> are considered to be OADL documents. The <OADL/> is an empty element that is only used to identify the document format. Any literal strings on the right side of the equivalence symbol must be quoted wherever they appear.

A document type called "letter" is defined with Sender, Receiver, Date, and Bodytext elements. The letter's EBNF syntax is as follows:

<Letter> -> <Sender>* <Receiver>+ <Date>? <Bodytext>

<Sender> -> <Name> <Address> | <Name>

<Receiver> -> <Name> <Address> | <Name>

<Bodytext> -> <Paragraph>*

We use the notation (*) to indicate zero or more repetitions of the term, the notation (+) to indicate one or more repetitions of the term, and the notation (?) to indicate zero or one occurrence of the term.

2.2.3 Document Type Definition

A document is a collection of information that is processed as a unit and is classified as a document type. A document type is a class of documents that share some common characteristics. For example, journal, article, technical manual, and memo are document types. The Document Type Definition (DTD) consists of the definition of element types, attributes, entities, and notations. The entity is a collection of characters that can be referenced as a unit. It declares which of these are legal within the document and in which places they are legal. OADL's DTDs are defined to comprise the heterogeneous document types, folder organization template, and active rule specification.

An element is a component of the hierarchical structure of a document; it is identified as a document instance using the descriptive markup, usually a starting tag and ending tag. Frame instance is the data and markup for a hierarchy of elements that are considered as an instance of a document type. The frame instance's attributes provide

meta-data for elements, such as a sender, receiver, etc. Table 1 depicts the data definition of a Memo document type. An element declaration is used to define a new element and specify its allowed content. The keyword 'ELEMENT' introduces an element declaration. Each document type is defined as an element in the OADL. For instance, a Memo document type is specified in a Memo element that composes a FrameTemplate element, a logo element, a sender element, a receiver element, a date element, a subject, and a body element. Each element may have multiple or zero occurrence depending on the symbol specified in the definition. The following is the Memo element definition.

Table 1 Memo definition

<!ELEMENT Memo (FrameTemplate,logo?, sender*, receiver*, date, subject, body)>
<!ATTLIST ParentType "GenericDocument">
<!ELEMENT FrameTemplate "Memo">
<!ELEMENT logo EMPTY>
<!ATTLIST logo
SRC CDATA #REQUIRED>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT receiver (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (paragraph+)>
<!ELEMENT paragraph (#PCDATA)>

The "<!ELEMENT Memo (FrameTemplate, logo?, sender*, receiver*, subject, date, body)> " is the root element declaration of the Memo. The root element has an attribute called ParentType that is used to specify the parent type(s) of its derived document type. The hierarchy of the document types is specified in the TEXPROS's Document Type Hierarchy. All of the document types inherit the root document type called "GenericDocument". The "GenericDocument" is defined as the root of the Document Type Hierarchy that does not have any attributes called "virtual document type". The Memo type directly inherits from the "GenericDocument".

This document type contains some child elements - FrameTemplate, logo, sender, receiver, subject, date, and body. These elements must appear in the order given in the definition. The question mark (?) following the logo means that the logo can have zero or one occurrence. The logo is optional and only allows one appearance. The asterisk mark (*) followed by the sender and receiver means that the sender and receiver can have zero or more occurrences. The FrameTemplate element is defined to specify the corresponding TEXPROS's frame template. In this example, it has an invariant and default value called "Memo". This means that this element is a "Memo" frame template. Each of the elements sender, receiver, date, and subject are declared to have values of regular character data (#PCDATA). The logo element is an empty element that does not need to have any value associated with it but contains a SRC attribute. The SRC attribute specifies the image location that refers to an image entity. The #REQUIRED means that this attribute is required to have a value. The body declaration is defined as a composite element. It is composed of paragraph elements. The plus mark (+) means that the paragraph must have one or more occurrences.

2.2.4 Well Formation

A well-formed document has the following characteristics: all tags must be balanced with a beginning tag and an ending tag, which are specified as <string of characters> and </string of characters>, respectively. For example, <sender> must have a matched pair </sender>, where <sender> and </sender> are the beginning tag and ending tag, respectively. An empty tag uses a special syntax, such as, <OADL/>. The value of an attribute is quoted and end-quoted. For example, <link href = "http: // www.njit.edu/index.html"> is a well-formed attribute's value. The link element has a balanced quoted attribute - href. The value of the href attribute is "http:// www.njit.edu/index.html".

Any document that is well formed is easier to use and reuse; these well-formed documents can be transmitted electronically through the network from one location to another and then reassembled efficiently and accurately at the receiving locations. A well-formed document provides the mechanism for specifying each data's boundary. The software agent is able to identify efficiently the beginning tag and ending tag and then filters the data between the tags. In addition, well-formed constraints let the authoring tools construct the documents easily. Each tag's data can be indexed for easy retrieval.

2.2.5 OADL Data Schema

The OADL data schema is described by the markup metadata. Each markup metadata can be thought of as one schema that needs to be replicated with each data item. However, such properties of the OADL data enclose each data item with some semantic meaning. We can retrieve the semantic meaning of the markup metadata to process the data or

create a query language to browse the OADL documents. The OADL documents have flexible usage in the application domain. The same document can be treated in different ways according to any specific purpose. Therefore, any application that retrieves the document syntax and parses the document is free to apply its own set of operational instructions to the data. For example, an invoice document can be interpreted differently by an accounting department and an inventory department. The accounting department uses this document to obtain financial information, but the inventory department uses the document to evaluate the product quantity in stock. Below is an OADL document, which illustrates a simple Memo document.

Table 2 Instance of the Memo element

<OADL/>
<Memo OID="&1" ParentType="GenericDocument">
<FrameTemplate OID="&2">Memo</FrameTemplate>
<logo OID="&3" SRC="njit.gif"/>
<sender OID="&4">Dr. Mike Thomas Graduate Advisor</sender>
<receiver OID="&5">John Smith</receiver>
<subject OID="&6">CIS Qualifying Examination</subject>
<date OID="&7">May, 21 1997</date>
<body OID="&8">
<paragraph OID="&9">I would like to inform you that .....</paragraph>
<paragraph OID="&10">In preparation for .....</paragraph>
</body></Memo>

This document contains Memo, FrameTemplate, logo, sender, receiver, subject, date, and body elements. The body element contains two 'paragraph' elements. The Memo element's attribute "ParentType" has a value of "GenericDocument" which means the Memo element's parent document type is "GenericDocument".

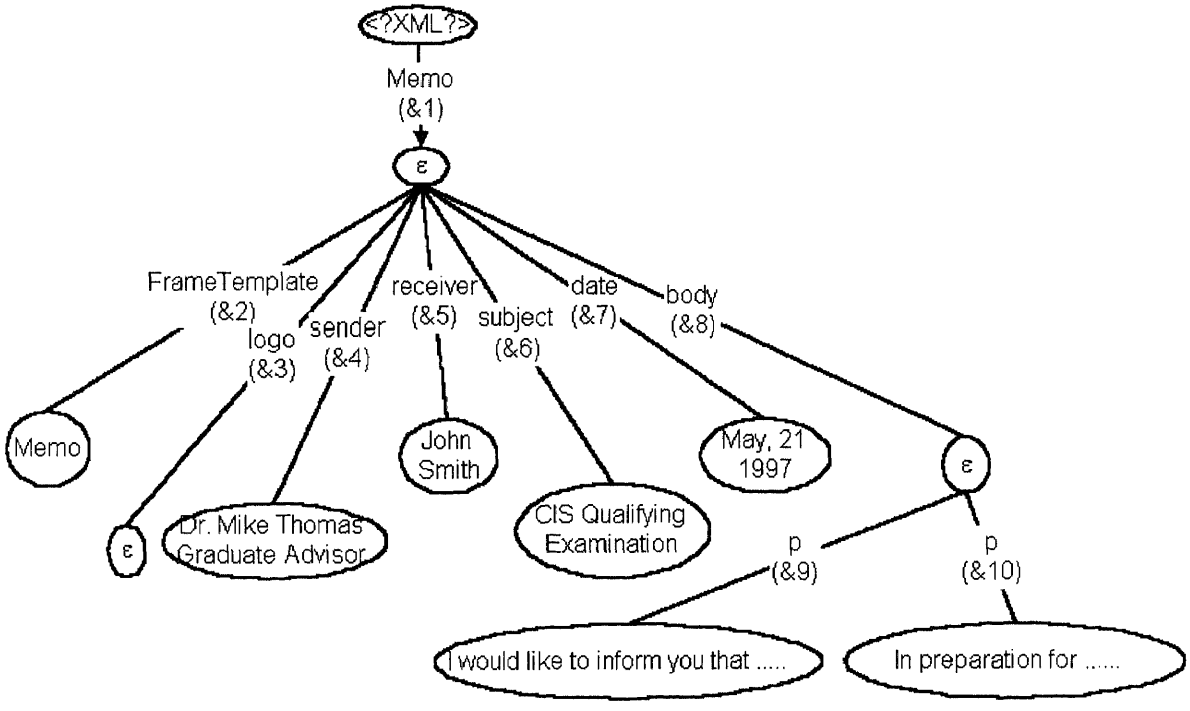

Figure 2 Hierarchical flow object tree

The Memo element shown in Table 2 can be used to extract the pieces of data to be represented in Figure 2 which shows the flow objects. Flow objects are any components of the markup document. A sender is a flow object, and the value of a receiver such as "John Smith" is a flow object. Each flow object is identified with an object id (oid). When there is a need to display this Memo document, we need to style the

document's flow object, and apply the styling rules to the flow objects. Essentially, these flow objects with styling rules can display on the computer screen or browsers.

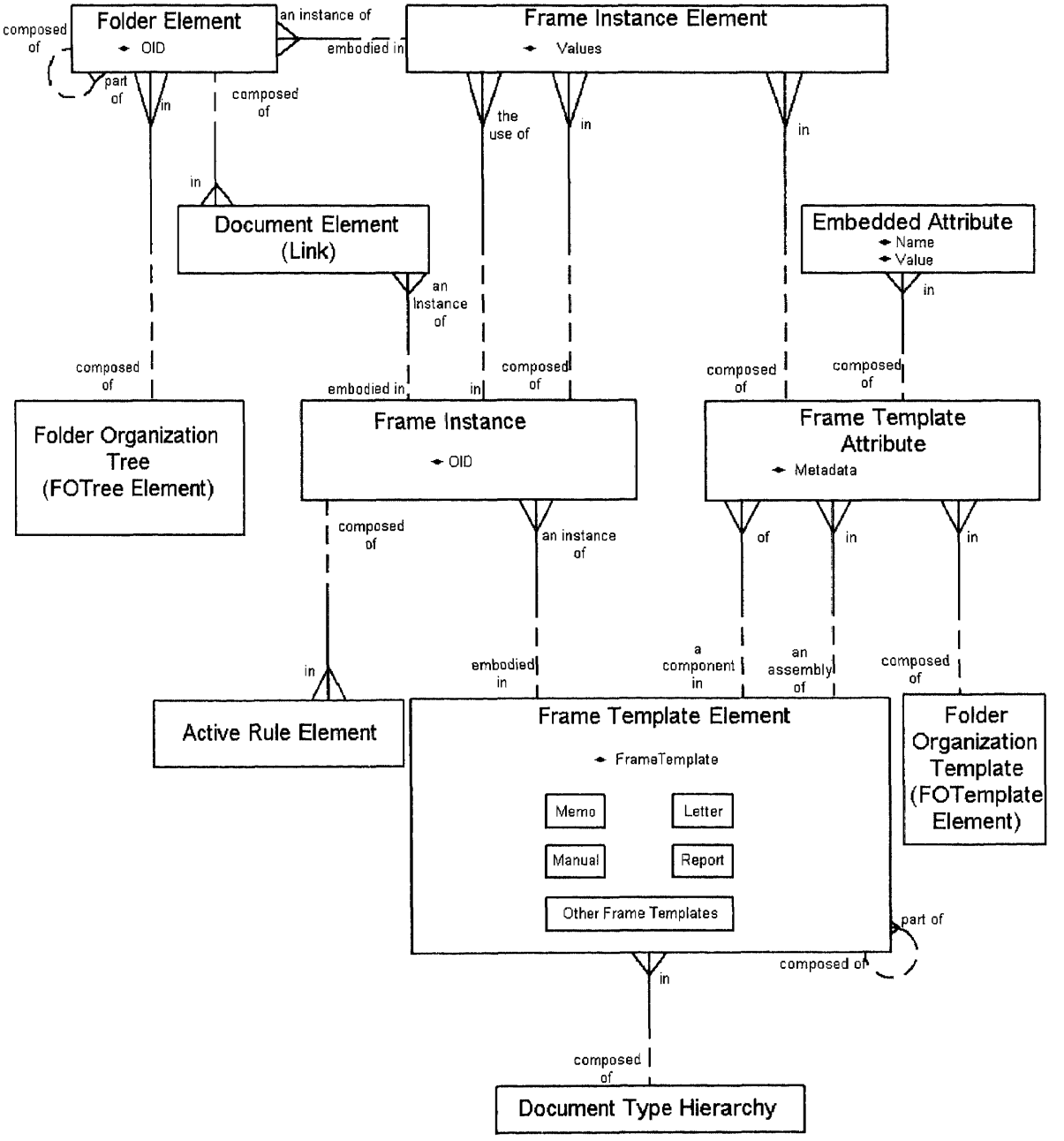

Figure 3 Object model

2.2.6 Document Type Hierarchy and Folder Organization Object Model

We now use CASE*Method [Barker 1990] notation to represent the TEXPROS's Folder Organization, Document Type Hierarchy, and Active Rule. Figure 3 shows the TEXPROS's object model. The object model creates a model at a higher level of abstraction. In this object model, the fundamental components are the Document Type Hierarchy, the Folder Organization, and the Active Rule.

2.3 Document Type Hierarchy

A Document Type Hierarchy is composed of the structural relationship of frame templates. Each frame template is represented by a frame template element and described by a set of attributes for the class of documents. The frame template elements are used to describe the document types that include Memo, Letter, Article, or other document types. Each frame instance must be an instance of a frame template, typically predefined as to its structure and contents. Conversely, a frame template element may be embodied in one or more frame instance elements. For example, a Ph.D. Dissertation Guideline is a frame template. A student's dissertation written according to the Ph.D. Dissertation Guideline is a frame instance of the Ph.D. Dissertation Guideline. Each frame template element is a collection of frame template attributes. Each frame template attribute may or may not comprise a set of attributes that are called embedded attributes. Therefore, the embedded attribute is the frame template attribute's attribute. For example, logo is the frame template attribute of the Memo element. There is a set of logo's embedded attributes called "SRC", "HREF", and "ALT". The embedded attributes are used to give the frame template attribute additional information.

How do we construct the hierarchy with heterogeneous document types? The answer is through document type inheritance. "Inheritance" has the feature of "is-a" relation between supertype and subtype. Basically, "inheritance" defines a relationship among classes, where a class shares the structure or behavior from one or more parent classes. "Inheritance" thus represents a hierarchy of abstractions, in which a subclass inherits from one or more superclasses [Booch 1994].

A child document type inherits all the properties of its parent document type. That is, every attribute or property of the parent document type is automatically an attribute or property of the child document type. Thus, the MeetingMemo document type has attributes sender, receiver, date, etc inherited from the Memo document type.

A document type may have more than one derived document type, with each child document type inheriting attributes from its parent document type. Furthermore, child document types may themselves have child document types, yielding a hierarchy of document types where each document type inherits the attributes from its ancestors. The MeetingMemo is defined below.

Table 3 MeetingMemo element definition

<!ELEMENT MeetingMemo (Meeting-Day, Meeting-Time, Meeting-Place)>
<!ATTLIST MeetingMemo ParentType "Memo">
<!ELEMENT Meeting-Day (#PCDATA)>
<!ELEMENT Meeting-Time (#PCDATA)>
<!ELEMENT Meeting-Place (#PCDATA)>

Table 3 shows the definition of MeetingMemo that defines its ParentType as "Memo". The MeetingMemo inherits the Memo's attributes such as logo, sender, receiver, subject, date, and body; it adds three attributes called Meeting-Day, Meeting-Time, and Meeting-Place.

It is possible for a document type to have more than one parent document type. Now, consider a project meeting memo that combines a Project and a MeetingMemo. We can describe this situation by declaring another child document type called ProjectMeetingMemo, which is a child document type of both Project and MeetingMemo. The declaration is:

<!ELEMENT ProjectMeetingMemo (...) >

<!ATTLIST ProjectMeetingMemo ParentType "Project, MeetingMemo">

In general, we may declare a document type to be a child document type of any number of other document types by following the declaration of the ParentType attribute with a colon and a list of those document types. A parent document type can propagate its attributes and structures to its child document type and the child document type inherits attributes from one or more parent document types. Thus, a ProjectMeetingMemo document is defined to have all the attributes of both document types, Project and MeetingMemo. Figure 4 Illustrates the child-parent relationships involving these eight document types. The ProjectMeetingMemo document type demonstrates the case of multiple inheritance. The ProjectMeetingMemo document type inherits from both Project document type and MeetingMemo document type.

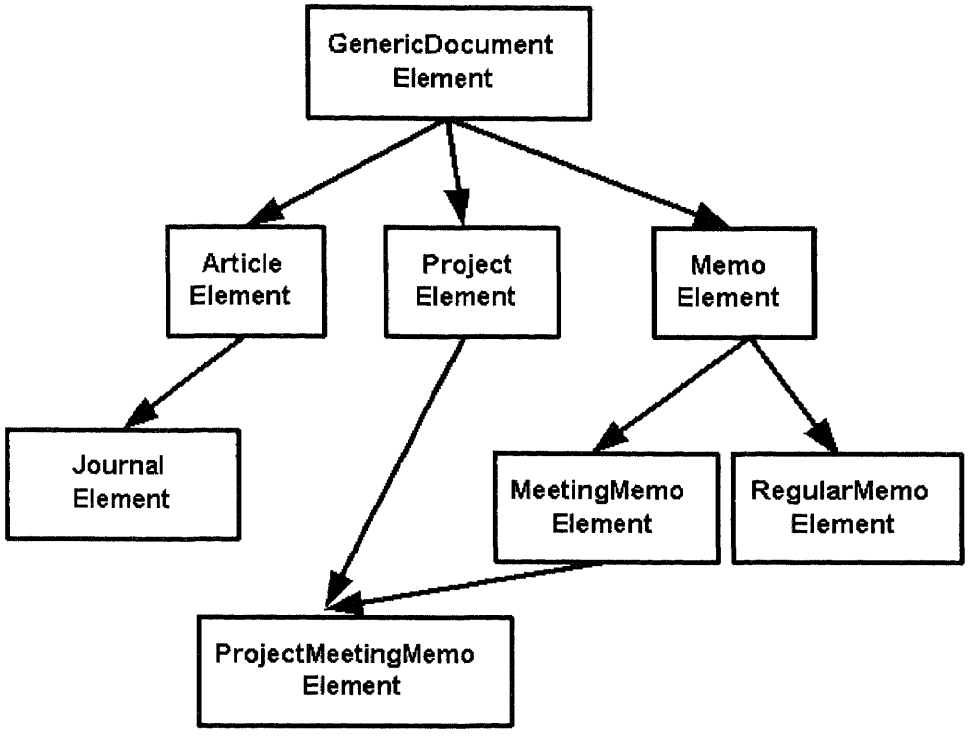

Figure 4 Document type hierarchy

Figure 4 shows that the Document Type Hierarchy is a Directed Acylic Graph (DAG). The Document Type Hierarchy is a hierarchical relationship among the sets of document types. The root document type is the GenericDocument document type that does not specify any attribute. The Memo, Article, and Project document types inherit from the GenericDocument document type. The MeetingMemo and RegularMemo document types inherit attributes from the Memo document type. The Journal document type inherits attributes from the Article document type.

2.3.1 Attribute Name Conflict

There is a potential problem for conflicts among attribute names. Two or more of the parent document types of child document type may have an attribute of the same name,

and the semantic meaning of these attributes may differ. That's why we need to define the software agent's ontology to interpret any OADL documents in a reasonable way, as they should be.

2.4 Folder Organization

There is a need for a document management system that can organize documents in many document bases. Each document is physically stored in a document base or a document repository. If we need to organize documents from different document bases, we should provide a virtual global view of the document bases. This means, the document management system should provide location independence for the documents. A useful technique is to provide a hierarchical naming model. The Folder Organization of TEXPROS provides the solution. A Folder Organization [Fan and Ng 1998] can be viewed as a facility that provides links connecting various groups of frame instances. A global document repository provides a hierarchical view of groups of frame instances. Figure 5 shows that a Folder Organization is comprised of a set of folders, each of which is comprised of a set of links or other folders. In TEXPROS-OADL, we represent a link as a hyperlink that specifies the physical location of that document. The Folder Organization Query Language (FO-QL) is designed to query the document base and generate the hierarchical folder structure. The hierarchy structure is a tree structure and is often called an inverted tree because it is normally shown with the root at the top of the tree. Inverted trees are the data structures used to represent hierarchical file structures.

In comparison with the traditional file system, there is no naming model for the entire file system. Each document in the file system's directory must exist physically.

Even though there are symbolic links or shortcut links in a current file system, it is still difficult to organize and structure a huge amount of document links.

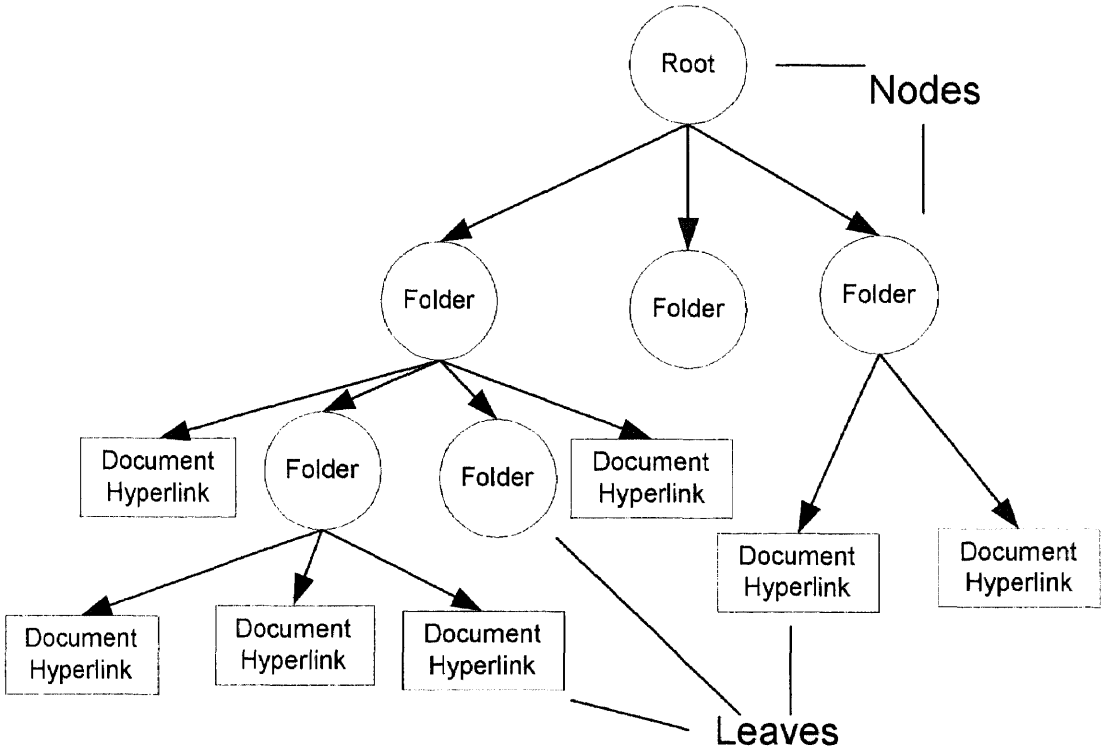

Figure 5 Folder organization

FO-QL contains the specifications of the folder organization template and the folder organization tree. The folder organization template defines the user's point of view concerning the hierarchy of the frame template's attributes. The folder organization tree contains document links and folders. A link to each document can exist in multiple folders.

The hierarchical view is comprised of a set of folders. Each folder specifies a predicate that is used to evaluate qualified documents to be deposited in the folder. If a document is qualified by the predicates of a folder, it will deposit its link in this folder. The process is called "filing process". A link is a mechanism that identifies a resource within a framework. For example, a link points to a document resource in a remote file system. Each document resource in a distributed environment must have an identifier. The official name for the identifier is the Universal Resource Identifier (URI). An identifier usually points to an information resource, thereby identifying it with a universally unique name. Using such a URI identifier, we can treat the global Internet as one global document repository.

2.5 Active Rule

Since there are no dynamic features in a conventional EDI, the capabilities to support dynamic functionality must be extended. The capabilities of active rules embedded in the document allow the software agent to provide dynamic features while supporting the traditional EDI systems. However, as the new technical functionality evolves, new challenges must be met.

2.5.1 Challenges

The challenges come from many directions, including the ways active rules are represented, the documents are delivered and transmitted, the active rules are executed, the active rules are allowed to access the persistent resources, and the ways active rules are modified. The next chapter will demonstrate how challenges are met.

CHAPTER 3

ACTIVE DOCUMENTS

In this chapter, we propose active rules in a markup representation to create a powerful distributed document management system. We integrate CSA into TEXPROS (abbreviation for TEXt PROcessing System) [Liu and Ng 1996] to solve the problems of how to embed active rules in the markup documents. Such markup documents, with embedded active rules, are called Active Documents. CSA formalizes the automatic event notification and handling that carry the characteristics of workflow automation in a collaborative environment.

A markup document could have a very versatile usage in a document management system. It could be used as a data message for delivering information from one location to another, such as electronic mail or memo. It could be used as a functional interface for software entities to use as a language for the input and output persistence. In general, an Active Document [Figure 6] is used as a data message and functional entity in the TEXPROS system.

3.1 The Problem and Approach

There are questions concerning active rules embedded in a markup document. How are the active rules represented in the markup documents? How do CSAs parse and execute the active rules? How do CSAs collaborate with each other to perform workflow automation? What is the status of information recorded inside the markup documents and used by CSA?

In general, many problems can be solved by embedding active rules inside documents as semantic markup that can be extracted by CSA. An Active Document carries the specified active rule and CSA parses and extracts the active rule. CSA may execute the active rules if the specified predicate is evaluated as being true.

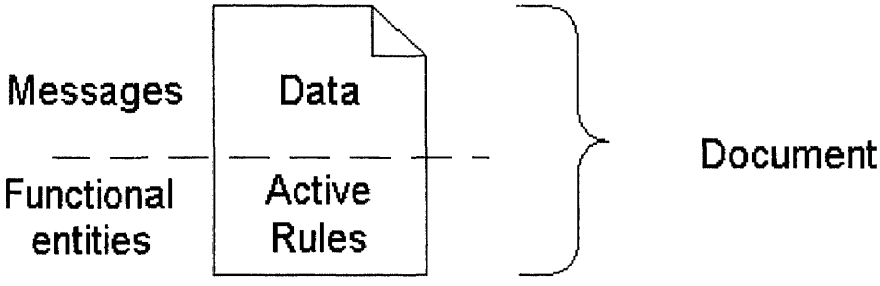

Figure 6 Active Document

Active rules are different from the data that represent the information about the document because active rules are transparent to the users. Transparent means that the user is not aware of the existence of active rules. Therefore, the active rules are used to embed process instructions that are not visible to the users. The active rule can enhance the linkage capability from the traditional simple link. A simple link is a primitive one-directional linking scheme to navigate between documents and can be invoked to retrieve a desired document. In the current Internet world, the Universal Resource Locator (URL) specifies the location of the documents. A simple link is an explicit relationship between two or more data objects or portions of data objects [Maler and DeRose 1998]. The relationships between documents represented in simple links can be obtained only

through an interpretation of the contexts of the documents. The contexts are specified by the URL. That is why the simple hyperlink only provides a static linking facility. This limited simple link capability makes it difficult to implement efficient support for complex link manipulation. Thus, the active rule with CSA provides a methodology designed to support a specific document flow for complex document linking.

3.1.1 Active Rule Representation

An active rule provides the document ontology to CSA and is used to specify any corresponding events, which are parsed and invoked by CSA. An event happens when something needs to be processed. This process must have some data to be evaluated by some conditions and an action has to be taken as a result of the evaluation. This measured condition specifies the predicate of an event. If the result of the condition has been evaluated to be true, an action will be initiated. Each active rule specifies what action should be executed. Table 4 shows the definition of the active rule. The active rule is defined to accommodate three elements: Event, Condition, and Action. The event element specifies only the name of an event embedded in this document. . The Condition element specifies a Boolean predicate for CSA to evaluate. The Condition element is optional. This means some actions may be executed automatically without checking the condition. The Action element specifies the function that needs to be done according to this event. Therefore, the active rule generally defines which function needs to be done, not the function itself. This increases the portability between function and data and function and systems.

Table 4 Active rule definition

<!ELEMENT	Active_Rule	(Event, Condition$^{?}$, Action^{+})>	
<!ELEMENT	Event	#PCDATA>	
<!ELEMENT	Condition	#PCDATA>	
<!ELEMENT	Action	#PCDATA>	

The active rule defines the actions needed (to be done). The convention of a function uses some symbols and identifiers. The dollar sign ($) represents the symbol to identify the function that is defined in the rule base. The conjunction (AND) or disjunction (OR) are used to link two or more evaluations. Active Document only records the function name and parameters in the active rule. The real implementation of the function is up to CSA and the corresponding rule base. The input parameters of the function are from the content of the document.

3.2 Collaborative Software Agent

CSA parses and executes the active rules, communicating with another CSA, and composing and decomposing active rules during document flow. These capabilities provide data access control, workflow status tracking, and document ontology recognition. The data access control focuses on parsing and executing the active rules in the documents, maintaining the database and rule bases, and checking the data validation of the Active Document. CSA is allowed to modify or update the document to meet the process needs. The content of the data records the knowledge in the represented document. Figure 7 shows the interactions between CSA and Active Documents.

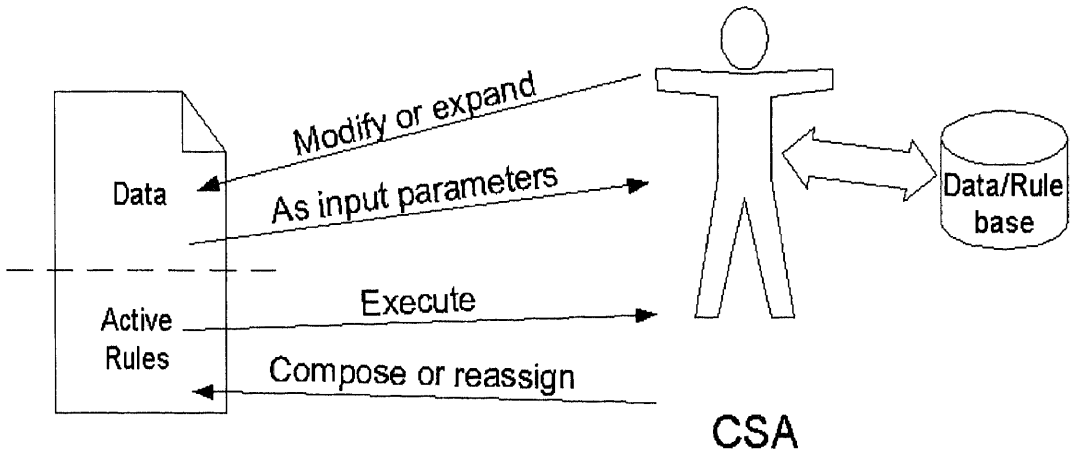

Figure 7 Interactions between CSA and Active Documents

The "workflow status tracking" means that each document carries attributes to represent the status information that needs to be monitored as the document transfers. Therefore, CSA can mediate the workflow process. "Document ontology recognition" means the participating CSA interprets the documents according a same principal and have a corresponding rule base to assist CSA to process the active rules. To "understand" the content of the document means that the attribute-value pairs of the semantic markup document can be recognized by CSA. That's why CSA executes actions according to its rule base. Each particular CSA can incorporate its needs into its rule base. The active rules are described to accommodate three elements: Event, Condition, and Action. The Event element is specified as the kind of event embedded in this document. We can define some events that CSA recognizes. The Condition element is optional; that means that some events will be executed automatically without checking conditions. The Condition element specifies the Boolean predicate. Therefore, the active rules can increase coordination and navigation by modeling and formalizing the document process.

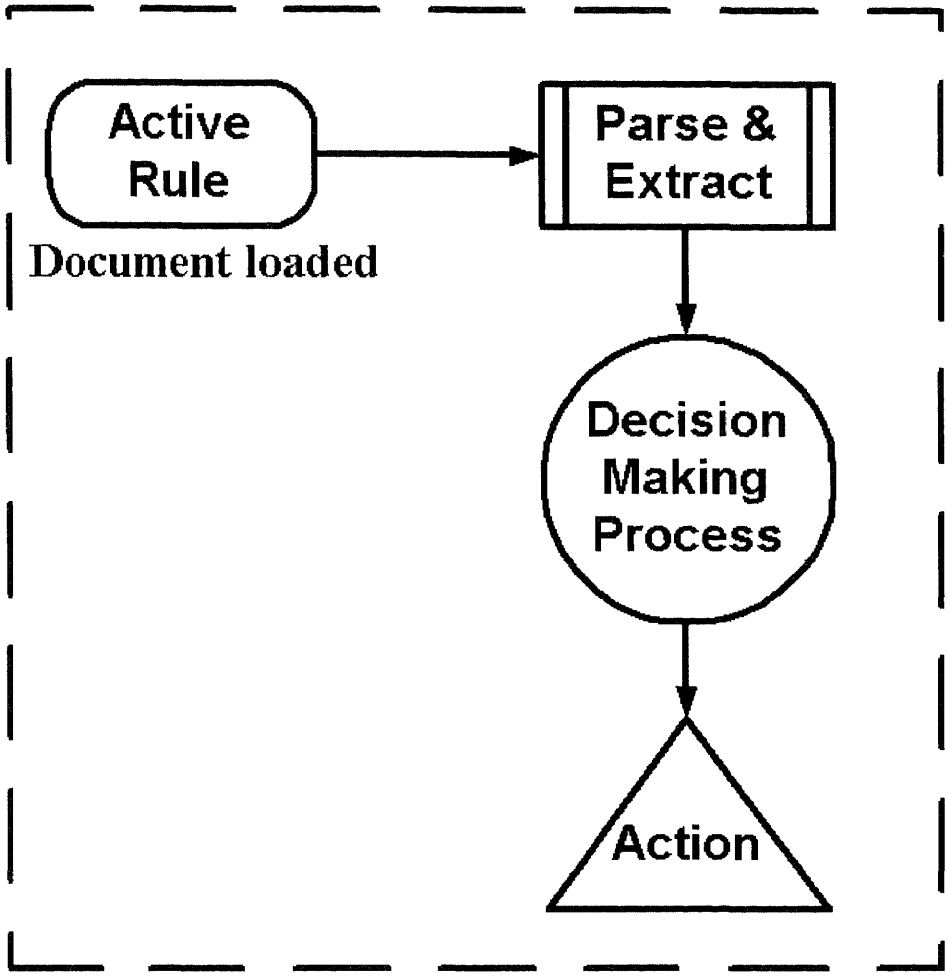

Figure 8 Execution model

Figure 8 shows the processes of CSA. CSA parses and extracts an event, evaluates the condition, goes through the decision-making process, and initiates the action. When one document is loaded, CSA parses the document. If there is any active rule defined, CSA will evaluate the conditions that are stored in the Condition element. CSA needs to determine which event needs to be generated first. Then, CSA starts to evaluate the predicate that is specified in the Condition element. If the evaluation is found to be true, the Action will be initiated.

3.2.1 Execution Model

Figure 9 shows the execution model. CSA plays an important role between Active Documents and database. CSA parses the Active Documents, extracts pieces of the structural markup documents, and matches the specified active rules. If there is any active rule in the document, it will evaluate the condition first. If the evaluation is found to be true, CSA executes the action.

To be more precise, when a document is loaded, CSA parses the document and allocates the required resources, such as memory space. If there is any active rule specified, CSA may need to evaluate the conditions. The conditions are the predicates that check whether an action needs to be executed or not. The condition can be just simple data checking or a complex Boolean expression. It can also call for a function that returns a Boolean value. The input parameters are stored in the document or come from the database. Different CSAs may access different rule bases that may cause different results. Each document allows one active rule to be specified at a time.

CSA applies its knowledge to interpret and extract the pieces of data from the Active Document. In the active rule, the function specifies only the prototype of the function and gives the name and input parameters. Each CSA has its rule base. An active rule only implicitly specifies the method's name and required parameters. The real implementation is specified in the rule base. Each rule may need to have some input parameters that can be extracted from its Active Document. Thus, the matching process finds the matched rule that is eligible to run.

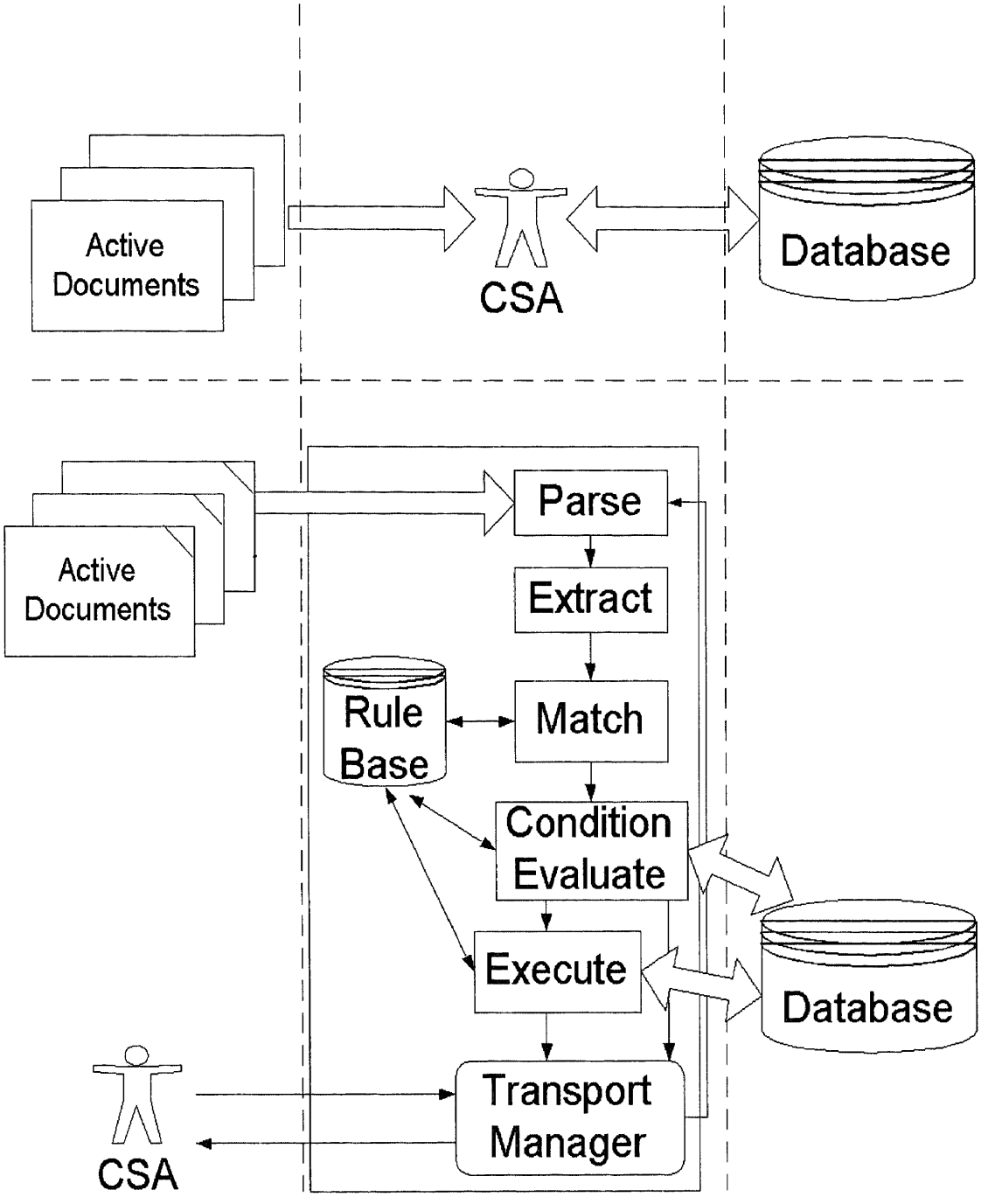

Figure 9 Execution model of CSA

3.2.2 Flexibility of Active Documents

An Active Document comprises the markup data and the markup active rule. Markup features completely insulate the application logic from the heterogeneous systems so

CSA can apply its knowledge to interact and handle the Active Document. CSA composes and de-composes the Active Document, as needed. The markup data can be expandable and modifiable. The active rules only define what to do and leave the process logic to CSA. Thus, the Active Document provides much more flexibility than normal documents.

Figure 7 and Figure 9 together show when CSA executes an active rule, the data portion can be modified or expanded. Furthermore, the Transport Manager may compose or recompose a new active rule if the document needs to be delivered to another CSA. CSA can dynamically change the data and active rules of the Active Document any time it is needed. There are two paradigms that we can apply to the active rule concept. One is the Peer-Peer paradigm that applies to an alternative electronic mail system and the other is the Coordinator-Executor paradigm that is needed when workflow is mediated at a centralized place.

3.3 Peer-Peer Paradigm

Distributing documents from a source (sender) to any destinations (receivers) can form a creative collaborative environment to make document management easier. Inside this collaborative working environment, some critical needs such as user interaction and event notification, can be accomplished by using Active Documents.

The strategy for dealing with this collaborative environment can be built on a medium with an open system architecture, such as electronic mail [Grudin and Palen 1995] and the World Wide Web. Electronic mail provides a means for point-to-point, almost instantaneous communications between a sender and a receiver. Such an approach

provides a Peer-Peer paradigm to transmit the messages. The World Wide Web provides a global collaborative environment for document management. But a regular electronic mail system isn't collaborative because the underlying electronic mail server is only responsible for delivering the message. The electronic mail client program simply sends its messages to the right server and eventually reaches the intended recipient(s) [Farley 1998]. But, there is an additional problem when one sends electronic mail and needs to get feedback or require some active interactions from recipients in a collaborative environment. The normal electronic mail system does not address this problem.

3.3.1 Markup Electronic Mail

Traditional electronic mail is not a markup document. It is composed of a header and a body.

Table 5 Electronic mail header

Received: by song.com (mailer v1.1); Mon, 14 Jun 1999 14:49:48 –0700
Date: Mon, 14 Jun 1999 14:50:01 –0700
Message-Id: 199906142150.OAA17538@song.com>
From: Web Survey <list@song.com>
To: <g_list@lists.song.com>
Subject: Making Styles and Tables Play Nice
Sender: owner@lists.song.com

The header contains Received Mail Server, Date, Message-Id, From, To, Subject, Sender, etc. The body of the electronic mail is the message itself that contains the text.

This text does not have any additional meaning or need for further processes, because the convention of traditional electronic mail is not designed to have dynamic features. In a collaborative environment there may be a requirement for specific needs that will make the electronic mail system more functional. If an electronic mail message accommodates active features, such as automatic feedback notification or interactive data processing, it would be more valuable.

The solution to provide electronic mail with active features is that we can reassemble electronic mail by a markup convention with active rules embedded. We can create a middle-tier process for CSA to recompose markup electronic mail at the sending side and also to decompose electronic mail at the receiving side. CSAs have different responsibilities on the sender and receiver sides. CSA at the sender side composes the electronic message into an Active Document with active rules. CSA at the receiver side decomposes the Active Document into a normal electronic message and does whatever the active rule specifies. A feedback notification active rule provides a capability to dynamically interact between a sender and multiple receivers. This process acts like a callback, where CSA calls back to another CSA to notify of the request.

3.3.2 Notification

In RFC 1891 [Moore 1996], it proposed the Delivery Status Notification (DSN) of electronic mail that added the mechanism of successful delivery notification so a sender can track whether electronic mail was delivered successfully or not. But the successful delivery of a message only means a message arrives at the recipient's Mail Transport Agent (MTA) mailbox and says nothing about whether the recipient has actually

retrieved or read the particular message. The current electronic mail system still lacks the capability to trace the real user's behavior such as when electronic mail is read.

We can define the "Feedback Notification" event as an active rule that is defined in the rule base. Electronic mail can carry a "Feedback Notification" rule that is composed by CSA. The mail data sent through the Mail User Agent (MUA) could be composed as normal electronic mail. Then, CSA re-composes the electronic mail and adds a markup active rule to it. (After the electronic mail arrives at the destination's MTA mailbox.) if the recipient tries to get the electronic mail message, CSA extracts the electronic mail and may invoke a feedback notification event. We identify this feedback notification as a "callback". A callback means that the feedback notification event is awakened at certain points when the active rule is parsed, much like interrupts are used in signal external events. This event will send information to CSA at the sender side.

Figure 10 shows a user sending electronic mail through this CSA-supported electronic mail system. The traditional electronic mail system only has two tiers: MUA and MTA. MUA handles the interaction with users. Users access MUA to read, reply, compose, and dispose electronic mail messages. MTA handles both the mail spool area and delivering process between machines. We provide a three-tier approach by adding CSA to this electronic mail system. CSA provides intermediate functionality in the three-tier approach to compose and decompose the markup active rule and notify CSA at the sender side.

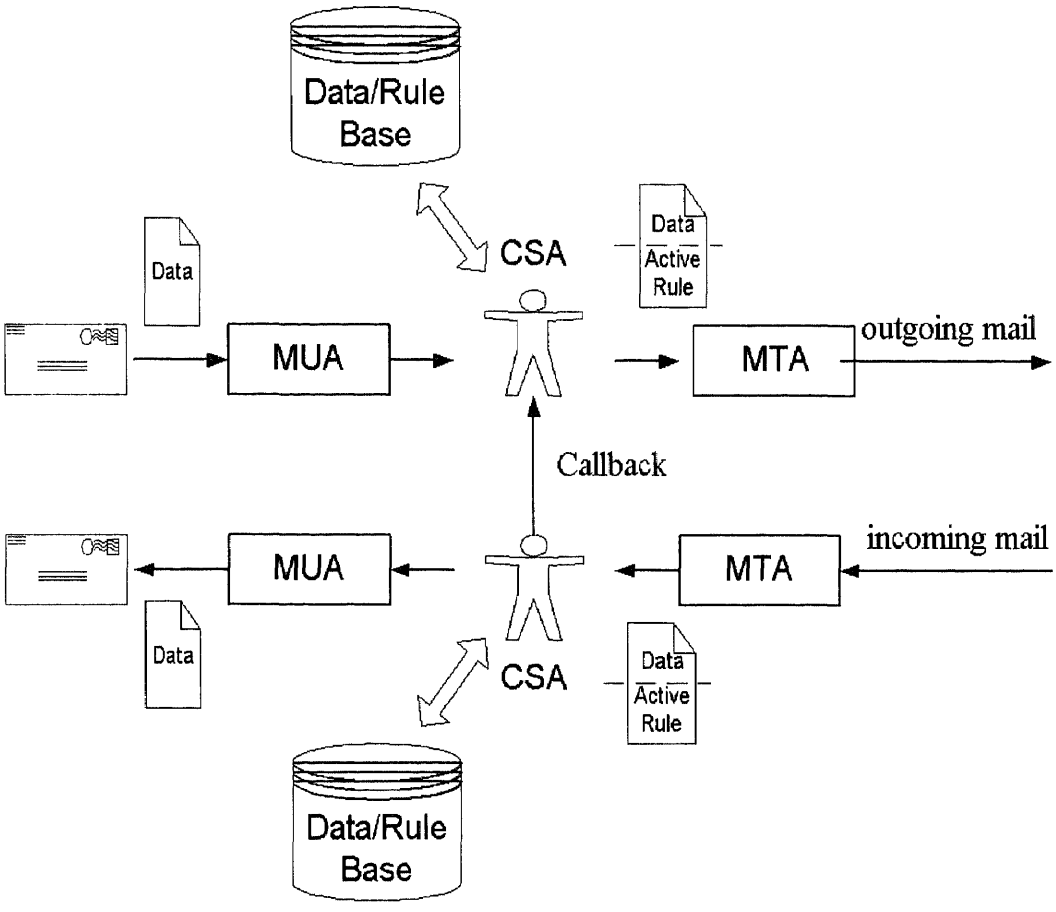

Figure 10 Email system with CSA

Such a "Feedback Notification" event is a tracing process. A trace is a simple way to represent the behavior of an interacting system. It is a sequence of communication actions, providing the input/output history of a particular instance of a system [Broy 1993]. Thus the "Feedback Notification" provides a useful way to communicate in a collaborative environment.

3.3.3 Case Study

A simple example is given as follows: electronic mail is sent to three recipients. This electronic mail records a "Feedback Notification" active rule [Table 6]. This active rule

defines only a callback method. If there is no condition defined CSA always invokes the callback method, no matter what. After sending this electronic mail, CSA at the sender side waits to receive feedback notifications from each of the three recipients.

Table 6 Feedback notification rule

<Active_Rule>
<Event>Feedback Notification</Event>
<Action>
$Callback(this.email)
</Action>
</Active_Rule>

Table 6 shows the sender's original document is a pure electronic mail message. CSA gets the data and references the rule base to compose a "Feedback Notification" active rule. When this electronic mail is sent to three receivers, each document comes with an active rule that specifies the return feedback notification. When CSA parses the document, it will invoke a callback routine to notify another CSA at the sender side. CSA at the receiver side invokes the callback function by HyperText Transfer Protocol (HTTP). Therefore, CSA can use the "Feedback Notification" active rule to monitor any sending processes. If the sender does not receive the receiver's "Feedback Notification", CSA may invoke another action to notify the recipient to read this electronic mail.

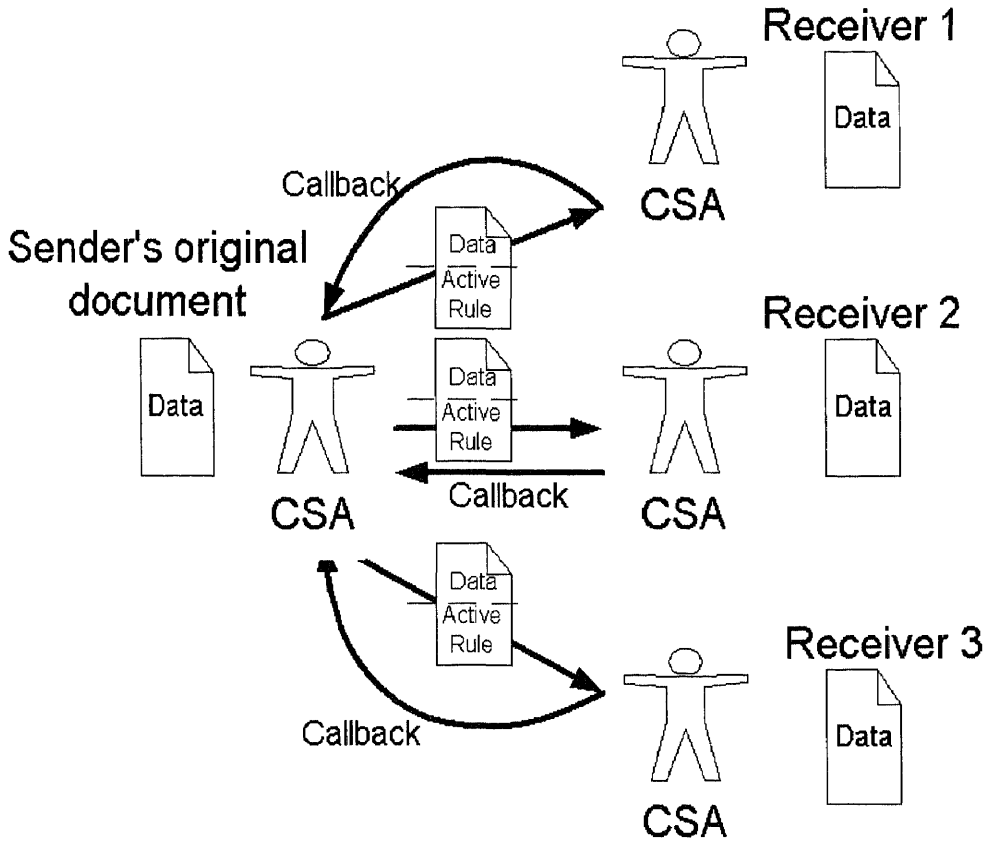

Figure 11 Callbacks

CSA can monitor the sent email documents' "Feedback Notification" events. If the sender does not receive the receiver's "Feedback Notification", CSA may invoke another action to report any exceptional conditions.

3.4 Coordinator-Executor Paradigm

In a Peer-Peer model, each CSA performs independently. But in a Coordinator-Executor model, the coordinator controls the entire asynchronous process and the executor performs the specified individual tasks that are recorded in the active rules. The coordinator composes an active rule and sends it to the executor for processing.

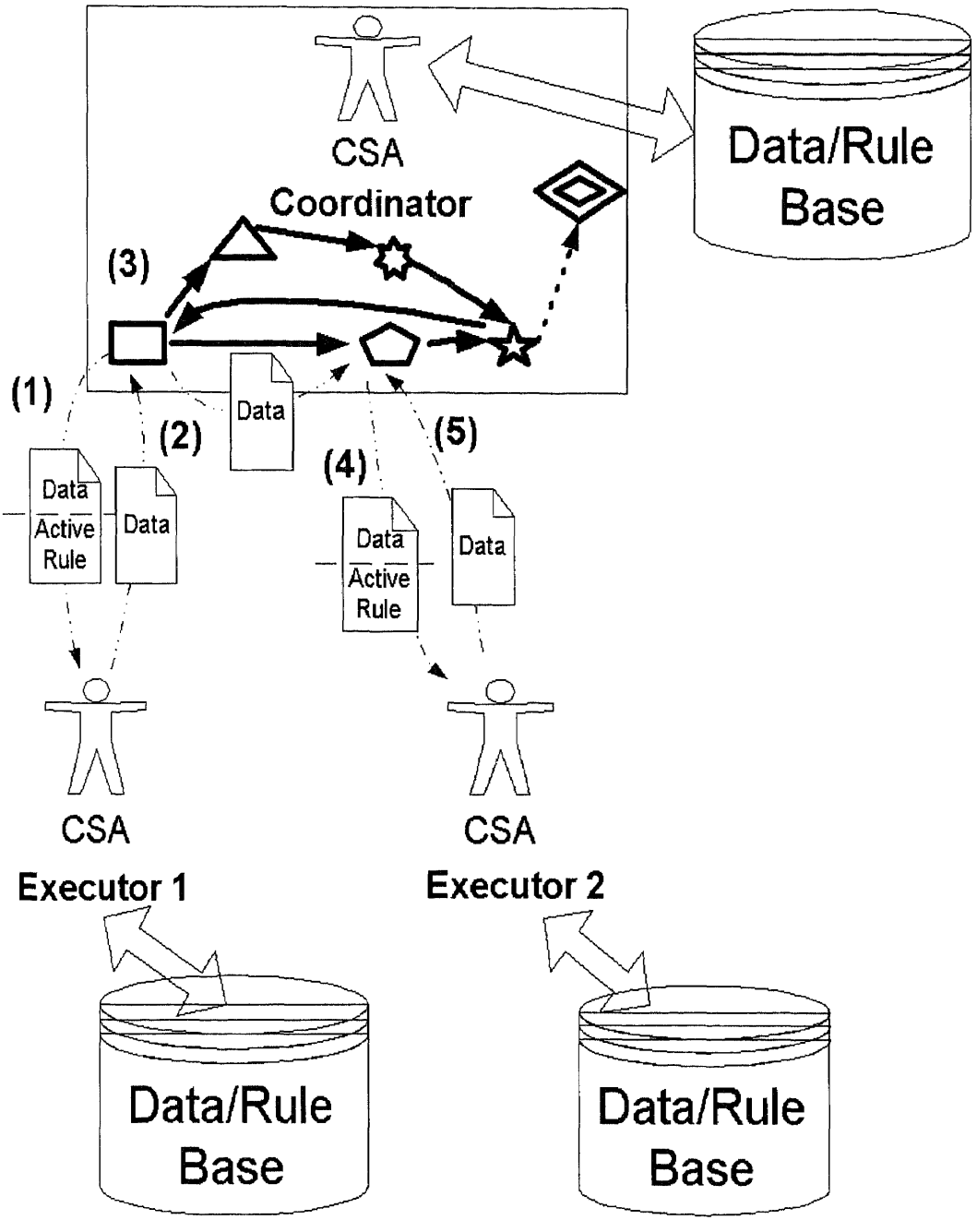

Figure 12 Coordinator-Executor paradigm

A graphical view of the Coordinator-Executor model is represented in Figure 12. There is a CSA that is a mediator or a coordinator. The rest of the participating CSAs are the executors. The collaborative interactions happen between the coordinator and

executors. At the CSA coordinator side, different shapes represent different states of the document flow processes. A CSA coordinator is responsible for monitoring and dispatching jobs to the executor. CSA executors handle the subtasks in a workflow process. When each subtask is completed, a CSA executor returns the resulting data to a CSA coordinator and may recompose a new active rule to another CSA executor during the same transaction period. Table 7 shows a purchase order, in which a purchaser orders a large amount of merchandise. A CSA coordinator needs to check the stock inventory and order from the supplier. We can see that from a single purchase order, it is possible to trigger many reactive behaviors. Each single reactive behavior needs the executor to perform an operation.

The Coordinator-Executor model, in the collaborative activity, can be considered an event-based behavior. An event is a thing that happens, especially when it has some relevance. Relevance can be measured by whether some sort of action has to be taken as a result of the event. Consequently, an event can be seen as a specific situation to which one or more reactions may be necessary [Paton and Diaz 1998]. In a document workflow management system, events can be any operation relating to the document. For instance, a document is being used as a communication media between different software entities, such as a coordinator and executor. This implies that the document workflow process is an active system. It has to provide some mechanisms for users to describe the reactive behaviors, support for monitoring and reacting to relevant circumstances, and for maintaining, browsing, and debugging reactive behavior. If no event is given, then the resulting rule is a condition-action rule like the "Feedback Notification" rule. If no condition is given, then the resulting rule is an event-action rule.

3.4.1 Case Study - Workflow Automation

Workflow automation via the Internet is revolutionizing the way software does computing because each process takes place at a different location. The problem is that different systems or groups normally have difficulty in communicating with each other or the data can't be shared because of the document format. Even in the same location, heterogeneous file formats are still encountered. Each party has its own way of doing business. How can we integrate these heterogeneous systems as that they work together? The answer is to provide a common knowledge interface that uses the same ontology to communicate. The markup active rule with common ontology is provided to facilitate this functional interface during document flow.

Figure 13 illustrates workflow automation of online electronic commerce activity. Each shape represents a process that a CSA coordinator may dispatch using an active rule to an executor under certain circumstances. A CSA coordinator controls the transient states. Whenever a CSA coordinator receives a purchase order, it will start a transaction process and dispatch the event to the customer order executor. This customer order executes a transaction and arranges the needed system resources. For instance, a CSA coordinator needs to store all the customer's information. After the customer order executor returns the result, the coordinator needs to determine the next step. If the order is a service order, it will dispatch the event to the service order executor. If the order is a product order, it will dispatch the event to the product order executor.

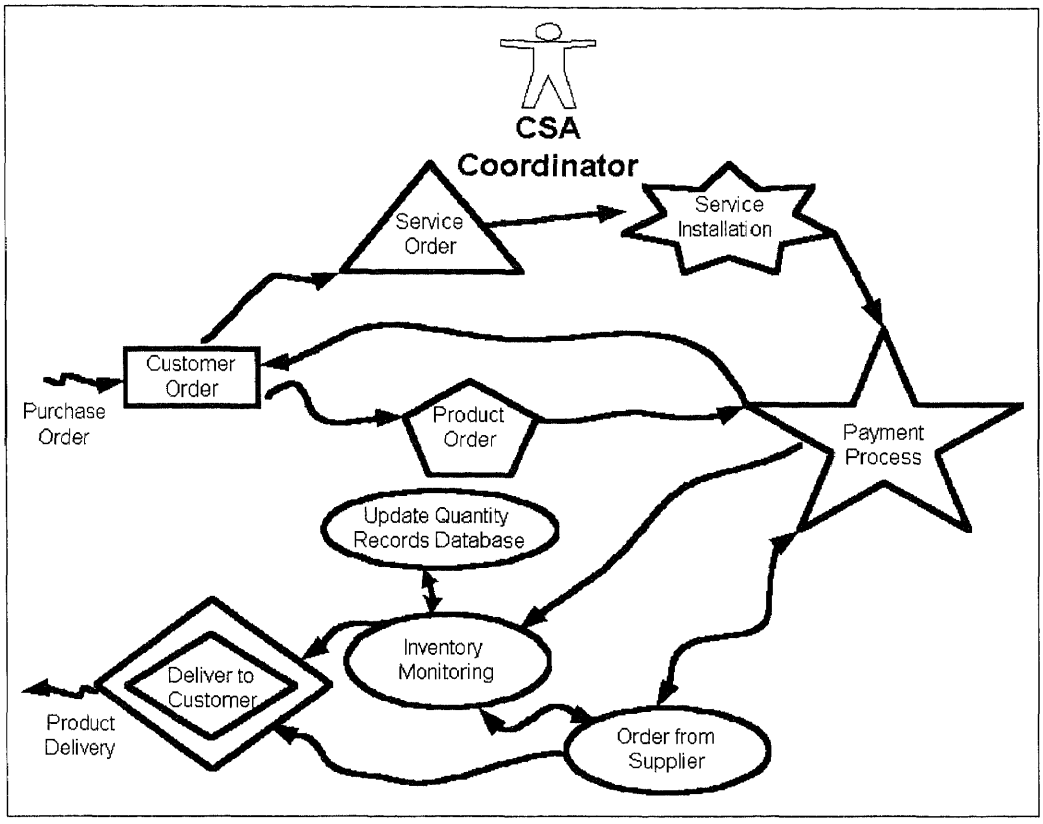

Figure 13 Workflow automation

This example shows a typical Internet sales procedure: while the client orders merchandise in the shopping cart, submitting the order form initiates an order action. This action will activate the purchase order process. Assuming the process allows the payment be paid by the credit card, a payment process is invoked to some authorities or banks to verify a credit check and a withdrawal. If the payment goes through successfully, the Inventory Monitoring executor checks the product inventory. Then the database that stores the quantity records of that product will be updated. If any product is out of stock or below the minimum inventory requirement, there will be another order to the supplier. Finally, the product will be shipped to the client.

Table 7 Product order document

```
<OADL/>
<Order_ID>11101110201</Order_ID>
<Merchant>
  <Name>Huge Hard Disk</Name>
  <Model>80109</Model>
  <Price>$180</Price>
  <Quantity>5</Quantity>
</Merchant>
<Active_Rule>
            <Event>Product Order</Event>
            <Condition>
            Merchant.Name <> NULL
            AND
            Merchant.Quantity>0
            </Condition>
            <Action>
            $Validate_Order(Order_ID)
            </Action>
</Active_Rule>
```

Table 7 shows this document is created right after a customer puts in an order to purchase a $180 hard disk. The functions are identified by the dollar sign ($) followed by the action name. This product order document is an Active Document that mainly comprises an order_ID, a merchant, and an active rule. The conditions specify that the name of the merchant can not be NULL and this merchant's quantity must be greater than zero. If the conditions are verified to be true, this is a valid order. A CSA coordinator will move the process to the next step – the payment process.

Table 8 Payment process document

```
<OADL/>
<Order_ID>11101110201</Order_ID>
<Merchant>
  <Name>Huge Hard Disk</Name>
  <Model>80109</Model>
  <Price>$180</Price>
  <Quantity>5</Quantity>
</Merchant>
<Payment>
  <Name>John Smith</Name>
  <Credit>**********</Credit>
  <Expire> May, 2001</Expire>
</Payment>
<Shipping>
  <Method>USPS 2 Days</Method>
  <Name>John Smith</Name>
  <Address>111 Main St., Little Rock,
          OR, 88888</Address>
</Shipping>
<Active_Rule>
          <Event>Payment Process</Event>
          <Condition>
           $Validate_Credit_Card(Payment.Name, Payment.Credit,
          Payment.Expire)
          </Condition>
          <Action>
           $Charge_Credit_Card(Payment.Name, Payment.Credit,
           Payment.Expire,
           Payment.Price*Payment.Quantity+$SH(Shipping.Address,
           Shipping.Method))
          </Action>
</Active_Rule>
```

Table 8 shows the payment process document that a CSA coordinator sends to the payment process executor. This process needs to verify the credit card information. If the authorized bank approves the purchaser's account, it will then withdraw the amount from the purchaser's credit card. The Validate_Credit_Card function verifies the accuracy of the purchaser's credit card information. The SH function calculates the amount of the shipping and handling charges. The Charge_Credit_Card function charges the fees to this credit card.

Table 9 Inventory monitor document

```
<OADI/>
<Order_ID>11101110201</Order_ID>
<Merchant>
  <Name>Huge Hard Disk</Name>
  <Model>80109</Model>
  <Price>$180</Price>
  <Quantity>5</Quantity>
</Merchant>
<Active_Rule>
        <Event>Inventory Monitoring</Event>
        <Condition>
        $Check_Inventory(Merchant.Name, Merchant.Model,
        Merchant.Quantity)
        </Condition>
        <Action>
        $Update_Inventory(Merchant.Name, Merchant.Model,
        Merchant.Quantity)
        </Action>
</Active_Rule>
```

If the payment process successfully goes through, a CSA coordinator will dispatch this transaction to the inventory monitor process. Table 9 shows the inventory

monitor document that was sent by a CSA coordinator to the inventory monitor executor. If the payment successfully goes through, the Inventory Monitoring executor will check the product inventory, then it will update the database that stores the quantity records of that product. If any product is out of stock or below the minimum inventory requirement, there will be another order to the supplier. Finally, the product will be shipped to the client. Each executor defines its input and output interface and its persistent data such as its own database and rule base. This example shows a typical Internet electronic commerce procedure: when the client orders merchandise in the shopping cart, it will submit the customer order form to generate an order transaction. This transaction will activate this workflow process.

3.4.2 Store and Forward

The Coordinator-Executor model demonstrates how a CSA coordinator composes and dispatches the active rules, a CSA executor evaluates the condition, initiates the action, and then a CSA coordinator recomposes another active rule to another CSA executor. The coordinator controls the flow of the process and all events are defined in the rule base. At each state, the coordinator needs to reference the rule base or calls rule base to find the suitable rules to be embedded in the document. That is why the coordinator or the executors dynamically change the data and active rules. The coordinator stores the returned data from the previous executor and forwards the document with new active rules to the next destination. That's why we call the data transmission "store and forward".

3.4.3 Knowledge Model

The core knowledge representation uses the markup metadata to represent the semantics of data and active rules. Such knowledge representation provides the document management system with consistent ontology to interact with heterogeneous systems. This semantic markup representation is generally referred to as a knowledge model. A common approach for this knowledge model uses active rules that have up to three components: an event, a condition, and an action. The event part of a rule describes a circumstance when the rule may be able to respond to the specified state. The condition part of a rule examines the Boolean predicate in which the event has taken place. The action part of a rule describes the task that needs to be done if the relevant event has taken place and the condition has been evaluated to be true.

3.5 Agent-supported Document Management

The agent-supported OADL documents can be thought of as "Active Documents" that act like functional objects stored in the document repository. This agent-supported framework is divided into three parts: data exchange control, data model, and execution model. This framework treats documents as active objects with functional capabilities. These functional capabilities are beyond SQL3 [ISO/IEC 1996] that is only proposed to give database data with triggers and assertions. These functional capabilities are different from active database systems.

These active database systems include NAOS [Collet, Coupaye, and Svensen 1994, Collet and Coupaye 1997, 1998], REACH [Buchmann, Branding, Kudraβ, and Zimmermann. 1992, Branding, Buchmann, Kudraβ, and Zimmermann. 1993, Buchmann,

Zimmermann, Blakeley, and Wells 1995], and ARIEL [Forgy 1982, Hanson 1996]. The proposed agent-supported documents with embedded rules are represented with markup tags. This active rule markup defines the event, condition, and action. The Event-Condition-Action stands for, when a document is parsed, if conditions are evaluated to be true, the specified action will be fired. CSA acts like a coordinator or an executor for retrieving the document and evaluating the rules.

3.5.1 Data Exchange Control

CSA may have the following properties for data exchange control: secured, trained, traceable, and unreliable. The data exchange control is more like network control in a distributed system.

1. Secured: With secured control, the sensitive documents should only be allowed for qualified access. CSA can enforce the authentication to grant to a client's request for a document or reject the request. The authentication information is stored in the directory storage.

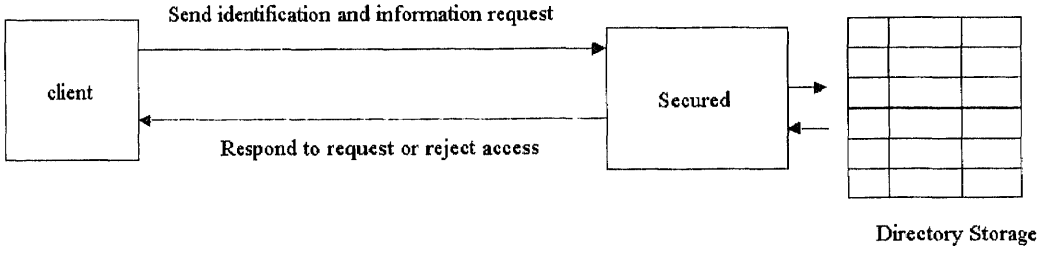

Figure 14 Data exchange control – secured

2. Trained: CSA is trained to respond to any legal client's request.

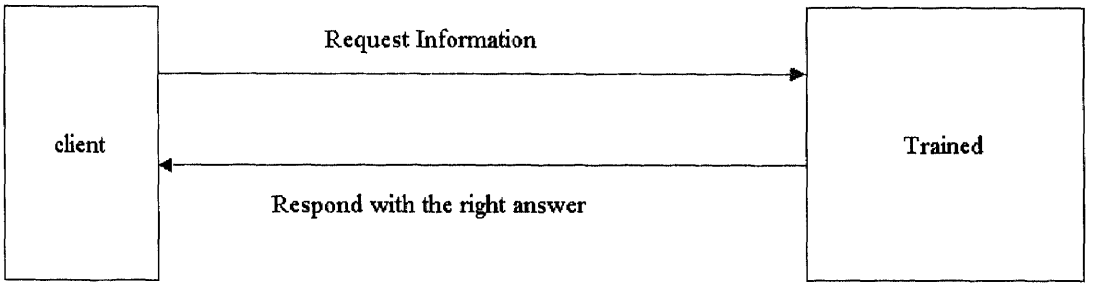

Figure 15 Data exchange control - trained

3. Traceable: CSA traces and monitors the detailed process of a transaction. In a distributed environment, each transaction takes a period of time. CSA monitors every process during this period of time and treats these processes as valid processes.

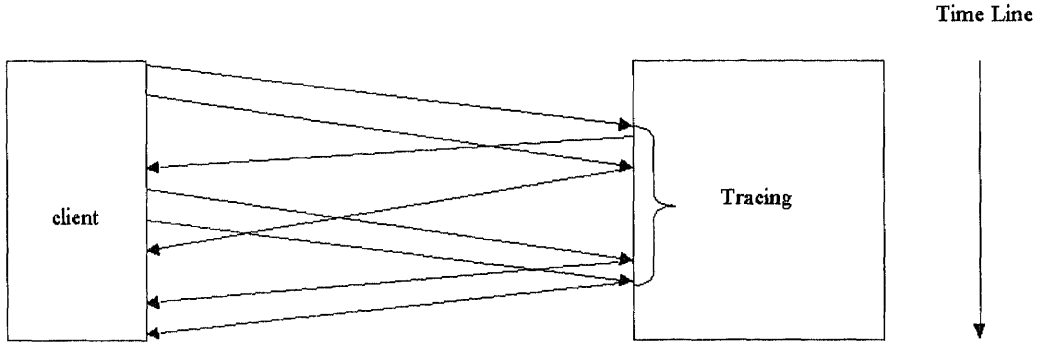

Figure 16 Data exchange control – traceable

4. Unreliable: Software agent does not perform well on the client's request. Normally, software agent responds to the wrong information or cannot communicate with the client.

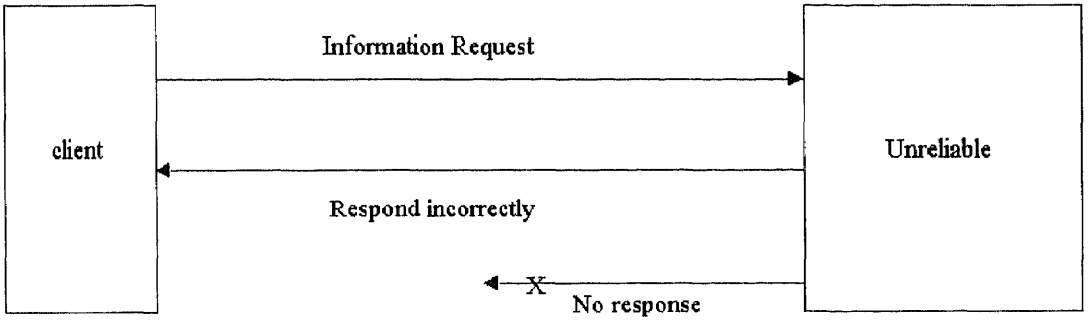

Figure 17 Data exchange control - unreliable

3.5.2 Data Model

The data model is the Office Automation Definition Language (OADL), which is an extension of the Document Object Model (DOM) [Apparao, Byrne, and etc. 1998] with intelligence to collaborate with different systems and provide an interface for CSA to process the knowledge and rules. The data are self-describing tags that are called "markup" and compose a whole document. CSA transforms the document by adding markup tags. This process may let TEXPROS's frame instances be described as functional interfaces that can be accessed by another CSA.

The OADL data schema is declared with the markup metadata. The left-markup and right-markup comprise a data element. Each markup metadata can also be thought of as a schema that needs to be replicated with each data item to represent the semantic

meaning of that data. Such a data model is easy for the software to parse, interpret pieces of the document and apply the semantic meaning.

3.5.3 Execution Model

The execution model is based on the event-condition-action. An event happens, especially when the document needs to be processed. This process must have data to be measured by some conditions and action must be taken as a result of the evaluation. The measured condition specifies the predicate of the event. If the condition has been evaluated to be true, the action will be initiated.

The software agent acts between the OADL documents and the database. The software agent parses the OADL document to verify the document type, which can be verified by the first line of the document <OADL/>. If the document is an OADL document, the software agent may apply its knowledge to interpret and extract rules from this document to get each piece of data in the document. The active rules may embed in the document for further processing. The software agent matches the active rules with its conflict list. This conflict list defines the rules that may be used for the OADL documents. One obvious example is constructing a folder organization that is defined in the conflict list. Each rule may need to have some input parameters that can be gotten by matching data from the extracting process. Thus, the match step finds the rules that are eligible to run in the rule base and also all the required data needed to be the input as the parameters. For example, in the conflict list, we define a credit-card validation. This rule needs some input data to verify the account (e.g. like purchaser's name, credit-card number, and expired date). The data must exist inside the document. For instance, the

purchaser's name can be represented as payment.name that will match the <payment> following a <name>. The value should be enclosed by both <name> and </name>. The active rule may define some conditions that need to be evaluated. If the evaluation is measured to be true, the software agent will initiate the execution. The software agent may modify the document. The execution may need some data from the database or modification of the database. After execution, this modified document may be transmitted to another software agent for the next step in the process. The dollar sign ($) represents the rule that is defined in the rule base. The conjunction (AND) or disjunction (OR) are used to link two or more evaluations. If the credit-card validation and checking merchant inventory both are evaluated to be true, the following action will be fired. The action includes three steps: charge the purchaser's credit card, update the merchant's inventory, and deliver the merchandise. All the required input parameters are defined in the document.

3.6 Conclusion

This chapter illustrates a Peer-Peer paradigm and a Coordinator-Executor paradigm to support active document management. These paradigms are built on the active rules that use the markup ontology. Such ontology solves the problems of the communications between different software entities that we call CSAs. The active rule provides the functional interface for defining the relevant events. For instance, "Feedback Notification" demonstrates the Peer-Peer paradigm and workflow automation demonstrates the Coordinator-Executor paradigm. Therefore, agent-supported document management with active rules can solve the communication problems between

heterogeneous systems. It can also provide collaborative interactions by using the markup technology - Active Documents.

CHAPTER 4

FOLDER ORGANIZATION QUERY LANGUAGE (FO-QL)

The vast number of documents available today is useless unless documents can be effectively and efficiently organized. Document integration is a well-known database problem especially when documents become less restrictive and are permitted to have more variations in the content. Given the flexibility of the TEXPROS [Liu and Ng 1996] folder organization [Fan and Ng 1998], it is desirable for it to be used to facilitate browsing and exploring a large number of documents in the Internet environment. However, the existence of a huge number of documents poses several problems that we must address. In particular,

- How to aggregate a huge amount of documents to construct the folder organization;
- How to construct the folder organization on multiple document repositories;
- How to provide a way to query the data that is untyped, irregularly typed, or even has missing fields;

We will design a query process to solve the three issues here. The document query processes mainly processes the OADL [Lin 1999] documents. OADL documents can specify that some fields are optional and that others may occur multiple times. We need to provide some semantic structures to query the OADL documents. I propose a *folder organization query language,* abbreviated as FO-QL, for specifying the query process to construct the Internet folder organization. FO-QL lets the software agent [Nwana and Azarmi 1997] aggregate documents to construct an Internet folder organization. In the Internet folder organization, we use the Universal Resource Location

(URL) to represent the hyperlink of each document. As part of the query process, the casting process provides the way to query the data that is untyped, irregularly typed, or even has missing fields. We will also introduce the distributed evaluation to construct a folder organization that needs to query multiple document repositories.

4.1 Document Aggregation

In the world of cyber technology, one of the most valuable tools is the organization of documents. The organization of information can be categorized by its elements. Each element has a value associated with it. The document instances with the same element-value pairs can be associated in a group that is called a folder. The contents of a folder can be considered to have a finite set of links, which point to frame instances, or other folders. The frame instances can be thought of as belonging to a folder based on whether they are qualified by a user-defined criterion that are specified as predicates. The conjunction of the predicates of the filing path from the rooted folder to the designated folder determines whether a frame instance belongs to the folder. The folder organization represents the user's logical view of the hierarchical structure of a document repository.

The Internet folder organization heralds the next generation of document management. It consists of two processes: folder tree generation and automatic filing. The folder tree generation includes parsing the Folder Organization Template and then creating the tree structure of the folder organization. Automatic filing is the process of depositing the document instances into dedicated folders. Folder organization is the process that finds, selects, and organizes all the documents in the TEXPROS's document repository. An Internet folder organization can be irregular. This means frame instances

or documents may change frequently and without notice. Using dynamic query processing to retrieve the latest document changes can solve this irregular problem. This is the reason FO-QL is needed to dynamically generate the result. The result is to construct the relations between the folders and also the relations between the folders and the documents.

4.2 Folder Organization

A folder organization is a Directed Acyclic Graph (DAG) which is a set of folders (vertices) that are interconnected by a set of arrows (edges). For graph G we denote the vertex-set by V and the edge-set by E and write G=(V, E), V={v1, v2,..., vn} E={e1, e2,..., em} The DAG's nodes are called folders that can contain zero or more hyperlinks to documents or sub-folders. The DAG's leaves are represented by hyperlinks. Each hyperlink links to a specified document. DAG has a distinguished node called "root". Each folder is explicitly defined by the predicate that contains its domain predicates and its parent folder's predicate. A domain predicate of a folder is the local constraint criteria that are specified on the Folder Organization Template. The conjunction of the domain and parents' predicates of the filing path from the rooted folder to the designated folder determines whether a frame instance belongs to that folder.

4.3 Folder Organization Query Language

The Folder Organization Query Language (FO-QL) is a query language defined to build the Internet folder organization on the OADL document domains. FO-QL specifies the query process used to query the markup metadata and associated data. Each FO-QL

document is represented as an OADL document. That means it is also composed of markup data. Such markup metadata specify the active rules to construct the folder organization. FO-QL is the extension of active rules mentioned in the previous chapter. The software agent is also responsible for parsing and extracting a FO-QL document to generate a customized folder organization. If there is no distributed evaluation, it's much like the Peer-Peer model, because we only need to query one document repository. If there is a distributed evaluation, it's much like the Coordinator-Executor model, because we need to query multiple document repositories and need to have many executors.

The markup metadata used for the Internet folder organization can be categorized by three different classes: the topic metadata, the access metadata, and the intrinsic metadata. The topic metadata consists of the information that demonstrates the author's argument, e.g., subject, document type, or title. The access metadata consists of features that serve to make the document usable, e.g., sender, receiver, or date. The intrinsic metadata consists of the features determined by the user's intentions.

There is a need to query the OADL documents for constructing the Internet folder organization. The OADL documents are composed of loosely structured (or *semistructured*) data, since there are no restrictions on the tags or nesting relationships. The OADL document's data are *self-describing*: structure and data are intertwined in one format. The Internet folder organization query process needs to:

- Define a template with specification of the criteria of the requested folder organization. The definition of the folder organization template is in Table 10.
- Associate each FOTemplate's Hierarchy with a query.

- Filter and sort the results of queries to generate the folder organization tree. The definition of the folder organization tree is in Table 12.
- Associate each FOTree's Folder with a query.
- Group the query by a set of conditions for nested or recursive processing.
- Select from the specified document repository to create the Internet folder organization.

4.3.1 Folder Organization Template

The purpose of the Folder Organization Template (FOTemplate) is to try to find a way to define the query criteria to generate a Folder Organization Tree. Table 10 shows the definition of FOTemplate. FOTemplate defines the layering of the folder organization and must have at least one Hierarchy element. The Condition attribute of the Hierarchy element is defined to specify the predicate's Boolean expression. The Boolean expression is specified as criteria required to query the document repository and find the suitable folders.

Table 10 Folder Organization Template definition

<!ELEMENT	FOTemplate	(Hierarchy $^{+}$)>	
<!ELEMENT	Hierarchy	EMPTY>	
<!ATTLIST	Hierarchy		
	Condition	CDATA	#REQUIRED>

Assume we want to construct a folder organization like that in Figure 18 with today's financial and political news.

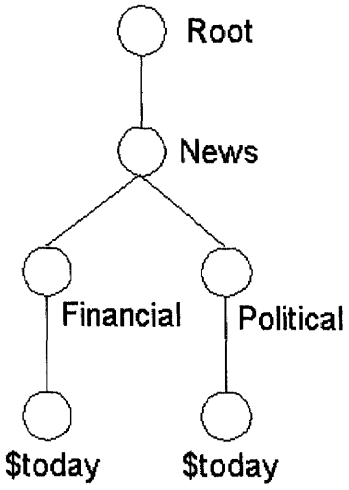

Figure 18 Initial concept model of folder organization

FOTemplate can be specified as Table 11 with a FOTemplate element and three Hierarchy elements. FOTemplate element contains the Hierarchy element without any attributes. We can say that FOTemplate is used like a container that composes only the other elements. The Hierarchy element has an attribute called condition and can't contain another element. In the Condition values, there are some symbols used to represent special meanings. The ampersand (&) represents a delimiter and a conjunction symbol. The values of the condition can use the delimiter to separate multiple markup metadata for more constraints of that layer. The dollar sign ($) represents a function call. The function $today returns the value of today's date.

Each markup metadata can specify the data type for use by the query process. For instance, the FrameTemplate's value is a type of string and can be represented as S. The Date's value is date and can be represented as D. The keyword 'in' specifies the pattern matching criteria. In FO-QL, the data types can be atomic types such as integer (I), string (S), floating point number (F), date (D), and pattern type (P).

Table 11 Folder Organization Template

<FOTemplate>
<Hierarchy Condition="FrameTemplate.S = NEWS"/>
<Hierarchy Condition="Classes.S in (Financial , Political)"/>
<Hierarchy Condition= "Date.D = $today"/>
</FOTemplate>

Table 11 shows there are three hierarchies defined in the template. The first Hierarchy is FrameTemplate equal to NEWS and the data type is string. The second Hierarchy is Classes equal to Financial or Political. The third Hierarchy is Date equal to today.

4.3.2 Folder Organization Tree

The purpose of Folder Organization Tree (FOTree) is to try to define the criteria of each folder's predicate. Table 12 shows the definition of FOTree element that defines the representation of the Folder Organization Tree. FOTree is composed of a set of Folder elements. Each Folder element can be composed of folder(s) or document(s). The condition attribute of the Folder element may come from the FOTemplate's Hierarchy element. The Document element contains two attributes: one is the DESC attribute that is the description of the document and the other is the HREF attribute that specifies the address location of the document.

Table 12 Folder Organization Tree definition

<!ELEMENT	FOTree	(Folder*)>	
<!ELEMENT	Folder	(Folder*, Document*)>	
<!ATTLIST	Folder		
	Condition	CDATA	#REQUIRED>
<!ELEMENT	Document	EMPTY>	
<ATTLIST	Document		
	DESC	CDATA	#REQUIRED
	HREF	CDATA	#REQUIRED>

Table 13 shows the example of FOTree. Each FOTree element contains a set of Folders. Each Folder may contain nested Folders. The Folder element has three attributes: name, type, and value. This example of FOTree is not completed yet; it is the intermediate result of the query process. That is why this FOTree contains only Folders and no Documents.

Table 13 Folder Organization Tree

<FOTree>
<Folder Condition="&FrameTemplate.S =NEWS">
<Folder Condition="& FrameTemplate.Classes.S = Financial">
<Folder Condition="&FrameTemplate.Classes.Date.D = 02061999">
</Folder>
</Folder>
<Folder Condition="& FrameTemplate.Classes.S = Political">

Table 13 (Continued)

<Folder Condition="&FrameTemplate.Classes.Date.D = 02061999">
</Folder>
</Folder>
</Folder>
</FOTree>

The completed FOTree may contain the Document element that includes the description of the document and the address location.

4.4 Folder Organization Generation Process

FO-QL defines the representation of the initial, intermediate, and final result of the folder organization generation process. Figure 19 shows the process generates the Internet folder organization. The process needs two procedures: Folder Tree Generator (FTG) and Automatic Filer (AF). The input template of the entire process is the FOTemplate. FOTemplate initiates the hierarchical skeleton of a Folder Organization Tree (FOTree). The first step begins with FTG that parses FOTemplate, browses the document repository to find the matched folders, and creates FOTree. This tree shows the hierarchical view of the folder organization structure. At this step, FOTree consists of a set of folders because the document's links are not deposited yet. The second step gets the output of FTG as its input, that is, FOTree with no document links. The second step is called the filing process. This process collects the links from a document repository and generates a folder organization.

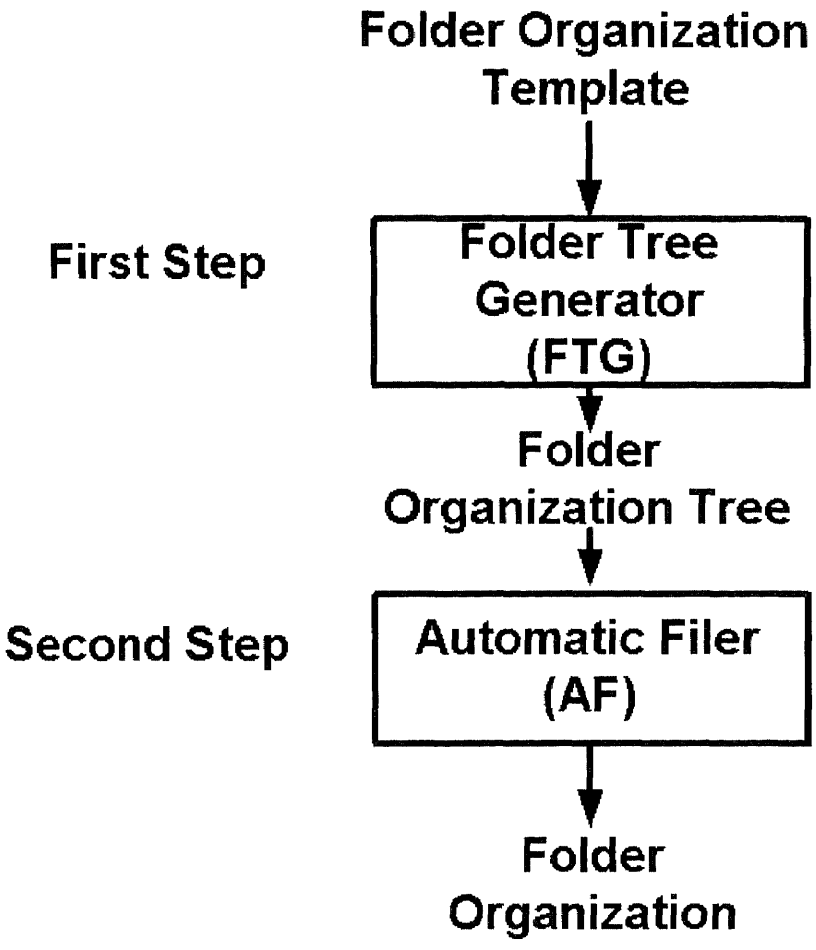

Figure 19 Folder organization generation process

Once the process is completed, the folders may contain links to documents or subfolders. The contents of a folder can be considered as a finite set of links, which point to frame instances, and/or other folders. The frame instances can be thought of as belonging to a folder based on whether they are qualified by its user-defined criteria, specified as predicates. The conjunction of the predicates of the filing path from the

rooted folder to the designated folder determines whether a frame instance belongs to the folder. The folder organization represents the user's logical view of a hierarchical organization of the document repository. In the Internet environment, each document is referenced as the hyperlink of the Universal Resource Location (URL). The document repository is the storage of the documents. The document repository can be a file system or a Web site. The Automatic Filer is the process used to browse the document repositories and collect the links of the qualified documents.

The input of the folder organization generation process is a FOTemplate and the output is the folder organization. The reason for separating into two steps is we may be able to manually input a FOTree to create a folder organization.

4.4.1 Manually Create FOTree

The advantage to manually creating FOTree is that we can narrow the query process to meet more specific needs. We define each folder's predicate and the Automatic Filer finds the matched documents deposited into this folder.

Directly assigning the predicate of the folder can apply the conjunction and disjunction norm form to the Boolean expression. The two steps of the folder organization generation process generate only the conjunction Boolean expression for FOTree. Therefore, the manually generated FOTree allows a more flexible assignment to the Boolean predicate.

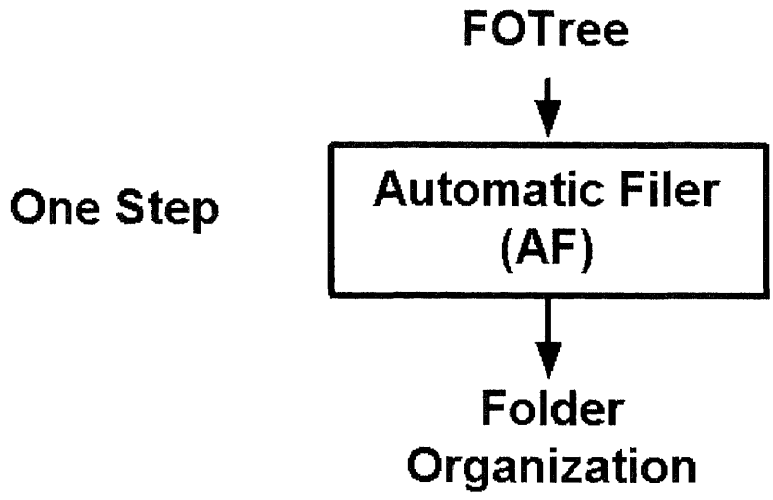

Figure 20 Manually created FOTree

4.5 Query Transformation

The query transformation is a process that parses a FOTemplate or a FOTree to form a query language expression. The query transformation transforms a markup query language representation to a structural query language. FTG and AF query the OADL document repository by using the pattern-matching search to find documents.

4.5.1 Hierarchy Query Transformation

We define FO-QL as a markup representation that must be transformed into another structural query expression representation. For example: a user needs to create an article's folder organization and the article's author can be any name(s). The FOTemplate's Hierarchy is as follows.

<Hierarchy Condition="&FrameTemplate.S='Article'&Author.Name.S like '%'">

This Hierarchy element defines the predicate in its Condition attribute which is defined to find the suitable folders under FOTree. Table 14 shows the structural query expression after the Hierarchy's query transformation.

Table 14 Query transformation

1. SELECT Author.Name AS Folder
2. WHERE Author.Name like '%'
3. FROM (
4. SELECT this
5. WHERE FrameTemplate = 'Article')

In Table 14, line 4 and line 5 are sub-queries that retrieve the documents with FrameTemplate, defined as an "Article". The keyword "this" specifies the document itself whenever it's an article. This sub-query may return a set of documents from line 1 to line 3 for using by the main query. From line 1, the article's author is specified to be the name of the folder. From line 2, following WHERE is a Boolean pattern-matching expression. The percent sign (%) symbol represents any characters matched in the "like" command. We can see that all the articles will match this requirement. This query eventually will generate a set of folders that match the Condition "&FrameTemplate.S='Article'&Author.Name.S like '%'" that is specified in the Hierarchy element.

4.5.2 Folder Query Transformation

After the FTG process, we can get a set of Folders for the next step in the process. FOTree is also a markup representation that needs to be transformed to a structural query language. We also can manually generate a FOTree without Document element inside the Folder element.

That's why we need another Folder element's query transformation. For instance, we have a Folder element like the following:

<Folder Condition="&FrameTemplate.S='Article'&Author.Name.S='John Smith'">

This Folder element can be manually created or gotten from the output of FTG. The predicate defined here is trying to query the document repository and find the matched documents to deposit into this folder. Table 15 shows the result of the Folder element's query transformation. This Folder's predicate is trying to find matches for FrameTemplate when it is equal to 'Article' and the 'Author.Name' is equal to 'John Smith'.

Table 15 Query transformation

1. SELECT this AS Document
2. WHERE Author.Name = 'John Smith'
3. FROM (
4. SELECT this
5. WHERE FrameTemplate = 'Article')

The result of this query tries to get the matched documents needed to deposit their links into this folder. This query retrieves the document instances, FrameTemplate when

it is equal to "ARTICLE" and the Author.Name is equal to "John Smith". This query will return a set of Documents and each document is represented as "this".

Also FO-QL can specify a nested structure. The following shows this nested structure query.

Table 16 Cyclic query structure

1. SELECT this AS Document
2. WHERE Author.Name = 'John Smith'
3. FROM (
4. SELECT this
5. WHERE FrameTemplate = 'Article'
6. FROM (
7. SELECT this
8. WHERE Author.Name = 'John Smith'))

The query transformation can avoid the cyclic query structure like that in Table 16 by making sure the result is a tree structure. Querying such a structure requires traversing through the Hierarchy or Folder elements.

FO-QL provides a regular path expression, which can specify element paths of arbitrary depth. For example, we specify two Folder elements and one Folder nested in another Folder element. For instance,

Table 17 Nested folders

1. <Folder *condition1>
2. <Folder *condition2>
3. </Folder>
4. <Folder>

For the nested Folders like that in Table 17, the inside Folder's predicate must be the conjunction of condition 1 and condition 2.

FO-QL's regular path expressions provide the alternation (|), concatenation (.), and Kleene-star (*) operators, similar to those used in regular expressions. For example, we specify the "author" as "John Smith". This means the value of tag "author" must match "John" follows by "Smith". If the Kleen-star (*) or % appears in the condition value, it means all values are qualified for the metadata tag.

4.5.3 An Example

In an agent-supported document management system, we define the active rules embedded in the document to provide more valuable features. FO-QL is one of the features that can be accomplished by the active rules. Table 18 shows how the active rules represent FO-QL. In OADL, an active rule that defines the folder organization is called "Automatic Folder Organization Construction". FOTemplate element is embedded inside the action part of the active rule. FOTemplate has three Hierarchy elements and their associated Condition attributes. The following is this document:

Table 18 Folder Organization Template

1. <OADL/>
2. <Active_Rule>
3. <Event>Automatic Folder Organization Construction</Event>
4. <Action>
5. <FOTemplate>
6. <Hierarchy Condition = "FrameTemplate.S in ('Memo' , 'Letter')"/>
7. <Hierarchy Condition = "Sender.S = 'John Smith'"/>
8. <Hierarchy Condition = "Sender.Date.D = '*'" />
9. </FOTemplate>
10. </Action>
11. </Active_Rule>

Table 18 shows line 2 is an active rule that contains an event in line 3 - Automatic Folder Organization Construction. This active rule is an event-action style and does not have the condition required. This active rule asks the software agent to automatically generate the folder organization. The condition of this rule is not necessary. In this template, the software agent will generate a folder organization of the John Smith's sending Memo and Letter without considering the date sent. Lines 4 to 10 define the action of the active rule. Line 5 is the "FOTemplate" element which defines the specified hierarchical structure of the folder organization. The "FOTemplate" contains a set of "Hierarchy" elements. The "Hierarchy" element introduces the layering of the folder structure. In this example, the first layer defines the qualified values of the

"FrameTemplate". The "Condition" attribute specifies the qualified requirement for the metadata values. For instance, we want to constrain the qualified types of "FrameTemplate" to be either "Memo" or "Letter". We will use the descriptive style to represent the "Condition" attribute value as "FrameTemplate.S in ('Memo', 'Letter')". The "in" operator means that the FrameTemplate's value can be either Memo or Letter. The Condition can specify the value using a wildcard value like '*' that specifies that all values match. Like the "Sender.Date.D= '*'", all the sending dates are matched to the Condition. The Sender.Date means that the Date markup must immediately follow after the Sender markup.

Table 19 Folder Organization Tree

1. <OADL/>
2. <FOTree >
3. <Folder Condition = "&FrameTemplate.S = 'Memo'">
4. <Folder Condition = "&Sender.S = 'John Smith'">
5. <Folder Condition = "&Sender.Date.D = '11/10/1998'">
6. </Folder>
7. <Folder Condition = "&Sender.Date.D = '11/11/1998'">
8. </Folder>
9. <Folder Condition = " &Sender.Date.D = '1/12/1998'">
10. </Folder>
11. </Folder>
12. </Folder>

Table 19 (Continued)

13. <Folder Condition ="&FrameTemplate.S ='Letter'">
14. <Folder Condition ="&Sender.S = 'John Smith'">
15. <Folder Condition ="&Sender.Date.D = '11/09/1998'">
16. </Folder>
17. <Folder Condition ="&Sender.Date.D = '11/12/1998'">
18. </Folder>
19. </Folder>
20. </Folder>
21. </FOTree>

Table 19 shows the result of the first step in the Table 18 example. Table 18 shows that FOTemplate contains Hierarchies. Table 19 demonstrates that a FOTree contains a set of Folders. Each Folder may contain other Folders. The Folder Tree Generator that browses the document repository to find the list of Folders generates FOTree.

Figure 21 shows the relationship between FOTemplate and FOTree. The right side of the figure is the predicate of folders. Three levels of hierarchies are created in addition to the root. Every layer, except the root, has its own predicate that is determined by the Condition pair. The root of the folder tree is called "FOTree". It comprises all the folders as its children.

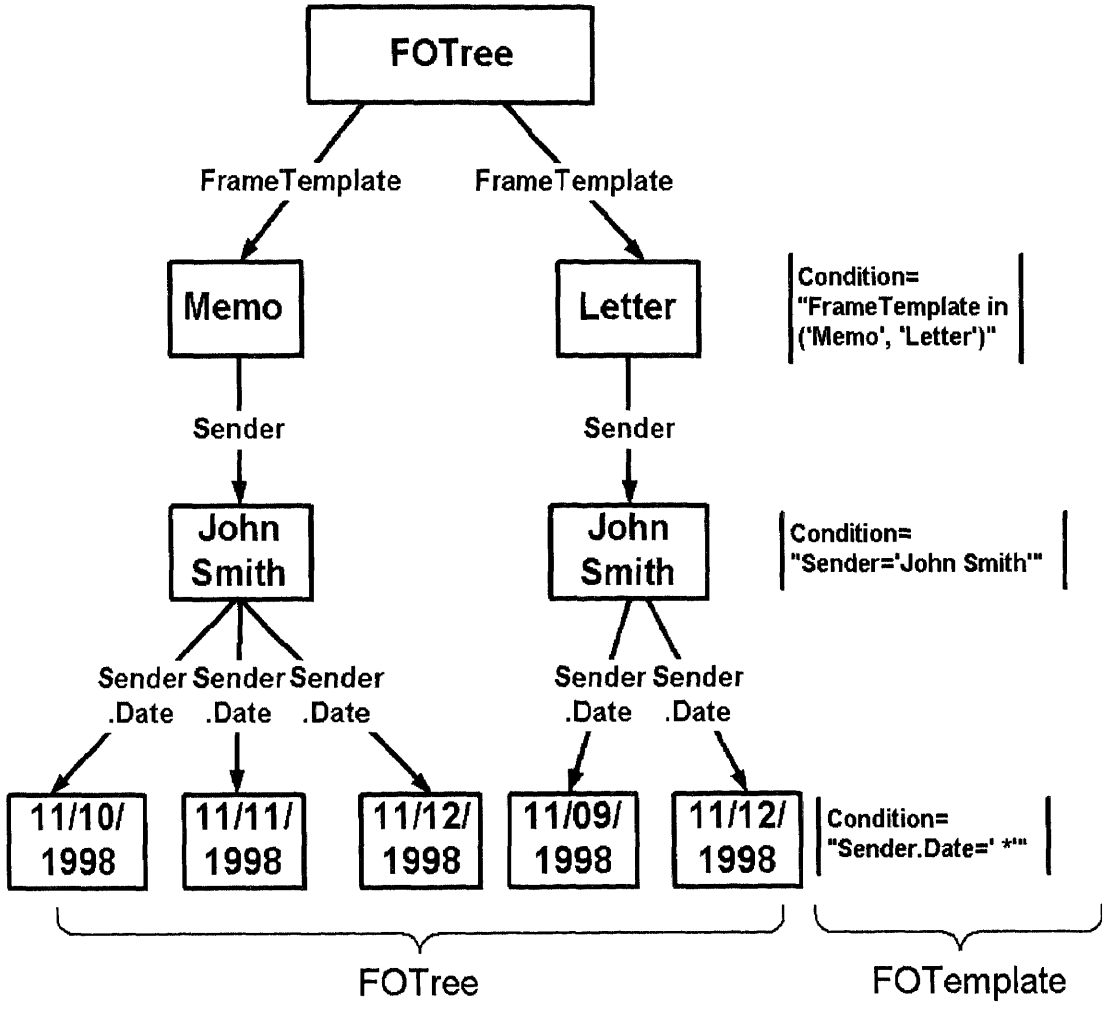

Figure 21 Folder Organization Tree

Figure 21 also shows that there are three days when "John Smith" sent Memo and two days when "John Smith" sent Letter. The folder has a nested structure just like the "Memo" folder which contains the "John Smith" subfolder. The "John Smith" folder contains "11/10/1998", "11/11/1998", and "11/12/1998" subfolders.

At the next step, FOTree is used as the input of the Automatic Filer (AF). AF browses the document repository and collects the links of the document frame instances thereby satisfying the predicates of each folder. We can call this process the automatic

filing process. The generated folder organization is like bibliographical or library classification records. The filing process organizes the document resources by cataloging them. Cataloging is the act of creating a template of descriptive information about document resources such as frame instances.

The filing process mines the document repository to create the specified folder organization. A link in this folder organization is "a pointer from a folder to a frame instance". We represent the link as a hyperlink that is addressed to be the remote location. AF collects the frame instances, which satisfies the predicate of the folder. Each frame instance can be linked by a hyperlink that can retrieve the entire document to which it is linked. The output of the automatic filer generates an OADL folder organization document that can be displayed by the software agent.

Table 20 Folder organization

1. <OADL/>
2. <Folder Condition = "&FrameTemplate.S='Memo'">
3. (A set of documents belong to this folder)
4. <Folder Condition ="& Sender.S ='John Smith'">
5. (A set of documents belong to this folder)
6. <Folder Condition ="&Sender.Date.D ='11/10/1998'">
7. (A set of documents belong to this folder)
8. </Folder>
9. <Folder Condition ="&Sender.Date.D = '11/11/1998'">
10. (A set of documents belong to this folder)

Table 20 (Continued)

11. </Folder>
12. <Folder Condition ="&Sender.Date.D ='11/12/1998'">
13. (A set of documents belong to this folder)
14. </Folder>
15. </Folder>
16. </Folder>
17. <Folder Condition ="&FrameTemplate.S = 'Letter'">
18. (A set of documents belong to this folder)
19. <Folder Condition ="&Sender.S ='John Smith'">
20. (A set of documents belong to this folder)
21. <Folder Condition ="&Sender.Date.D ='11/09/1998'">
22. (A set of documents belong to this folder)
23. </Folder>
24. <Folder Condition ="&Sender.Date.D ='11/12/1998'">
25. (A set of documents belong to this folder)
26. </Folder>
27. </Folder>
28. </Folder>

Table 20 shows the result of Automatic Filer. Each Folder may contain a set of Documents or some other Folders. Table 21 shows the 'John Smith' Folder.

Table 21 Folder

1. <Folder Condition ="&Sender.S ='John Smith'">
2. <Document DESC="Appointment with Tom" HREF="http://Data1/321">
3. <Document DESC="EDI and the Web" HREF="http://Data1/322">
4. <Document DESC="Annual Meeting" HREF="http://Data1/323">
5. <Document DESC="Product Order Form" HREF="http://Data1/324">
6. <Document DESC="Family Party" HREF="http://Data1/325">
7. </Folder>

In this 'John Smith' folder, there are five documents. Assuming that the software agent browses the Data1 document repository, all of the document hyperlinks will have the prefix http://Data1 and each document has an identifier such as 321. Each Document's DESC attribute is only used for displaying the description of that document.

4.6 Type Casting

In the OADL document, there is only a "CDATA" type for the data. This means, only the character data is defined. A query should be able to access the OADL documents with all the data by using "CDATA" type. One of the main issues of FO-QL is how to force the comparison between objects and values. In Structural Query Language (SQL) and Object Query Language (OQL), comparing different types of objects will return a type error. We may avoid this error and extend the query capability by defining this new query language - FO-QL. In the OADL data model, all entities that are an array of characters are objects. Metadata are attributes of the document and are taken from the atomic type *string*. The

values of the metadata are atomic and contain a value from one of the disjoint basic types, e.g., *integer*, *real*, *string*, *date*, *currency*, etc. even though the representation of this value is only a set of characters or an array of characters. We can use these characters to compose the OADL documents.

4.6.1 Comparing Values and Atomic Objects

In Hierarchy and Folder elements' Condition attributes, some predicates can accept more than one type of objects. We need to deal with the basic comparison operators (e.g., like, in, =, <, >, <>).

$$\text{Attribute} \stackrel{\text{def}}{=} \text{String}$$

$$\text{Value} \stackrel{\text{def}}{=} \text{Int} \cup \text{Float} \cup \text{String} \cup \text{Date} \cup \text{Pattern}$$

Figure 22 Type definition of attribute and value

When comparing atomic objects, we need to force the two operands to be comparable whenever possible. Let X be the metadata and let Y be the value of the condition for the metadata. From the OADL data object model, X.T is the attribute X with type T and Y is the value. X is always a string value of the metadata. Assume that X' is a match of X and Y' is the value of X'. To compare Y and Y', let us assume that Y is an integer object. However, we must first force the object Y (string) to its integer value and then check the Y' atomic type. If Y' is an integer or real, then compare Y and Y'. The type function checks the string value that can be compared with any atomic types. The algorithm proceeds as follows:

Table 22 Data comparison algorithm

1. let X.T be the Hierarchy or Folder's Condition X with Type T
2. let Y be the value of the Condition X
3. let X' be a document markup and Y' be the associated string value
4. if X = X' then
5. case T of
6. S: compare Y and Y'
7. I: compare Int (Y) and Int (Y')
8. D: compare Date(Y) and Date(Y')
9. F: compare Float(Y) and Float(Y')
10. P: compare Pattern (Y) and Pattern (Y')
11. default return false
12. else return false

In general, casting rules should provide for the basic atomic types and the corresponding predicates. They also could provide for application-specific atomic types e.g., date, pattern, etc. For instance, we use S as a string type, I as an integer type, D as a date type, F as a floating-point number type, and P as a pattern type. To compare two values, we need to cast type first except for the string type. For instance, a date atomic type is given a general representation for the date like "mm/dd/yyyy". We need to force the date type string value to this format first e.g., " June 11 1998" to "06/11/1998". Assume X to be the metadata for the date and let Y be the condition value for X. First we need to force Y to have the format of "mm/dd/yyyy" first. Let X' match X which is the

date atomic type. Let Y' be the corresponding value of X'. We need to check the atomic type of Y' to make sure that Y' is compatible with the date type. If Y' is the date type, we can compare Y and Y'. The Date function will force the string value to take on the date format.

4.7 FO-QL Advantages

The Folder Organization Query Language is an active rule used to create the Folder Organization in the TEXPROS system. The main contribution is that FO-QL is set up to browse markup documents, which have no type. They are of irregular type, or they have no rigid data schema. Thus, the TEXPROS system can organize markup documents such as HyperText Markup Language (HTML), eXtensible Markup Language (XML), and Office Automation Definition Language (OADL).

Manually browsing the document repository to find the right document is time consuming and unrealistic, especially in a multiple document repositories' environment. FO-QL is used to solve the time-consuming process of browsing huge document repositories.

4.7.1 Two Ways to Construct Folder Organization

For flexibility and usability, we describe two ways of constructing the folder organization: define the conceptual folder organization hierarchy and directly assign each folder's predicate. The process of defining the conceptual folder organization hierarchy is illustrated in chapter 4.3.1. The folder organization generation process automatically constructs the folder organization based on the higher level hierarchy to perform a document-aggregation process and build a folder organization. Directly assigning each

folder's predicate manually can define any user's real-world view of personal folder organization. A predicate is the Boolean expression for setting evaluation criteria on that folder. The following list gives two examples to define the folder organization.

Defining the conceptual folder organization hierarchy.

<Hierarchy Condition="&Author.Name.S=John Smith">

<Hierarchy Condition="&FrameTemplate.S=Memo">

Directly assigning of each folder's predicate.

<Folder Condition="&Author.Name.S=John Smith | FrameTemplate.S=Memo">

4.7.2 Ontology Representation

We need to follow the same process to let the software agent interpret any predicate that we defined in FO-QL. There are two forms of representation: one is the Boolean expression and the other is the data type representation. The Boolean expression uses a disjunction norm form representation (|) and a conjunction norm form representation (&). "|" represents an optional (or) condition symbol and delimiter. "&" represents a mandatory (and) condition symbol and delimiter.

The data type representation is attached to the bottom of each attribute name. Each attribute must have a type specification following the attribute name. Here is a list of current defined data types.

Table 23 Types representation

.S	Stands for a string type
.I	Stands for an integer type
.D	Stands for a date type.
.F	Stands for a floating number type
.P	Stands for a pattern type (like image, sound, etc.)

4.7.3 Distributed Evaluation

The other advantage of FO-QL is that such a process can propagate among multiple document repositories. The task of querying multiple document repositories is accomplished through distributed evaluation. The distributed evaluation is done by a software agent called the coordinator. It is responsible for collecting all the local software agent's query results and joining them together.

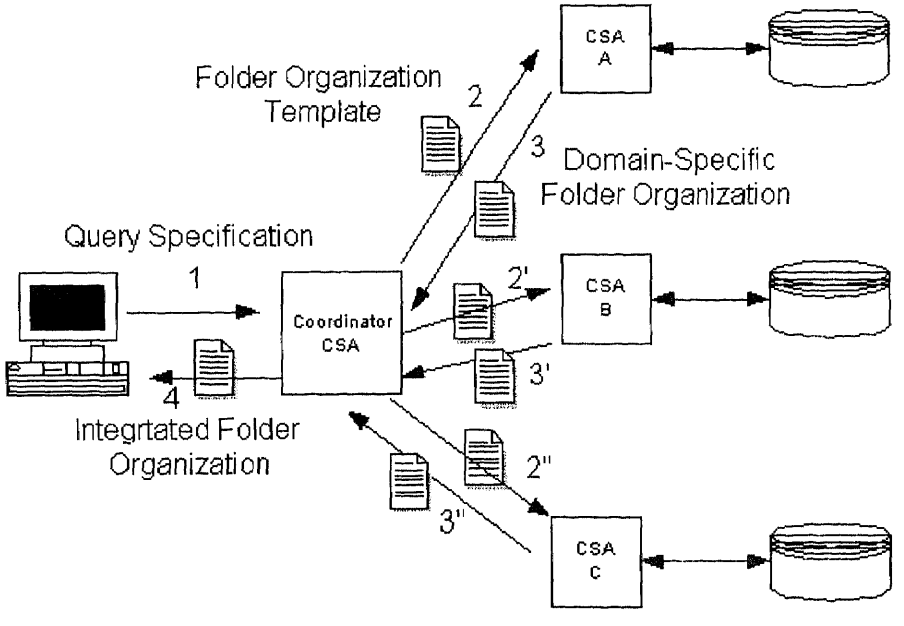

Figure 23 Distributed evaluation

1. A user specifies a query.
2. 2'. 2''. The coordinator delivers FOTemplate to three available document repositories with software agents (A,B,C)
3. 3'. 3''. Software agents (A, B, C) execute the folder organization process and return the results with the document links that are the location of the documents.
4. The coordinator joins together the results from A, B, and C and generates a customized folder organization.

4.8 Internet Folder Organization

The folder organization is designed to fit into the Internet environment. That is why we use the markup technology to represent it. Several features are given below for using this folder organization approach. The user interface uses the ubiquitous Web browser as the platform. Each user only needs to define the high level hierarchy of folder organization that is called a FOTemplate.

4.8.1 Folder Organization in Cyberspace

A folder organization is a guideline for users to use information more efficiently. Hence, it would be useful to develop a framework that would facilitate the classification of data. At first glance, it might appear that such a classification would be relatively easy to develop.

From a user's perspective, the World Wide Web (WWW) is a global information repository, which connects different geographic areas and heterogeneous systems. So making the TEXPROS document repository available everywhere holds a big advantage

for Web browsing users who wish to view documents. Web browsers are ubiquitous; both local and global users can have controlled access to the TEXPROS document repository via the Internet, Intranets and Extranets. Web browsers and Web-based server applications are platform-independent; hence, it does not matter which client operating system is used. The Web's thin client computing paradigm means that most of the tasks are done by the server applications. The user does not have the administration overhead and it is easier to distribute modifications. The software agent can fit easily into the server application to provide more robotic usage.

Internet folder organization correctly manages the relationships between information and human experience. More importantly, if leveraged correctly, each folder can collect needed information from the TEXPROS document repository. Folder organization seems to be able to handle very large heterogeneous enterprise document collections and is available for any user. The variety of users who wish to organize folders is broad. The user may need an overall view of the document collection: what kind of document types exist, what kinds of frame instances exist, how the frame instances are related, and so on. On the other hand, each user may want to find a specific piece of information in which the user is more interested. For instance, the user may need memos received from "John Smith" and the received dates are between "09/01/1998" and "10/01/1998". A common feature for all the tasks mentioned is that the user does not know exactly what he/she is looking for. Hence, providing a graphical visual browsing technique is useful for both trained and non-trained users.

4.8.2 Dynamic Versatile User Workspace

The design criterion for the Internet document management system is usability. The user interface design is the use of knowledge about the psychology of interaction between the users and the machines. A given view of a document can be recognized by its layout structure without reading word by word or character by character. If this view composes a set of hierarchical folders and each folder contains a set of associative links of documents, all the information should be clear enough for browsing. The layout views can be used to assist the user obtain a better understanding of the flow of content. The flow can be represented as the visual hierarchical representation to get quick ideas of the meanings from the view. This structured Internet folder organization that can easily deal with a huge number of documents in the database.

The reactive interaction is based on active rules between documents and software agents. The event is passive behavior and the action is active behavior. By providing the reactive interaction and folder organization, we propose to construct a dynamic versatile user workspace.

A distinct but complementary motivation for the "dynamic versatile user workspace" is to overcome many problems of the current generation of various user interface approaches. For example, the MS Windows file manager is not designed to organize the content structure of the documents. From 1984, direct manipulation interfaces [Hutchins, Hollan, and Norman 1986] have become the standard GUI design, which has made a huge improvement over the command line interfaces. But the lack of scalability and dynamic features create some limitations for the direct manipulation interfaces. In the past several years, interface design has been oriented toward

conversational interfaces, where the user and the agent "take turns" acting [Lieberman 1991]. Current interface agents aim at different approaches with the concept of indirect management style of interaction [Kay 1990].

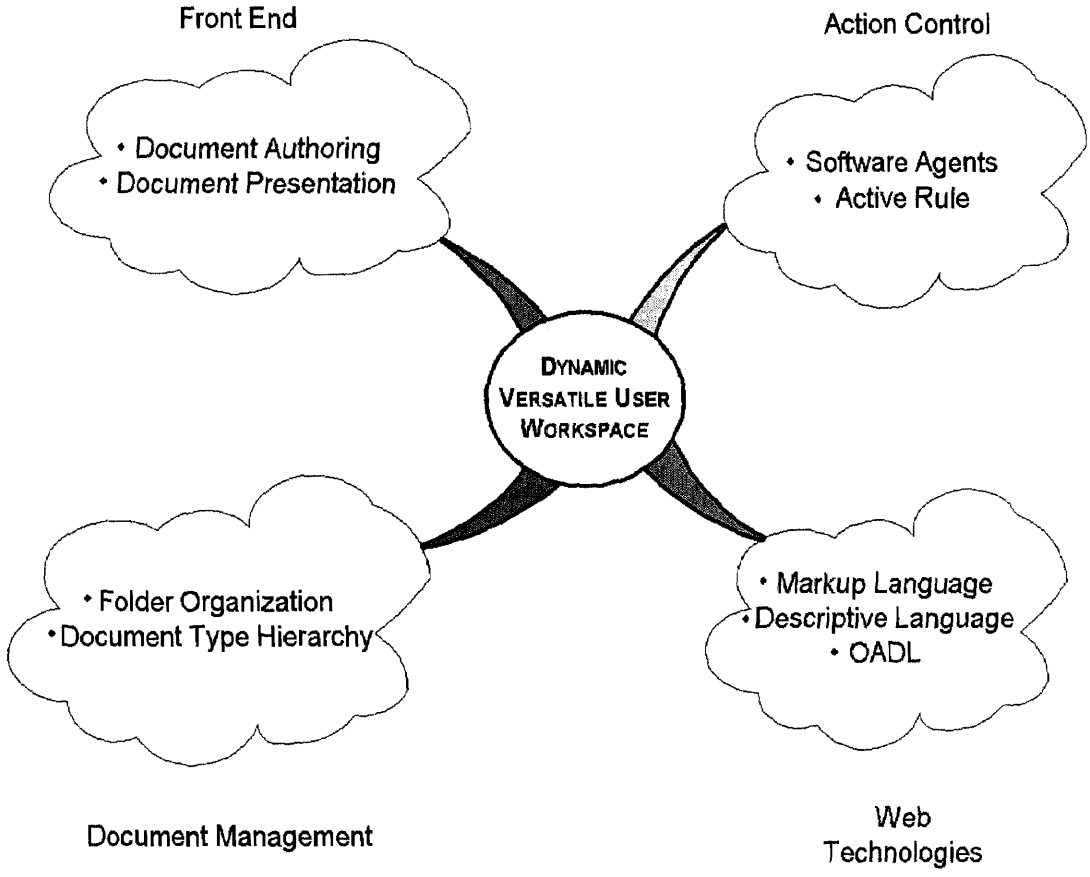

Figure 24 Dynamic Versatile User Workspace

In Internet user interface design, the human-computer interaction allows users to access documents in a wide area network. When using a *Graphical User Interface*, a user

uses a computer to navigate from place to place and perform operations by using a mouse to click on locations, icons and toolbars within windows rather than pressing keys on a keyboard to perform the same tasks. Even though many people believe that the Graphical User Interface concept was invented by Apple Computer, Inc and first introduced on its Macintosh computer; in fact, the first GUI was developed by the Xerox Corporation at its Palo Alto Research Center in the 1970's. WWW was created under the assumption that a group of users could work collaboratively by putting information on a Web of markup documents. The Web browser allows a user to access information stored on a remote location. The Web server responds to the browser's request and sends this document to his browser. His browser interprets the document and displays it on his screen.

The "dynamic versatile user workspace" gives a significant improvement in the usability of the document management system. Figure 24 shows a dynamic versatile user workspace comprises of four parts: front-end processes, action control, document management, and Web technologies.

Front-end Processes

The front-end processes deal directly with the users. The processes in the front-end are the document authoring process and the document presentation process. The document authoring is a process for users to create documents. The document presentation process generates the layout of the document and creates the links for other documents.

The document creation process presents a big challenge when we want to create heterogeneous types of documents. Different types of documents must have different

components. Each document is composed of a set of components or elements. We call the components "attributes". A generic form for document authoring should be convenient and efficient. That is why the document creation process is semi-automatic. We need to manually create some templates for each document type.

Document presentation deals with the layout style to display the document on the computer screen. The style and composition of the elements make up an interface. We need to display pieces of data separately, and also set up a complete picture for putting the pieces together. There are three criteria for the presentation process. The document presentation keeps it simple, active, and uses the computer-generated masks or annotations to attract more attention on some important features. For example, given a display of a letter, we can use different colors to show various significant elements composed in the letter. The document presentation keeps it consistent to certain styles for each particular document type. It is not always a good idea for professional users to reinvent the wheel for the look and feel of every document type. The document presentation gives a series of single designs and is incredibly helpful. We call these designs "layout templates". The layout for such documents should be dynamically created. That is why we need to create some layout templates for each particular document type. Each template can be manually designed to meet different situations. For example, we design the Memo type documents as five different layout templates: confidential layout, rush layout, meeting layout, appointment layout, or normal layout.

Document Management

TEXPROS is the core document management system in the dynamic versatile user workspace. To migrate the conventional document management system to the Internet-ready system, the initial requirement is that the user interface be changed. The ubiquitous software is the Internet Web browser. We want to set up the workspace platform as the Web-base because users are already familiar with the usage of the Web browser.

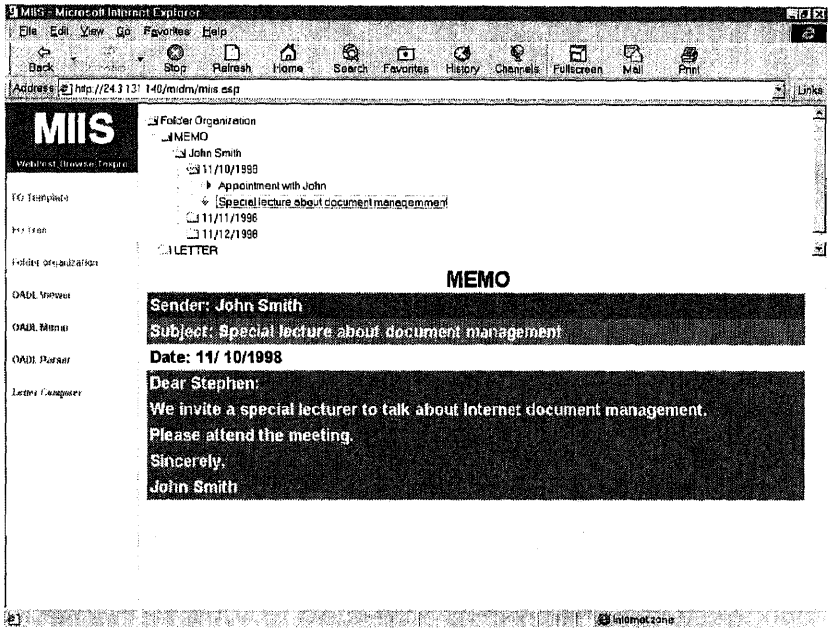

Figure 25 Folder Organization User Interface

The greatest advantages of folder organization are flexibility and efficiency of use. Folder organization can speed up document browsing for the expert users and inexperienced users alike. Current Windows file manager or the UNIX file system needs the user to memorize each individual document's stored location. Therefore, users

normally find it difficult to memorize all the document locations. Figure 25 shows the folder organization user interface that organizes different types of documents into a hierarchical view. This hierarchical view can be shown on the Internet browser screen. We use the linking facilities to create grouping and formatting in the hierarchical view to make the documents navigable.

Action Control

CSA handles the action control that involves the rule-based event handling. The action control manipulates the document data. CSA explores the document base to retrieve the links and creates the folder organization. CSA also evaluates the condition criterion. Since querying and retrieving data are the most important functions in document management, we design FO-QL to perform the retrieve action.

Web Technologies

Web technologies include descriptive language, markup documents, and OADL. The important feature of descriptive language is the ability to define the interactive event that triggers a specified function.

OADL defines the document markup and linking in the TEXPROS domain. Electronic mail is one of the document delivery methods in a Peer-Peer paradigm. Automatic Folder Organization Generation is also a Peer-Peer paradigm. If we apply the distributed evaluation to create the folder organization, we see that the process performs a Coordinator-Executor paradigm.

4.9 Conclusion

We live in a world where memos, letters, proceeding articles, journals, reports and a whole host of other document types are common place. Yet there are significant differences between these documents in terms of layout, style, content, function, usefulness, size and so forth. The categorization of data is the subject of research in information retrieval. De Beaugrande (1980) says that the data categories are not mutually exclusive and are not distinguishable in any one dimension. However, we use folders to categorize the data, making it more capable than other research.

FO-QL provides support for querying, constructing, transforming, and integrating the OADL data. The OADL data is very similar to semistructured data [Abiteboul 19976, Buneman 1997, Fernandez, Popa, and Suciu 1998, Fernandez, Florescu, Levy,and Suciu 1997], which has been proposed in the database research and unstructured data community as a data model for data sources that have irregular or rapidly evolving structure. We have designed FO-QL based on our TEXPROS experience for organizing heterogeneous type of documents. From FO-QL, we can export the document frame instances to cyberspace and organize the specified document repository.

Markup technology is powerful because it introduces the hyperlink [Smish and Weiss 1988]. The Internet folder organization consists of many folders that contain the hyperlinks of the associative documents. To navigate the document base, one chooses a particular folder that consists of the hyperlinks. When users click on a link they can retrieve a document from a remote location. The user holds the mouse cursor over the folder, and the menu shows up on the screen. The screen presents those hyperlinks and/or subfolders and lets the user select which documents to retrieve.

The features of the folder organization are scalability, flexibility, and navigability. Even though the document repository is changing all the time, FO-QL can help to dynamically reorganize the changing folder organization in the changing environment of an enterprise document management system.

CHAPTER 5

SIGNIFICANT CONTRIBUTIONS

This dissertation mainly has two contributions: implementation of an agent-supported document management system and definition and test of a Folder Organization Query Language (FO-QL). Active database currently is a popular research topic. However, documents with active features pose a great challenge for us to deal with. The issue is that documents with embedded rules acting like active databases still have not utilized agent (or AI) support. Therefore, we apply agent technology to document management and call these agent-supported OADL documents Active Documents. When we produce a document, we employ certain knowledge about what the document is. The question is, how this knowledge can be described and reconstructed in a computational model. Documents with embedded rules acting like active objects are answers for this knowledge model. With embedded rules and self-describing data features, Active Documents, which can supply solutions for these problems, provide collaborative interactions with software agents. FO-QL declares the hierarchy of a customized folder organization that CSA uses to query the document repositories.

5.1 The Advantages of Active Document

Active Document is composed of the markup data and markup active rule. Markup features completely insulate the application logic from the heterogeneous systems, so that CSA should be able to apply its knowledge to interact and handle Active Documents. CSA composes and de-composes Active Documents when needed. This means that the

markup data are expandable and modifiable by CSA. In an Active Document, an active rule only defines what to do and leaves the process logic to CSA. Therefore, Active Documents provide much more flexibility than normal documents.

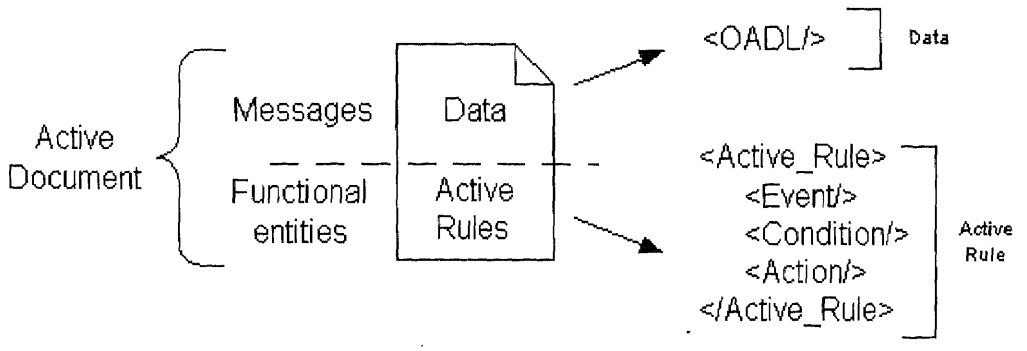

Figure 26 Active Documents

Figure 26 shows that an Active Document comprises data and active rules. The main content of the document consists of the data portion which can be modified or expanded. Data are the represented messages or the memory of the knowledge representation. Active rules are functional entities. They are represented as the action knowledge of the Active Document. CSA can dynamically change the data and active rules of the Active Document any time it is needed. Thus, the Active Documents are fully controlled by CSA.

Active Documents' knowledge is understood in a dynamic sense. Document knowledge is a summarized representation of action knowledge. Action knowledge is understood in terms of special purposes. In other words, Active Documents define the actions that should be treated as purpose-oriented documents. The active rules should be explained by means of modeling actions.

5.2 Active Documents Contribute to TEXPROS

Manually browsing a document repository to find the right document is time consuming and unrealistic especially in multiple document repositories. A folder organization query process automatically retrieves document links that are deposited into folders. A folder organization is constructed in a centralized document repository or in a distributed document management environment. TEXPROS includes the classification subsystem, information extraction subsystem, filing subsystem, storage subsystem, and retrieval subsystem.

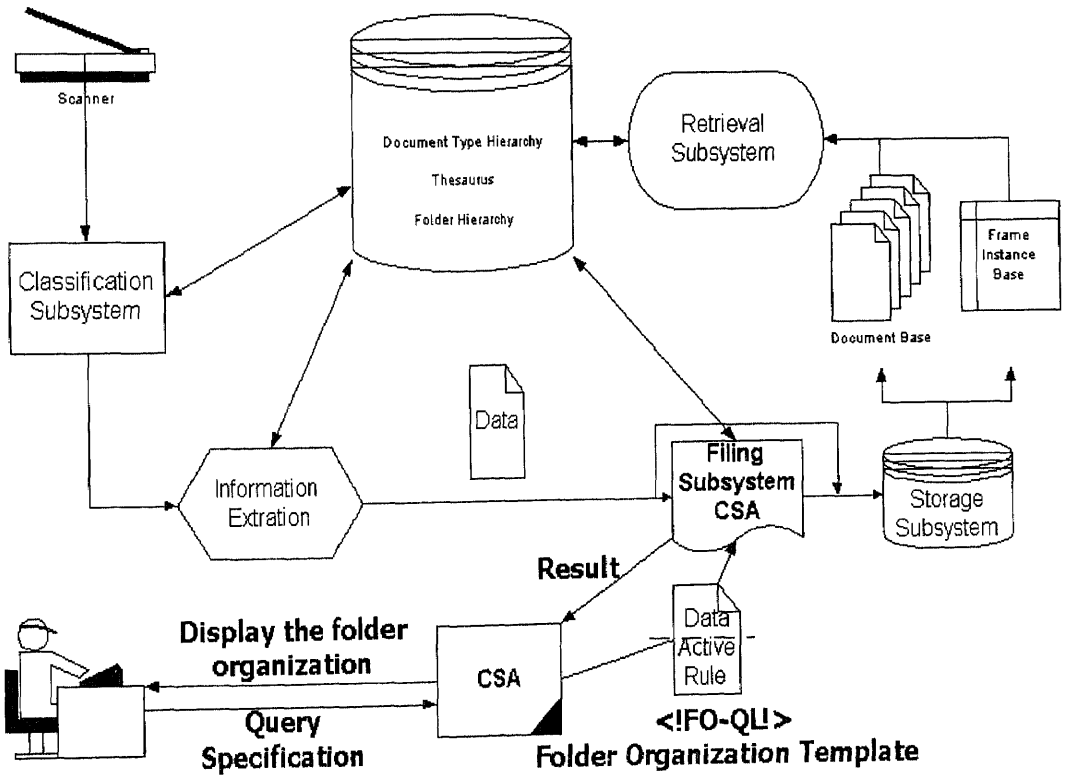

Figure 27 CSA supports TEXPROS

FO-QL constructs a customized folder organization that needs to have CSA to generate an Active Document with the FO-QL active rules and then send it to the filing subsystem. This Active Document with the FO-QL active rules is called a folder organization template. The filing subsystem starts the folder organization generation process.

Figure 27 shows that CSA supports TEXPROS document management. In the filing subsystem, we make an enhancement to execute the folder organization generation process and parse the folder organization document. We can say that there is a CSA in the filing subsystem that is responsible to process the folder organization template and browse the storage subsystem that is the document repository.

To create a folder organization, a user first needs to create the query specification. To formulate a query, a user must select collections, metadata descriptions, or information sets against which the query is to be matched, and must specify words, phases, descriptors, or other kinds of information that can be compared to or matched against the information in the collections [Baeza-Yates and Ribeiro-Neto 1999]. Shneiderman [Shneiderman 1997] identifies five primary human-computer interaction styles. These are command languages, form filing, menu selection, direct manipulation, and natural language. Recently most commercial full-text systems only supported limited Boolean queries. To improve standard Boolean syntax, a filter/flow model [Young and Shneiderman 1993] provides users with a direct-manipulation method. The user is shown a scrollable list of attribute types on the left-hand side and selects attribute types from another list of attribute types shown across the top of the screen. We modify the filter/flow model to fit into our query specification user interface.

5.2.1 Problems Inherent in Paper-based Documents

Traditional paper-based documents still have a lag time of up to several days while in transit via regular mail and a substantial amount of handling, key-entry, and processing time required at both sending and receiving sites [Sokol 1995].

TEXPROS's classification and extraction subsystems can translate hard copy documents into digitized documents. In a digital format, we can encode data items such as document types, document attributes etc. by using the markup tags. With markup features, we can easily index those documents by their important data items. Therefore, CSA can process and retrieve those documents.

5.2.2 Ubiquitous Work Environment

FO-QL is designed to generate a customized folder organization for users to categorize documents and retrieve documents more efficiently. From a user's perspective, the World Wide Web (WWW) is a global information repository, which connects different geographic areas and heterogeneous systems. Hence, making the specified document repositories available everywhere has advantages for end users to retrieve documents everywhere. Web browsers are ubiquitous, so both local and global users can have controlled access to TEXPROS document repository via WWW.

Web browsers and markup documents are platform-independent, so they are not concerned with which client systems are used. The Web's thin client computing paradigm means that most of the tasks are done by the server applications. The user does not have the administration overhead and it is easier to distribute modifications. CSA is designed to fit into the server application to provide great robotic usage.

5.2.3 Simultaneously Browse Multiple Document Repositories

FO-QL is designed to allow CSA to propagate the query to multiple document repositories. Distributed evaluation is the answer for browsing multiple document repositories. Distributed evaluation proceeds as follows: (1) Select document repositories to search. (2) Distribute the folder organization template to the selected document repositories. (3) Evaluate the query to execute the folder organization query process in parallel. (4) Combine results from the distributed document repository into a final result to form a customized folder organization.

At this point we have covered everything except how to merge the results. First, the issued folder organization template is the same for every document repository. Second, the hierarchy of each partial folder organization should be similar. Third, each document link points to the physical document location. From the three scenarios, we can join multiple results without conflict.

5.3 Active Documents Contribute to Workflow Automation

The principal of the declarative representation of the active rule, which can be applied to the workflow automation, makes an Active Document an appropriate instrument for the representation of a scheduled plan.

5.3.1 From a Centralized to a Distributed Environment

A user can store documents in a distributed environment without a centralized control. In this case, there is no central control of the documents and the question is how to browse the document repositories without remembering the specific locations of the documents.

5.3.2 Active Documents on Electronic Commerce

From a business process perspective, electronic commerce is the application of technology toward the automation of business transactions and workflow [Kalakota and Whinston 1996].

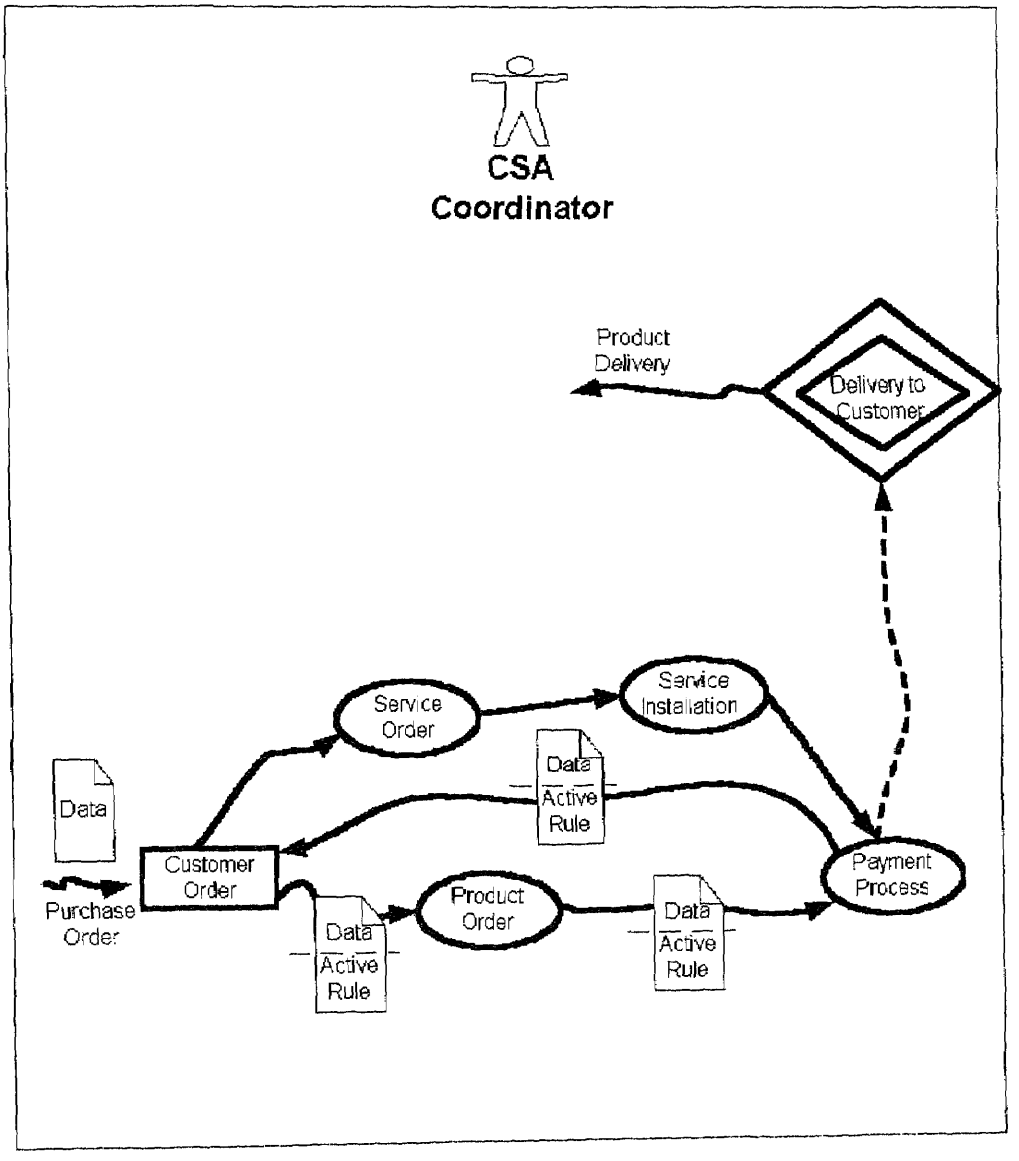

Figure 28 Active Documents on Electronic Commerce

Workflow automation via the Internet is revolutionizing the way software does computing because each process is taking place at a different location. During workflow, assume each process comes with a corresponding document that stores the required data and procedures for what needs to be done.

The problem is that different systems or groups normally have difficulty in communicating with each other or the data can't be shared because of document format. Even in the same location, they still encounter heterogeneous file formats. We use the same ubiquitous data format – markup to keep the same meaning for different parties. Markup provides a common knowledge interface that uses the same ontology to communicate. The active rule in the markup document provides a functional interface during document flow with declarative functionality.

Figure 28 shows the flow of the Active Documents during electronic commerce. A CSA coordinator moderates the whole process of the electronic commerce. Each shape represents a process that a CSA coordinator may dispatch an active rule to an executor. A CSA coordinator controls the workflow as the document's states change. Each executor work independently applying its own ontology to interpret the Active Documents. Different executors use the Active Documents as the media to communicate and interact. Thus, these executors increase the comparability and efficiency of the whole workflow automation.

5.3.3 Improve Execution of Transaction over Networks

We say that a transaction normally happens at different locations. The transmission of the states and messages may generate too much data for the normal workflow process. With

the Coordinator-Executor model, each Active Document only records the required data for each process. The coordinator keeps all the information of the whole transaction. Thus, more effective performance and rapid exchange are created. With portable features of Active Documents, the coordinator can also simultaneously parallel dispatch to multiple executors for high speed, accelerated, or real-time interaction.

5.3.4 Share Documents with Heterogeneous Systems

Markup features completely insulate the application logic from the heterogeneous systems. Therefore, the markup documents can be easily shared by the heterogeneous systems. CSA interprets and recognizes those markup documents using the same ontology. Then, CSA applies its own rule base to implement the actual process. This method creates an advantage in hiding the real implementation in each CSA and leaves the documents free to be transported with restorations.

CHAPTER 6

RELATED WORKS

There is a great deal of research on active database processes and irregular data types. There are some studies on software agents and active rules in active database systems. They are SQL3 [ISO/IEC 1996], AMOS [Risch, Skold 1992; Fahl,Risch,and Skold 1993], NAOS [Collet, Coupaye, and Svensen 1994, Collet 1996] and ARIEL [Forgy 1982; Hanson 1996]. Some work close to FO-QL are Lightweight Object REpository Language (LOREL) [Quass, Rajaraman, Sagiv, Ullman, and Widon 1995], UnQL [Buneman, Davison, and Suciu 1995], and XML-QL [Deutsch, Fernandez, Florescu, Levy, and Suciu 1998].

6.1 Related Works of Active Documents

There has been a significant amount of research on active databases that is related to Active Documents. We call an OADL document an Active Document due to the support of software agents. However, the active database system projects either do not have a software agent's support; do not address any data exchange model with a heterogeneous system; or do not address any distributed computation.

6.1.1 SQL3

More recently, SQL3 [ISO96] introduced active database functionality in the form of triggers. A trigger in SQL3 is a named event-condition-action rule that is activated by a database state transition [Kulkarni, Mattos, Cochrane 1998]. SQL3 applies to both

relational and object-oriented database systems. The database engine is responsible for detecting the event and executing the action. Each event will create a trigger execution context after or before some operation is applied to the specified table. The triggering operations need to be INSERT, DELETE, or UPDATE on the specified table. Those triggering operations can be tuple-based or statement-based events. Each tuple-based operation is triggered by an operation on each row of the table. Each statement-based operation is executed once per statement of the query operation.

SQL3 provides standard support for commercial database systems. The following is a simple example to monitor the inventory table. After updating the inventory table, the monitor_inventory trigger will be initiated.

```
CREATE TRIGGER monitor_inventory
AFTER Update on Inventory
FOR EACH ROW
BEGIN ATOMIC
    Select quantity, max_quantity, threshold
    Into my_quantity, my_max_quantity, my_threshold
    From Inventory, Merchant
    Where Merchant.Name=:new.Name;
    If (my_quantity < my_threshold) then
        Order (:new.Name, my_max_quantity-quantity)
    End if
END
```

There is a tremendous shortcoming in the SQL3. Like the previous example, SQL3 does not specify the execution semantics for the monitor_inventory's order action. Different database systems need to develop their own execution semantics.

6.1.2 AMOS

AMOS is the abbreviation of Active Mediators Object System [Risch and Skold 1992, Fahl, Risch, and Skold 1993]. AMOS is based on object-relational DBMS. The condition checking is based on the technique called "Incremental evaluation". The data model contains objects, types, and functions and is based on the functional data model of Daplex [Shipman 1981] and Iris [Fishman etc. 1989]. The execution model applies the AI's production rules, which are defined in AMOSQL. AMOSQL is the query language of AMOS and is based on OSQL [Lyngbaek 1991]. The production rules are also called CA rules. The condition (C) is a query, and the action (A) is a procedural expression. For example, we want to monitor the quantity of each item in the inventory. We can create a rule like:

```
CREATE RULE monitor_inventory() as
FOR EACH ITEM i
WHEN quantity (i) < threshold (i)
DO order (i , max_quantity(i)-quantity(i));
```

The AMOS rule is executed after all the transactions are finished and the Boolean expression is always evaluated to be true. Any instance that causes the Boolean

expression to be false will not fire the action. This ensures that the net changes only apply to the total condition checking. Such behavior is very similar to the SQL3's statement-based execution.

6.1.3 NAOS

NAOS is the abbreviation of Native Active Object System [Collet, Coupaye, and Svensen 1994, Collet and Machado 1995]. It incorporates an active behavior within the object-oriented database management system O2 [Bancilhon, Delobel, and Kanellakis 1992]. The execution model is inherited from the Flexible Active Rule Execution (Fl'ARE) [Coupaye and Collet 1997]. Fl'ARE is an ECA execution style. The events to be triggered are creation, deletion, access, update, and method call. A predicate holds if the result of the corresponding query is not empty [Collet 1998]. The condition specifies the predicate of any data that needs to be evaluated. If the evaluation is true, the action will be fired. The action performs some procedures or methods defined in the system. For example, we want to monitor the quantity of each item in the inventory. We can create a rule like:

```
Create rule monitor_inventory
coupling immediate
on after update merchant->quantity with e
if  new(e)->quantity < threshold (new(e)->name)
do {
    order(new(e)->name,max_quantity(new(e)->name)-new(e)->quantity);
      }
```

The NAOS supports immediate and deferred rules that are defined after the keyword coupling. Immediate coupling means that after each tuple of data is altered, the rule needs to be evaluated. Deferred coupling means that after accumulating all the events in one transaction only one rule will be initiated. Therefore, deferred rules are like the statement-based operation.

6.1.4 ARIEL

ARIEL [Hanson 1996] is a research project at the University of Florida, which is tightly coupled with a query processor. ARIEL mainly focuses on a condition-testing mechanism for rules and rule evaluation and is based on a discrimination network with the TREAT algorithm [Miranker 1987]. It is a relational database model. The execution model is based on a production rule and applies the TREAT algorithm. The query language, POSTQUEL, with rules is called ARIEL Rule Language (ARL) and uses a discrimination network for rule condition testing.

For example: Define a rule which will never let the product's quantity be less than the threshold quantity.

```
Define rule monitor_inventory
on replace inventory
if inventory.quantity< inventory.threshold
then append order(inventory.name, inventory.max_quantity-
inventory.quantity)
```

The Ariel rule system uses a production system model. The rule system is set-oriented. This means that it is statement-based in SQL3 or the deferred model in AMOS.

Each database transition is executed after a simple command or a list of commands between **do ... end** block [Hanson 1998].

6.1.5 Comparisons

The comparisons between Active Documents and related systems focus on the data model, execution model, and coupling. Active Documents are based on the Document Object Model that is a markup-based data model. All the related systems are based on a database data schema that may be Object-Oriented and/or a relational data model.

Table 24 Active Documents compared with related systems

	Active Documents	Trigger of SQL3	AMOSQL of AMOS	Fl'ARE of NAOS	POSTQUEL of ARIEL
Data Model	OADL	Relational or Object-Oriented database	Object-Relational database	Object-Oriented database	Relational database
Execution Model	ECA descriptive rule system	TRIGGER	AMOSQL	Flexible Active Rule Execution	POSTQUEL with TREAT algorithm
Coupling	Immediate	Immediate	Deferred	Immediate or deferred	Deferred

6.2 Related Works of FO-QL

Data integration of heterogeneous information has attracted great research interest from the database and Internet community. However, most of the work just applies to the standard SQL or OQL query of the unstructured or untyped data. Some work addresses the query of semistructured data but still misses the distributed feature of diverse information sources. The TEXPROS's Folder Organization [Fan and Ng 1998] gives an excellent example for organizing documents in an efficient and fluent way. I extend the features of the TEXPROS's Folder Organization to query and organize office documents, such as OADL documents, in a centralized or distributed environment and generate a virtual hierarchical view of a user's document repositories. I will list and compare some of the related works with FO-QL.

6.2.1 Lightweight Object REpository Language (LOREL)

LOREL is the abbreviation of Lightweight Object REpository Language. LORE (Lightweight Object Repository)[Quass, Rajaraman, Sagiv, Ullman, and Widom 1995} is a general-purpose data management system for semistructured data and LOREL is its query language. The data model is based on semistructured data with the Object Exchange Model (OEM) [Papakonstantinou, Garcia-Molina, and Widom 1995]. The OEM model was originally introduced for the Tsimmis project [Papakonstantinou, Garcia-Molina, and Widom 1995]. For example, a book can be described using the OEM model as:

book:&01

{Author:&02 "elmasri",

Author:&03 "navathe",

TitleofBook:&04 "Fundamental of Database Systems",

Publisher:&05 "Addison-Wesley",

Date: &06{ Month:&07"December",

Year:&08 1997},

ISBN:&09 "0-8053-1748-1"}

The OEM data model is represented as a rooted, labeled graph and the object is represented as a set of data. The execution model of LOREL is derived from OQL [Cattell 1996], an object-oriented query language. The following LOREL query returns the author of a book written in 1990.

```
select Book.Author
from Book
where Book.Date.Year > 1990
```

The meaning of this query is as follows: (1). Find all the documents with the root element Book. (2). Traverse the root to the possible paths to find Year followed by Date and Book has a value greater than 1990. (3). Evaluate whether or not the condition is true and then return the value of Author followed by Book.

6.2.2 UnQL

UnQL stands for Unstructured Query Language [Peter Buneman, Susan Davidson, Mary Fernandez, Gerd Hillebrand, Lucian Popa, and Dan Suciu 1995]. The data model is the same as LOREL's edge-labeled graph but objects are represented as a bag of data. The execution model comes with a mathematical query construct called "structural recursion" that provides a datalog like rule expression. For example:

```
g({})={}
g(l:t)= if l=book then h(t)
        else {l:g(t)}
h({})={}
h({l:t})= if l=Date then i(t)
          else if l=Author then
i({})={}
i({l:t})= if l=Year and t>1990 then
            g(book:{Author:l})
```

The above query returns the author of books written since 1990.

6.2.3 XML-QL

XML-QL is a Query Language for XML [Deutsch, Fernandez, Florescu, Levy, and Suciu 1998]. The data model is an ordered tag-based element that contains character data

(CDATA) only (strings). The execution model uses pattern match to query the desired document repository, such as any particular Web site. The query uses Where-Construct query style. For example:

```
WHERE <book>
        <Author>$a</>
        <Date><Year>$y > 1990</></>
      </>
Construct   <BookAuthor>$a</>
```

The above query returns the author of books written since 1990.

6.2.4 Comparisons

The comparisons between FO-QL and related works focus on the data model, the execution model, type checking, distributed computing and values returned. FO-QL is mainly defined to query the OADL documents with distributed evaluation. FO-QL has type checking for the markup data to optimize the query processing and gives a more accurate result. The related works all generate a set of objects but FO-QL outputs a folder organization.

Table 25 FO-QL compared with related systems

	FO-QL	LOREL	UnQL	XML-QL
Data Model	OADL documents	Semistructured data	Semistructured data	XML documents
Execution Model	Descriptive markup	SQL or OQL like	Datalog (structural recursion)	SQL or OQL like
Type Checking	Yes	No	No	No
Distributed Computing	Yes	No	No	Yes
Values Returned	A folder organization	A set of objects	A bag of objects	A set of tagged results

CHAPTER 7

CONCLUSION

TEXPROS's powerful features include Document Type Hierarchy and Folder Organization. Taking advantage of TEXPROS, OADL is designed for the current distributed Internet environment, such as thin clients and object technologies. Some basic ideas for using OADL to design the Document Type Hierarchy and Folder Organization with CSA are proposed here. The basic findings and ideas may be summarized as follows:

The values of using tagged-documents are interchangeability, flexibility, and extensibility. The frame instances can be transferred into any OADL documents, which can be interchanged among systems. They are flexible because we allow presentation and authoring of documents to be specified interactively without explicit programming. They are extensible because the uses can define the domain document type and organize the related documents to be the Document Type Hierarchy. Moreover, we can extend the functionality by defining more elements in the definition. Therefore, Active Documents are born with active rules embedded in the OADL documents.

CSA supports both the Peer-Peer and the Coordinator-Executor paradigms of document management. These paradigms are built on the definition of active rules. The active rules provide the functional entities in the markup document. Even though active rules are only part of a prototype of a process, such flexibility can give CSA more independence to execute the active rules. The Peer-Peer paradigm is used in electronic mail notification. Electronic mail is one of the major mechanisms for current Internet

communications. CSA traces the sending document and handles feedback notification. The Coordinator-Executor paradigm is used to create a workflow automation process. The active rule, along with CSA, provides the event-condition-action and event notification capabilities.

FO-QL is designed to be the principal for a folder organization query process and to customize an Internet folder organization that is used to organize and link globally document instances. Making the specified document repositories available everywhere is a big advantage. End-users can retrieve documents everywhere as long as the network is on. Web browsers are ubiquitous, so any users can access the TEXPROS document repository via WWW.

7.1 Open Research Issues

Areas that are open to research include 1. Experiment with detailed codes inside active rules. Currently, Active Documents only declare the prototype of the implementation. 2. Extend FO-QL to accommodate a natural language query and generate a more sophisticated user friendly folder organization. 3. Design a real-time distributed object to simulate a versatile CSA. The real-time process is a great challenge for Internet computing. With some compromises, we can design semi-real- time workflow automation. 4. Make knowledge ontology consistent with broad knowledge acquisition. It is very important to represent documents consistently. 5. Protect Active Documents from unauthorized usage for security issues. The above five issues are much more complex and are areas for open research issues.

REFERENCES

1. Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., and Simeon J., 1997, *Querying Documents in Object Databases.* International Journal on Digital Libraries, 1(1). pp. 5--19.

2. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J., 1997, *The Lorel Query Language for Semistructured Data.* International Journal on Digital Libraries, 1(1): 68-88, April 1997.

3. Abitelboul S., 1997, *Querying Semi-Structured Data.* In Proceedings of 6th International Conference on Database Theory, Delphi, Greece, January 8-10, 1997.

4. Akpotsui E., Quint V., and Roisin C., 1997, *Type Modeling for Document Transformation in Structured Editing Systems*, Mathematical and Computer Modeling, Volume 25, Number 4, pp. 1-19.

5. Allen C., 1997, *WIDL: Automating the Web with XML*, in World Wide Web Journal, Volume 2, Number 4. 1997.

6. Baeza-Yates R., and Ribeiro-Neto B., 1999, *Modern Information Retrieval.* ACM Press, Addison Wesley, Harlow, England.

7. Bancilhon F., Delobel C., and Kanellakis P., 1992, *Building an Object-Oriented Database- The Story of O2.* Morgan Kaufmann, San Francisco, CA.

8. Barker R., 1990, *CASE*Method: Entity Relational Modeling.* Addison-Wesley, Reading, MA.

9. Berners-Lee T., 1989, *Information Management: A Proposal,* CERN European Laboratory for Particle Physics, France, March 1989.

10. Berners-Lee T., 1996, *Keynote Address*, Seybold San Francisco, CA, February 1996.

11. Booch G., 1994, *Object-Oriented Analysis and Design with Applications* 2nd Edition, Addison Wesley, Reading, MA.

12. Boy G.A., 1991, *Intelligent Assistant Systems.* San Diego, CA.: Academic Press.

13. Branding H., Buchmann A.P., Kudraβ T., and Zimmermann J., 1993, *Rules in an Open System: The Reach Rule System.* In M. Williams N. Paton, editor, Rules in Database Systems, Proceedings of 1st Intl. Workshop on Rules in Database Systems, Edinburgh, Scotland.

14. Broy M., 1993, *"Functional Specification of Time-Sensitive Communicating Systems"*, ACM Transaction Software Engineering and Methodology, pp.1-46 January 1993.

15. Buchmann A.P., Branding H., Kudraβ T., and Zimmermann J., 1992, *Reach: A Real-Time Active and Heterogeneous Mediator System.* Bulletin of the TC on Database Engineering. December 1992.

16. Buchmann A.P., Zimmermann J., Blakeley J. and Wells D., 1995, *Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions.* In Proceedings of 11th Intl. Conference on Data Engineering, Taipei, Taiwan, March 1995.

17. Buneman P., 1997, *Tutorial: Semistructured Data.* In PODS 1997.

18. Buneman P., Davidson S., and Suciu D., 1995, *Programming Constructs for Unstructured Data.* In proceedings of the 1995 International Workshop on Database Programming Languages (DBPL).

19. Buneman P., Davision S., Hillebrand G, and Suciu D., 1996, *A Query Language and Optimization Techniques for Unstructured Data.* University of Pennsylvania, Computer and Information Science Department Technical Report, Number 96-09, 1996.

20. Carr L. A., Hall W., and Hitchcock S., 1998, *Link Services or Link Agents?* Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems, 1998, pp. 113 – 122.

21. Collet C. and Coupaye T., 1997, *Architecture and Implementation of the NAOS Active Rule System.* In "In Preparation," 1997.

22. Collet C. and Machado J., 1995, *Optimization of Active Rules with Parallelism.* In Proceedings of the Intl. Workshop on Active and Real-Time Database Systems (ARTDB-95), Sweden, June 1995.

23. Collet C., 1998, *Active Rules in Database Systems* Edit by Paton W. Norman. Springer-Verlag, New York, NY.

24. Collet C., Coupaye T., and Svensen T., 1994, *NAOS Efficient and Modular Reactive Capabilities in an Object-Oriented Database System.* In Proceedings Of the 20th Intl. Conf. On Very Large Database, pp. 132-143, Santiago, Chile, September 1994.

25. Comer, D. E., 1995, *"Internetworking with TCP/IP Volume I Principles, Protocols, and Architecture"* 3rd ed., Prentice-Hall, Englewood Cliffs, NJ.

26. Connolly D. and Bosak J., 1998, *Extensible Markup Language (XML) 1.0 Specification.* W3C Recommendation, Feb 1998.

27. Coupaye T. and Collet C., 1997, *FL'ARE: a Flexible Active Rule Execution Model.* Technical Report. Technical report, LSR-IMAG, University Joseph Fourier, Grenoble, France, July 1997.

28. De Beaugrande R., 1980, *Text, Discourse and Process.* Ablex, Norwood, NJ;

29. Dertouzous M., 1997, *What Will Be*, HarperEdge, San Francisco, CA.

30. Erickson T., 1997, *Designing Agents as if People Mattered. In Software Agents,* ed J.M. Bradshaw. AAAI Press, Melo Park, CA.

31. Fan X. and Ng P.A., 1998, *Personal Document Management and Retrieval: A Knowledge-Based Approach*, Journal of Systems Integration, 8(2).

32. Farley J., 1997, *Java Distributed Computing,* O 'Reilly & Associates, Sebastopol, CA.

33. Fernandez M., Florescu D., Levy A., and Suciu D., 1997, *A Query Language and Processor for a Web-site Management System.* In Proceedings of the Workshop on Management of Semi-structured Data.

34. Fernandez M., Popa L., and Suciu D., 1997, *A Structured-Based Approach to querying Semi-Structured Data.* In Proceedings of the Workshop on Database Programming Languages, 1997.

35. Forgy C.L., 1982, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.* Artificial Intelligence, 19:17-37.

36. Goldman R. and Widom J., 1997, *DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases.* Proceedings of the Twenty-Third International Conference on Very Large Data Bases, pp. 436-445, Athens, Greece, August 1997.

37. Goldman R., McHugh J., and Widom J., 1999, *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language.* To appear in Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999.

38. Graesser A. and Clark L., 1985, *Structure and Procedures of Implicit Knowledge*, Ablex, Norwood, NJ.

39. Grudin J. and Palen L., 1995, *Why Groupware Succeeds: Discretion or Mandate*? Proc. ECSCW'95, Kluwer, Dordrecht, The Netherlands, pp. 263-278.

40. Hanson E.N., 1996, *The Design and Implementation of the Ariel Active Database Rule System.* IEEE Transactions on Knowledge and Database Engineering, 8(1): 157-172 February 1996.

41. Hanson E.N., 1998, *Active Rules in Database Systems* Edit by Paton W. Norman. Springer-Verlag, New York, NY.

42. Hewitt C., 1977, *Viewing Control Structure as Patterns of Passing Messages*, Artificial Intelligence, 8, No 3, pp. 323-364.

43. Hutchins E. L., Hollan J. D., and Norman, D. A., 1996, *Direct Manipulation Interfaces. In User-Centered System Design,* eds. D. A. Norman and S.W. Draper, pp.87-124, Hillsdale, N.J.: Lawrence Erlbau.

44. ISO 8879, 1986, *ISO 8879, Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML),* 1986.

45. ISO/IEC 1996, *Working Draft: Database Language SQL (SQL3).* ISO/IEC DBL MAD-007, June 1996.

46. Kalakota R., and Whinston A.B., 1996, *Electronic Commerce: A Manager's Guide.* Addison-Wesley Longman, ACM Press, New York, NY.

47. Kay A., 1990, *User Interface: A Personal View.* In The Art of Human-Computer Interface Design, ed. B. Laurel, pp.191-208. Addison-Wesley, Reading, MA.

48. Khare R. and Rifkin A., 1997, *X Marks the Spot: Using XML to Automate the Web*, in IEEE Internet Computing, Volume 1, Number 4, pp. 78-87, July/August 1997.

49. Kim J. and Yoo B., 1998, *Does Every Link Have the Same Usability? An Exploratory Study of the Link Structure of Cyber Malls* Proceedings of the CHI 98 summary conference on CHI 98 summary: Human factors in computing systems, 1998, pp. 315.

50. Lieberman H., 1997, *Autonomous Interface Agents*, CHI 97.

51. Lin J.T., 1999, *Office Automation Definition Language (OADL),* Dissertation Proposal, Computer and Information Science, New Jersey Institute of Technology, Newark, NJ.

52. Lin J.T., Shen N.H., Chu T.M., Doong S., Curtis R., Hung D.C., and Ng P.A., 1999, *Folder Organization Query Language,* Proceedings on the Fourth World Conference on Integrated Design and Process Technology (IDPT), Kusadasi, Turkey, June 27-July 2, 1999.

53. Liu Q.H. and Ng P.A., 1996, *Document Processing and Retrieval: TEXPROS*, Kluwer Academic Publishers, Boston, MA.

54. Lyngbaek P., 1991, *OSQL: A Language for Object Databases.* Technical Report HPL-DTD-91-4, Hewlett-Packard Laboratories, January 1991.

55. Maes P., 1997, *Agents that Reduce Work and Information Overload.* In Software Agents, ed. J.M. Bradshaw. AAAI Press, Melo Park, CA.

56. Maler E. and DeRose S., 1998, *XML Linking Language (XLINK)* World Wide Web Consortium Working Draft 3-March-1998.

57. Malone T. W., Grant K. R., Lai K.Y., 1997. *Agents for Information Sharing and Coordination.* In Software Agents, ed J.M. Bradshaw. AAAI Press, Melo Park, CA.

58. Malone T. W., Grant K. R., Lai K.Y., Rao R.,and Rosenblitt D., 1987a, *Semistructured Messages are Surprisingly Useful for Computer-Supported Coordination*, ACM Transactions on Information Systems Vol. 5, No. 2 (April 1987), pp.115-131.

59. Malone T. W., Grant K. R., Lai K.Y., Turbak F.A., Brobst S.A., and Cohen M.D., 1987b, *Intelligent Information-Sharing Systems.* Communications of the ACM 30,5(May), pp. 390-402.

60. McHugh J. and Widom J., 1999, *Query Optimization for XML.* Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases, Edinburgh, Scotland, September 1999.

61. McHugh J., Abiteboul S., Goldman R.,Quass D., and Widom J., 1997, *Lore: A Database Management System for Semistructured Data.* SIGMOD Record, 26(3): 54-66, September 1997.

62. MIRANKER D.P., 1987, *TREAT: A Better Match Algorithm for AI Production Systems.* In Proc. AAAI National Conference on Artificial Intelligence, pp. 42-47, August 1987.

63. Mittal V.O., 1999, *Generating Natural Language Descriptions With Integrated Text and Examples*, Laurence Erlbaum Associates, Mahwah, NJ.

64. Negroponte N., 1997, *Agents: From Direct Manipulation to Delegation.* In Software Agents, ed. J.M. Bradshaw. AAAI Press, Melo Park, CA.

65. Nwana H. S. and Azarmi N., 1997, *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence; Concepts and Applications.* Springer-Verlag, Chicago, IL.

66. Paton W.N. and Diaz O., 1998, *Active Rules in Database Systems* Edit by Paton W. Norman. Springer-Verlag, New York, NY.

67. Pearl J., 1988, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA.

68. Quass D., Rajaraman A., Sagiv Y., Ullman J., and Widom J., 1995, *Querying Semistructured Heterogeneous Information.* In proceedings of the Fourth International Conference on Deductive and Object-Oriented Database (DOOD), pp. 319-344, Singapore.

69. Quass D., Widom J., Goldman R., Haas K., Luo Q., McHugh J., Nestorov S., Rajaraman A., Rivero H., Abiteboul S., Ullman J., and Wiener J., 1996, *LORE: A Lightweight Object REpository for Semistructured Data.* Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada, June 1996.

70. Raggett D., Hors A. L., and Jacobs I., 1998, *HTML 4.0 Specification.* World Wide Web Consortium Recommendation, April 1998.

71. Rick J., 1998, *The XML and SGML Cookbook: Recipes for Structured Information.* Prentice Hall, Englewood Cliffs, NJ.

72. Robie J., Lapp J., and Schach D., 1998, *XML Query Language (XQL).* W3C QL'98 The Query Languages Workshop.

73. Robie J., Lapp J., and Schach D., 1998, *XQL: A Query Language for XML Data.* W3C QL'98 - The Query Languages Workshop.

74. Rothkegel A., 1993, *Text Knowledge and Object Knowledge.* Communication in Artificial Intelligence Series, United Kingdom.

75. Salton G., 1968, *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, pp. 381-383.

76. Shneiderman B., 1997, *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, Reading, MA.

77. Smiss J.B and Weiss S.F., 1998, *An Overview of Hypertext.* In Communications of the ACM, July 1988 Vol. 31, No. 7.

78. Sokol P.K., 1995, *From EDI to Electronic Commerce: A Business Initiative.* McGraw-Hill, New York, NY.

79. Stotts P.D., Furuta R., and Cabarrus C.R., 1998, *Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking,* ACM Transactions on Information Systems, pp.1-30, January 1998

80. Suciu D., 1996, *Query Decomposition and View Maintenance for Query Languages for Unstructured Data.* In Proceedings of the International Conference on Very Large Data Bases , 1996

81. Suciu D., 1998, *Semistructured Data and XML,* In Proceedings of International Conference on Foundations of Data Organization, 1998.

82. Takahashi K., 1998, *Metalevel Links More Power to Your Links* Commun. ACM 41, 7 (Jul. 1998), pp. 103 – 105.

83. van Dijk, T.A. and Kintsch, W., 1983, *Strategies of Discourse Comprehension,* Academic Press, New York, NY.

84. Wang J.T.L. and Ng P.A., 1992, *TEXPROS: An Intelligent Document Processing System.* International Journal of Software Engineering and Knowledge Engineering, 15(4): 171-196, April 1992.

85. Winograd T. and Flores F., 1986, *Understanding Natural Languages,* Academic Press, New York, NY.

86. Winograd T. and Flores F., 1987, *Understanding Computers and Cognition: A New Foundation for Design,* Addison-Wesley, Reading, MA.

87. Wood W. J., 1985, *What's in a Link? Foundations for Semantic Networks.* In R. Brachman and H. Levesque. Editors., Readings in Knowledge Representation, Morgan Kaufmann, Los Altos, CA.

88. Young D. and Shneiderman B., 1993, *A Graphical Filter/Flow Model for Boolean Queries: An Implementation and Experiment.* Journal of the American Society for Information Science, 44(6): pp.327-339, July 1993.