Hypermedia Document and Workflow Management Based on Active Object-Oriented Databases

G. Kappel, S. Rausch-Schott, S. Reich, W. Retschitzegger
Dept. of Information Systems
Johannes Kepler University of Linz
A-4040 Linz, Austria
{gerti|stefan|sre|werner}@ifs.uni-linz.ac.at

Abstract

Challenging application domains for workflow systems such as electronic publishing and technical engineering require equally progressed document management features. This is due to the fact that document management within these application domains is characterized by huge amounts of data, highly structured documents, as well as hypermedia information. Integrating existing services for workflow management based on an active object-oriented database system with document management based on commonly accepted international standards like SGML (Standard Generalized Markup Language) and HyTime (Hypermedia/Time-based Structuring Language) allows to benefit from the advantages of both technologies. This results in document-based workflow management systems which can cope both with frequently changing organizational requirements and with the exchange of platform independent interlinked documents across system boundaries.

1. Introduction

In large engineering enterprises business is critically dependent on the provision of effective means for people to access and collaboratively use all types of technical as well as business information. In addition to this issue of document management, the issue of workflow management is seen as key supportive technology of information systems for a collaborative environment [17, 18].

This paper describes how document and workflow services can be elegantly realized on top of an active object-oriented database system. In particular, we combine rule based mechanisms for workflow enactment with object-oriented modelling techniques in order to deal with extensibility and flexibility issues, and extend our existing workflow prototype system towards hypermedia services in order to cope with challenges being posed by structured, interlinked, and heterogeneous documents.

The paper is structured as follows: Section 2 presents an application scenario from which the requirements for our work are derived. Section 3 and 4 present the design decisions for hypermedia document and workflow management in order to cope with the defined requirements. In Section 5 we introduce the architecture of our prototype system to put our design decisions into practice. We finish with a summary and discussion of our approach in Section 6.

2. Requirements for WFMSs: An Application Scenario

In order to illustrate the requirements which have driven the extension of our workflow management system TriGFlow with hypermedia document management facilities, in the following a typical application scenario in the area of technical engineering is described. This scenario represents the maintenance process for a product (in our case a motorcycle) which is based on feedback from customers.

Figure 1 shows the activities comprising the workflow, i.e., the so-called activity net, as well as the documents used by these activities. Note, that in order to reduce complexity, the agents responsible for performing the activities have been omitted.

2.1. Description of the Scenario

The workflow is started as soon as a certain number of error reports is received by the manufacturer of the product from several garages. These error reports contain inquiries for changing a certain part of the product, e.g., the starter because of recurring operating problems stated by customers. The main idea of this workflow is to handle such operating problems in a proper way, i.e., to decide (cf. activity 1) on
the basis of the analysis of the error reports whether to redesign (parts of) the product including all necessary steps (cf. activities 2 and 5 to 10), or whether to recommend new service instructions (cf. activity 3), or to report recurring problems to the management only (cf. activity 4). The analysis activity (cf. activity 1) groups related problems within the error reports and summarizes them within an analysis report. In case of a redesign (cf. activity 2), which produces a new technical drawing and a new bill of material, the costs for this redesign are computed. If the resulting bill of costs indicates that a redesign is not affordable, a decision between activities 2 to 4 has to be made. Otherwise, necessary parts have to be produced (cf. activity 6) and/or ordered (cf. activity 7) according to the new bill of material. As soon as all parts are available, a prototype can be assembled (cf. activity 8) on the basis of the technical drawing. In the meantime, the owners manual is updated (cf. activity 9) on the basis of the old and new bills of material and the drawing.

After finishing the assembly of the prototype and updating the documentation, the prototype has to be tested for three months. Note that this is an activity external to the organization and thus is not shown in the workflow. After these three months, the effect of the redesign is evaluated (cf. activity 10) by means of error statistics received from the garages. This evaluation activity has to be done three months after activity 3, too. The resulting evaluation report also flows into succeeding report analyses (activity 1).

2.2. Derivation of the Requirements

Naturally, the derivation of requirements leads to a large number of different requirements of different granularity. We thus have grouped the identified requirements into the following categories:

- heterogeneity comprising the requirements R1, R2, R3, R4, R5, and R6.

- configurability at runtime with the requirements R7, R8, R9.

- other requirements comprising R10, R11 and R12.

At first sight, it can be recognized that the workflow shown in Figure 1 has to deal with a large number of heterogeneous documents (requirement R1), which on the one hand have different origins resulting in different formats (R2) and on the other hand have to be processed by different activities, probably using different tools individually available on the user's desktop (R3). The documents may be structured, such as SGML conformant documents (R4, see below Section 3.1), or time-based, such as videos and sound (R5), and may include links to other documents (R6), such as a maintenance manual including a link to a video (see Figure 1). In addition to that, the platform independent addressing of (parts of) structured documents is inevitable. As is typical for manufacturers, different activities are likely to be performed at different sites (R7) using different tools (R3).

Moreover, in such a workflow, the kind and order of activities are usually not fixed. Rather it should be possible to configure the workflow according to actual needs at runtime (R8). For example, it might be possible, that the decision whether to redesign or not is not only based on the bill of costs but rather on an external expertise, too. Then a new activity "request expertise" has to be integrated into the activity net.

As soon as new activities are introduced, new types of documents for these activities may arise (R9), which could be, for example, an expertise document. Another example would be that the incorporation of a new garage into the company probably results in error reports which are structured differently to existing reports. Further important dynamic changes, which have to be supported in workflow systems, concern the assignment of agents to activities (R10) and policies for the processing of orders assigned to a specific agent (R11).

A general requirement of workflow systems, which can be seen also in our scenario, is that the enactment of activities is based on events (R12). In other words, the start of activities or even of the workflow itself should be automatically triggered by certain (predefined) events. For instance, a policy can be that the first task of our workflow, i.e., the analysis of error reports, is started after 30 error reports have been received, or at the latest at the end of every month. As the scenario shows, there are a lot of requirements for workflow management systems including flexibility, extensibility and interoperability issues. In the following, we identify technologies for an integrated document and workflow management system that could tackle these requirements.

3. Hypermedia Document Management

As already illustrated with our scenario, the specification of a workflow comprises among others not only specifying the process by means of a number of related activities, but also specifying the data flow in form of documents which are used by the activities. Especially concerning technical engineering applications, documents can be of a very complex nature. Unfortunately, the flexible management of such hypermedia documents is hardly dealt with in existing systems.

To achieve the above stated requirements we try to integrate three basic technologies. By integrating active facilities with an object-oriented database system, dynamic adoption to the frequently changing requirements of a workflow system can be achieved. This has already been done within our
research prototype TriGSflow. But now we go a step further and try to enrich this system with powerful document management features. In order to cope with the heterogeneity of structured hypermedia documents within a distributed environment we propose to use well agreed standards such as SGML and HyTime as basis for flexible document management.

In the remainder of this section we discuss the problem of representing hypermedia documents. In Section 4 then, we show the usage of hypermedia documents in our workflow system prototype.

3.1. Representing Structured Documents

In large engineering enterprises, cooperation is mainly based on the exchange of documents, but most tools rely on different data formats. Thus, the problem of document exchange is hard to solve (cf. Requirement R1). To make document exchange possible, a system- and device-independent data representation format for hypermedia documents within a WFMS is required (cf. R1, R2, R7, R9). Besides the aspect of device independent storage it is important that the information being handled is of high quality in terms of query, markup and structuring (cf. R4). Another aspect of hypermedia publications concerns the multimedia nature of information, i.e., the documents representing the information are compound documents containing besides text additional types of media, such as technical drawings, images, videos of maintenance instructions, and sound (cf. R4, R5).

WFMSs demand for standardized as well as extensible document interchange formats. The Standard Generalized Markup Language (SGML) [12, 15, 22, 23] — a well established ISO-standard for structured documents — and its hypermedia "extension" Hypermedia/Time-based Structuring Language (HyTime) [6, 7, 9, 16, 20, 21] clearly are standards which allow the platform independent exchange of time-dependent structured data. Thus, we consider the support of these two standards as crucial for WFMSs.

With respect to the above mentioned issue of interoperability, SGML is considered the most significant standard for information exchange. It allows the system independent specification and handling of a documents contents and structure (cf. R1, R2, R4). SGML standardizes the syntax for tagging documents. Thus, it can be used to represent arbitrary document types, e.g., a biography of an artist, a spreadsheet with references to other documents, or a hypermedia presentation. For example, Figure 2 depicts an SGML document instance of type "analysis" as is used in the scenario described in Figure 1.

From a documents viewpoint the processing during a workflow looks as shown in Figure 3. The way of pro-
ing as input and a new, modified version as output document. In a WFMS, many structured documents exist, thus also parts of documents, e.g., a certain section, can be assigned to tasks.

However, SGML suffers from some impediments in link management, multimedia, and data location, which are needed in WFMSs. The Hypermedia/Time-based Structuring Language HyTime extends SGML by addressing the problems of:

- Locating data of any type by using a flexible standard notation that is independent of the WFMS and the data itself (cf. R2 and R6).
- Describing links within and between documents (cf. R6).
- Describing relations between temporal and spatial events occurring in documents (cf. R5).

HyTime is characterized by its “meta” definition capabilities. It consists of a set of basic element types called architectural forms. These may be re-used for specific workflows. Architectural forms can be compared to abstract classes in object-oriented programming.

According to the HyTime standard we specify the element types indoc, outdoc and subend for referring to the documents involved in a workflow. We refer to the actual documents via hyper-links, in particular, via so-called contextual links (clink). The contextual link allows a reference to a location to be made in context, which means that one end of the link is the element itself, in our case this is the task, and the link-end is a “pointer” to the referenced document.

A contextual link can point to a document in different ways. One might for instance want to address a document directly by its name — then the architectural form called name location (namedloc) can be used. In Figure 4 this would result in accessing the analysis report directly by its name, i.e., analyses\30596. One might also want to refer to the first item of the second error list of an analysis report — in this case a tree location (treeloc) may be useful (see element Listitem with subject “Cannot start...” in Figure 2). In addition, documents can be addressed by using path locations (pathloc). Path locations are very similar to name locations except for that they are addressing a document not only by its individual name but also by a (system) path expression. For our purposes, we use name and tree locations. Figure 4 gives a graphical representation of these addressing facilities.

### 3.3. Representing Structured Documents within an Object-Oriented Database

The usage of SGML/HyTime as interchange formats demands for modelling possibilities. There exist various ap-
proaches in the literature to make databases — especially object-oriented databases — SGML aware, i.e., to allow them to deal with SGML conforming documents [1, 8, 24]. In addition, as is shown in this section, for elegantly and efficiently modelling SGML documents concepts such as metaclass mechanisms are indispensable.

SGML can be considered as a meta language or metaclass, respectively. The instances of a metaclass, i.e., classes, therefore are the document-type-definitions being defined in SGML. The actual documents then are instances of classes, i.e., they can be compared to objects. Figure 5 depicts this relationship.

<table>
<thead>
<tr>
<th>SGML</th>
<th>dtd</th>
<th>documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 8879</td>
<td>book.dtd</td>
<td>charlesGoldfarb.xls</td>
</tr>
<tr>
<td></td>
<td>newspaper.dtd</td>
<td>times.newspaper.xls</td>
</tr>
<tr>
<td></td>
<td>HTML.dtd</td>
<td>faz.newspaper.html</td>
</tr>
<tr>
<td>MetaClass</td>
<td>Class</td>
<td>Instances</td>
</tr>
<tr>
<td>1</td>
<td>some</td>
<td>many</td>
</tr>
</tbody>
</table>

Figure 5. Metaclass—Class—Instances

Being restricted to a fixed set of document-type-definitions (dtds) does not appear to be worthwhile. It has to be possible to insert new or modified DTDs at runtime without affecting running workflows. Thus, we have extended GemStone to be able to intuitively handle SGML documents. Thereby, we make use of GemStones metaclass mechanism. This allows us to generate classes for element types at runtime and thus to add specific type information to each instance of an element type. Figure 6 expresses these different approaches. Rounded elements depict instances, white rectangular boxes are classes, and metaclasses are displayed as shaded rectangular boxes. The main difference is that by the use of metaclass mechanisms we are able to dynamically generate a GemStone class for each corresponding SGML element type during runtime (see left part of Figure 6).

Our research framework allows us to deal with arbitrary types of information because we are able to deal with any SGML document-type-definition. By defining a meta DTD which is a DTD defining DTDs we are able to treat dtds like "normal" documents. Such a meta DTD has been developed in [14].

4. Incorporating Hypermedia Document Management into TriGS_flow

Within the last section we have shown that standards like SGML/HyTime are an appropriate means in order to model structured documents used in a WFMS and how such documents can be represented within an object-oriented database system. Now we describe how this hypermedia document management system is used by our WFMS TriGS_flow.

As we have seen within our scenario, one of the most challenging requirements of workflow management systems is to provide concepts allowing flexible reaction to changes during workflow enactment (cf. R8, R10, R11). To achieve this, we have integrated three basic technologies when developing TriGS_flow. Firstly, we use object-oriented database technology to build a workflow management system providing both database functionality and possibilities for modelling and reusing complex business domain objects [10]. Secondly, to cope with changes of the personnel, roles have been integrated into the object-oriented environment separating activities from particular persons executing the activities. Thirdly, those parts of a workflow subject to frequent changes, such as the ordering of activities, are modeled by using ECA rules which may be adapted dynamically.
The basic model used in TriGS_flow to represent a business process including activities, agents, and documents, comprises a number of classes within an object-oriented setting and a number of ECA-rules. The classes comprise a generic workflow model in the sense that different kinds of workflows such as application for scholarship, reimbursement of business trips, and reordering of goods are modelled by specialising and instantiating the corresponding classes of the generic workflow model. For a detailed description of the object-oriented model we refer to [17].

Figure 7 shows the relevant classes of the object-oriented model of TriGS_flow which are used for the representation of hypermedia documents.

We currently distinguish between four types of documents which can be contained within the folder: SGML documents (SGML-Document) are structured according to the SGML standard. Unstructured Documents (UnstructuredDocument) are documents like, e.g., bitmapped images, whose structure has to be interpreted by the appropriate document processing tool, e.g., the viewer for a corresponding image. Time-Based Documents (TimeBasedDocument) such as videos for demonstrating maintenance instructions, are to be handled, too. Finally, also objects from the database itself can be referenced from within a folder (DataObject). Additional document types can be easily incorporated by deriving new corresponding subclasses from the class Document (cf. R9).

The following subsections not only illustrate the different application areas of ECA rules (Event-Condition-Action rules) within our WFMS TriGS_flow thus tackling some of our requirements but also show how to work with hypermedia documents within these rules. All examples are based on our application scenario which was given at the beginning.

### 4.1. Activity Ordering

Activities have to be executed in a certain order determined by their relationships within an activity net. These relationships comprise basic control structures, namely sequencing, branching, and joining. A detailed discussion of the different kinds of branching and joining can be found in [17].

Considering the realization of an activity net such as for example the one shown in Figure 1, requirement R8 “Runtime modification” can be easily met by mapping all relationships between activities to ECA rules (Event-Condition-Action rules) according to the basic control structures [18]. If the order of activities is to be changed, simply the corresponding rule has to be changed, which can be done even dynamically, i.e., while the workflow is running. It is also possible to specify several alternative configurations of the activity by means of several rules, and to experiment between these alternatives just by activating and deactivating appropriate rules.

The event of an activity ordering rule always constitutes the end of the preceding activity or — in case of an AND-Join or of an inOR-Join (inclusive OR) — of all participating preceding activities, respectively (cf. R12). The condition checks whether the succeeding activity has to be performed or not possibly by accessing a document represented by SGML. This is done by evaluating queries against the actual folder and/or other data concerning the actual workflow. In case of an inOR-Branching the result sets of the conditions may overlap, whereas an exOR-Branching (exclusive OR) is reflected by mutual exclusive conditions. AND-Branching is realized by using the same condition in all branches.

The action of an activity ordering rule has to notify the agent responsible for performing the succeeding activity. The method notifyAgent puts a corresponding item for the successor activity into the worklist of the agent chosen to perform this activity (for selecting agents see Section 4.2). The method perform starts the execution of the activity no matter whether a human agent or a software agent is engaged. In case of a human agent it waits until the human agent signals the end of the execution. Every rule responsible for activity ordering contains the methods perform and notifyAgent in its event part and action part, respectively. With this, it is possible to process different activities using tools individually available on the users desktops (cf. R3 and [18]).

As an example, consider the following rules, realizing an exOR-Branching between the activities “reports analysis”, “redesign”, “recommend new service instructions” and “create report” (cf. activities 1 to 4 in Figure 1). The rules named R.Redesign, R.NewServiceInstr, and R.CreateReport are activated for the same instance of class Activity, namely a.reports.analysis. That is, after the method perform has been executed on that instance (denoted by the keyword POST), all three rules are triggered. The conditions of the rules R.Redesign, R.NewServiceInstr, and R.CreateReport are complementary, checking on the basis of the reports analysis whether a redesign is necessary (which is assumed to be the case if the percentage of conceptual errors, i.e., errors of type CONCEPTUAL, is above 10 percent), new service instructions are to be recommended (percentage of conceptual errors is less than 10 percent and the percentage of errors of type maintenance is high), or whether the error reports cannot be reacted upon properly at time (type of error is other) and, thus, are reported to the management, respectively. Since the analysis reports are structured SGML documents the requested information can simply be retrieved by means of querying attributes of the document. The action of the rule for which the condition evaluates to true is executed by sending the message notifyAgent to the respective instance.
Figure 7. Part of the Object-Oriented Model of TriGSpflow

(a.redesign, a.new_servicelnstr, or a.create_report) of class Activity representing one of the mutual exclusive successors of the activity a.reports.analysis within the activity net (cf. Figure 1).

DEFINE RULE R.Redesign
ON POST (Activity, perform) DO
IF ((actFolder docNamed: analysis report 310596)
getElems: subject)
detect: [:item | item getAttrValue: errorType = conceptual] ...
THEN
EXECUTE a.redesign notifyAgent
ACTIVATED FOR (a.reports.analysis)
END RULE R.Redesign.

DEFINE RULE R.NewServiceInstr
ON POST (Activity, perform) DO
IF ((actFolder docNamed: analysis report 310596)
getElems: subject)
detect: [:item | item getAttrValue: errorType = conceptual] ...
THEN
EXECUTE a.new.servicelnstr notifyAgent
ACTIVATED FOR (a.reports.analysis)
END RULE R.NewServiceInstr.

DEFINE RULE R.CreateReport
ON POST (Activity, perform) DO
IF ((actFolder docNamed: analysis report 310596)
getElems: subject)
detect: [:item | item getAttrValue: errorType = other] ...
THEN
EXECUTE a.create.report notifyAgent
ACTIVATED FOR (a.reports.analysis)
END RULE R.CreateReport.

4.2. Agent Selection Policies

During workflow enactment, the assignment of activities to agents is done on the basis of agent selection policies [4, 18]. An example of an agent selection policy is that agents on holidays are not allowed to be selected. The decision which agent(s) to select for a specific activity is based on data within both, the history of the actual workflow, and the documents actually contained within the folder. In order to keep the selection process flexible (cf. requirement R10), these policies are encoded into ECA rules. In this way, on the one hand, immediate reaction to state changes of some agent, like a machine breakdown, is allowed, and on the other hand, different selection policies can be realized and switched between by simply activating and deactivating the corresponding rules at activity level. Note, only one specific selection rule can be active for a certain activity at one point in time. For example, the following rule realizes a policy based on the qualification necessary for handling the specific error as described within the analysis report:

DEFINE RULE R.Qualified
ON PRE (Activity, notifyAgent) DO
IF (receiver possibleAgents selQualifiedFor:
((actFolder docNamed: analysis report 310596)
getElems: subject)
THEN
EXECUTE (receiver actAgRoles) removeAll;
add: Query.Result
ACTIVATED FOR (a.redesign)
END RULE R.Qualified.
4.3. Worklist Management

As has been shown in the previous section, on arrival of the folder at a certain activity a subset of the agents associated to that activity is selected on the basis of a specific policy. Each of the selected agents is notified about the job to be done by inserting an appropriate order into its worklist. The agent itself is responsible for executing the orders in its worklist. In TriGS\textsubscript{flow}, such an agent might be either a human agent representing a human able to perform the activity, or a software agent, i.e., an automatic process able to perform some requested activity without user interaction. In case of a human agent, the execution of orders has to be done in interaction with a certain user. In contrast, software agents are able to perform their orders without user interaction. Instead, ECA rules are used to coordinate these orders (cf. R11). The rules, e.g., monitor the worklists of software agents and start processing the next order every time a software agent becomes inactive. This state is indicated after an order is removed from the worklist. Similar to activity ordering and agent selection, the coordination of worklist items might also be done on the basis of the document to be processed. For example, assume that a designer not only gets orders for redesigning parts of a product resulting from our workflow scenario, but also gets orders for designing new products or versions of a product, which come from other workflow types. The designer usually can select any order from her worklist to be performed next. However, if it is further assumed that redesign tasks are more urgent than new design tasks, redesign orders should be favored. The following rule realizes that policy by assigning a high priority to a redesign order as soon as it is inserted into the designers worklist:

\begin{verbatim}
DEFINE RULE R.SetHighPriority
ON POST (Worklist, insert: item) DO
IF (item activity = a.redesign) THEN
EXECUTE item setProcessingPri: #high
ACTIVATED FOR (AllDesigners worklist)
END RULE R.SetHighPriority.
\end{verbatim}

5. Prototype Architecture

The set of requirements has led to the development of an overall architecture depicted in Figure 8. The functional architecture providing the requested workflow and document management services is based on a modular approach by separating the whole system into components as is shown in the figure. In the following we describe the architectural components of the TriGS\textsubscript{flow} prototype in detail.

TriGS\textsubscript{flow} has been implemented as a layer on top of the object-oriented database system GemStone extended by the component TriGS providing ECA rules and a module providing roles. Currently, two front-end tools named MasterTOOL and AgentTOOL are being developed providing a graphical user interface for the document and workflow modeling phase and the workflow enactment phase, respectively.

The GemStone \cite{5} database management system is a commercially available object-oriented database system. Its data model is based on the GemStone Smalltalk programming language, which is a derivate of Smalltalk-80 \cite{11}. In addition to Smalltalk, GemStone Smalltalk provides some predefined classes in order to realize database functionality. Considering query facilities, GemStone Smalltalk provides a limited query language allowing associative access to objects residing in a single collection. To speed up the retrieval of objects, identity indexes as well as equality indexes can be defined. Concerning concurrency control, GemStone supports optimistic as well as pessimistic mechanisms.

The Roles module used in our prototype is based on the work of \cite{13}. A role hierarchy can be specified orthogonal to the class hierarchy permitting GemStone objects to learn and forget roles, i.e., behavior, dynamically and to play several roles at the same time. In TriGS\textsubscript{flow}, roles are used to specify capabilities and duties of agents, consequently determining the assignment of agents to activities. For details of the role model applied in TriGS\textsubscript{flow} we refer to \cite{17}.

The active component of TriGS\textsubscript{flow} is based on TriGS (= Trigger System for GemStone) \cite{19}. TriGS has been implemented on top of GemStone and makes explicit use of objects, message passing, inheritance, and overriding to provide a seamless integration between rules and the object-oriented data model of GemStone. TriGS rules consisting of events, conditions, and actions monitor the behavior of objects and can be attached to specific classes or defined inde-
pendently of any class hierarchy. Rules can be (de)activated at different levels of granularity ranging from the object instance level to the object class level. Moreover, since rules and its components are first-class objects they can be dynamically defined, modified, and extended independently of any application. TriGS_flows employs rules for activity ordering, specifying organizational policies, and worklist management (cf. sections 4.1, 4.2, and 4.3).

The SGML component of our system allows to manage arbitrary SGML documents, i.e., any document type definition and its instances can be stored and manipulated by the SGML layer. We make use of the Perl SGML parser. The SGML layer also builds the basis for a future HyTime engine (see also below on ongoing work). We are currently working on the implementation of the HyTime hyperlink module which seems especially useful for our purposes.

Especially for document management the inclusion of external events is crucial. Most active object-oriented databases so far handle only events raised internally by the database system; external events can be manually invoked by a prescribed API (Application Program Interface). To overcome this deficiency, we are currently implementing a dedicated event manager for handling external events. Similar to the Apple Macintosh operating system we distinguish between three kinds of events [2]: low-level events, such as mouse, keyboard, insert disk, or null event (= no other events to report), operating-system events, e.g., if an application is sent in background the operating system sends a background event; and high-level events between applications for communication, e.g., requesting a dictionary application for information on particular word. For the third case, an event message protocol is necessary. Our current implementation is based on version 2.5 of the Solaris operating system and uses the Unix monitoring features as basic event mechanism.

6. Summary and Future Work

Existing WFMSs hardly address the management of hypermedia documents which is required by application domains like electronic publishing and technical engineering. Therefore, we have integrated an WFMS prototype system with the possibility of representing hypermedia documents within an object-oriented database system. Hypermedia documents are modelled by using the SGML/HyTime standard and incorporated into the active object-oriented WFMS TriGS_flows. With this combination, issues like flexibility, interoperability and extensibility can be dealt with in a natural and intuitive way. The architecture of our prototype system has been illustrated.

A very interesting and promising issue with respect to event based mechanisms which has to be further investigated is to what extent TriGS_flows can benefit from the so-called "activity tracking" mechanism of HyTime. Activity tracking in HyTime is done by referencing activity elements which define actions to be taken and the conditions under which they will be taken. The standard defines the activity types "create", "modify", "link", "access", "unlink" and "delete" [16, page 32]. The accessing of an activity by a human agent, for instance, is an activity of type access, the triggering conditions could be a specific user name, and the frequency of access, and the action taken could be the logging of the access action in a log file.

For workflow management and document management this sort of triggering mechanism seems to be especially useful for the following situations:

- Notification: the owner of an accessed document might be notified of the use of his document.
- Intellectual property rights: the information that a document is accessed may be used for presenting a copyright notice as well as for billing the user.
- Triggering: different actions might be triggered by so-called encapsulated data handlers according to the type of access to an object. The modification of an object could invoke a notification message for all objects referencing this modified object.
- Monitoring: information like "who worked at a certain time on which document" can be easily recorded in a log.

The main benefit of using HyTimes activity tracking lies in a standardized, platform independent representation of event based mechanisms.

Another issue which is further considered is how far our integration of SGML/HyTime documents into the object-oriented database system GemStone could form the basis for a future HyTime engine. A "HyTime engine" is "a program (or portion of a program or a combination of programs) that recognizes HyTime constructs in documents and performs application-independent processing of them"[16, page 5].

HyTime support is comparatively easy as opposed to SGML support. As already mentioned, HyTime standardizes a set of element-type forms which are called architectural forms. These architectural forms are fixed specializations of element-types. For example, hyperlinks as known from the World-Wide Web document-type-definition HTML would be modelled according to HyTime as follows [9] (Note that HyTime uses SGML as means for expressing its semantics).

```xml
<ELEMENT A ->{PCDATA|BR|IMG|emph;}*>
<ATTLIST A
  href |URL| #IMPLIED
  id |ID| #IMPLIED
  HyTime NAME |clink>
```

385
The usage would stay the same, e.g.,

```
<some html code>
```

Thus, the HTML did would have to be changed as described above by that making all HTML document instances HyTime conforming. A HyTime engine then could do a basic processing of hyperlinks and would thus guarantee standardized link management. In a first step we are currently working on the implementation of parts of the HyTime “hyperlinks” module which we believe is especially important for our purposes. The implementation is based on previous experience we have made with the development of HyTime architectural forms in the application area of workflow management.

References


