

XDoC-WFMS: A Framework for Document Centric Workflow Management System

Rupa Krishnan

Lalitha Munaga

Kamalakar Karlapalem

International Institute of Information Technology,
Gachibowli, Hyderabad - 500019, INDIA
www.iiit.net; {rupa,lalitha}@dit.iiit.net; kamal@iiit.net

Abstract. Document Management is an important component of a workflow management system. XML has become a prominent language for document processing and management, and has an additional advantage of managing persistent documents using XML document management system. In this paper, we develop an **XML Document Centric Workflow Management System (XDoC-WFMS)** for specifying, executing and controlling workflows. A workflow consists of a set of interrelated activities. These activities involve accessing multiple documents, each of which is encapsulated within a micro-agent. A micro-agent is a system agent within an XDoC-WFMS that performs document management (i.e., retrieving, concurrent updating, and transferring documents). The XDoC-WFMS supports seamless integration of document processing within a workflow, flexible and dynamic management of XML documents, micro-agent based event and exception handling, and secure document access.

1. Introduction

Information flow through an organization can be both complex and intricate. This sophistication arises from the specific rules and constraints within business processes inherent in each organization. There are some processes that are wholly centered on document movements. Some common examples of this are Newspaper Editing, Graduate Admissions, Job Application Processing and tendering and procurement. Document flow through an organization can be modeled in terms of workflows. With the advent of XML, most of the organizations are adopting XML as the document description language. Documents in XML have certain inherent structure, which can be exploited to have a document flow specification. Documents have sub-documents that can be collaboratively worked on by different users. Any document processing requires routing of documents, checking the integrity of the documents, retrieving and combining various (sub-)documents for an activity to start executing, and finally, once the activity completes execution, handling the (sub-)documents and routing them to other activities.

A workflow in a business organization is more than just document processing, it involves other activities, such as, authorization/approval, executing certain activities and communicating certain decisions. Therefore, there is a need to have a seamless integration between document processing and other activities being executed. A document centric view of a workflow management system that caters to other

activities facilitates this integration. In this paper, we address this problem by proposing an XML document centric workflow management system (XDoC-WFMS), which exploits the organization of the XML documents while having the full functionality of workflow management system to execute other activities. Notions of views, templates, document read units, smallest concurrency unit of a document are used along with micro-agents to provide support for document handling in a user transparent manner.

1.1 Contributions and Organization of the paper

The main contributions of the XML Document Centric Workflow Management System:

- Document Centric Work flow modeling, which allows for distinct document flows to be embedded within workflows.
- View based document interfacing, which allows the user to access only those (sub-)documents (s)he is restricted to view. This allows for the implementation of least privilege security mechanisms.
- Micro-agents are used to handle document transfers and entries into the database. Micro-agents are also used to handle some of the exception conditions implicitly within the XDoC-WFMS.
- A workflow management system that facilitates execution of document-processing activities and other activities in a seamless manner. A framework for full functionality of a workflow management system based on capability modeling and event-driven execution is presented.

The rest of the paper is organized as follows, Section 2 describes the architectural model of XDoC-WFMS, Section 3 describes concepts and terminology and the specification, Section 4 elaborates an Example, Section 5 describes the Execution Model of XDoC-WFMS, Section 6 presents the related work, and Section 7 the conclusion.

2. Concepts, Terminology and Specification

A *Workflow* aims to automate business processes, where documents and information are passed between agents according to a set of rules to achieve or contribute to an overall business goal. A *Workflow Management System* is the software for specifying, executing and monitoring workflows. Modeling a business process would involve describing *users* who interact with the workflow management system to perform *activities*, *documents* that the users modify and share, and the internal structure of the workflow. The concepts used, and introduced in XDoC-WFMS are described below.

Template : A template abstracts logically related data of the workflow. The execution of an instance of the workflow is equivalent to populating the template instances of the workflow. A template is an XML document while the read units and modifiable units within a template are XML sub-documents.

Templates are specified as classes in C++. Table 1 shows the C++ class specification of a template. Template objects are created for each instance of workflow execution.

Every template has an *identifier* and a *description*. It also contains a recursive structure representing the elements in the template, the element structure starts with the root node of the document and its child elements are all the child nodes of the root. The child elements are specified in the Element Structure by giving the root of the child element.

Table 1. Template, View and Agent Class Specifications

<pre> Class template { struct Child_Element { Element_Name; Attribute[]; Child_Element[]; } Template_Id; Template_Description; DTD { DTD rules } Child_Element root; } </pre>	<pre> Class view { viewid; viewname; agentrolename; Struct Read_Element { name; SCUid; elementdata; } Element_Structure //ES { SCU_s[]; Read_Element Read_Elements[]; } } </pre>	<pre> Class agent { agentrole; agentid; name; capabilities[]; Activities[]; } </pre>
---	--	--

Table 2. SCU Class and Object Specification

<pre> class SCU { SCU_Id, SCU_Name, Element_Structure//ES { Element_Name Template_Id ES Child[]; } } </pre>	<table border="1"> <tr> <td>SCUid</td> <td>1</td> </tr> <tr> <td>SCUname</td> <td>StudentInfo</td> </tr> <tr> <td>Element_Structure</td> <td></td> </tr> <tr> <td>ElementName</td> <td>StudentInfo</td> </tr> <tr> <td>TemplateId</td> <td>1</td> </tr> <tr> <td>Element_Structure Child[1]</td> <td></td> </tr> <tr> <td> ElementName</td> <td>Address</td> </tr> <tr> <td> Element_Structure Child[1]</td> <td>NULL</td> </tr> <tr> <td>Element_Structure Child[2]</td> <td></td> </tr> <tr> <td> ElementName</td> <td>Address</td> </tr> <tr> <td> Element_Structure Child[1]</td> <td>NULL</td> </tr> <tr> <td>Element_Structure Child[3]</td> <td></td> </tr> <tr> <td> ElementName</td> <td>Qualification</td> </tr> <tr> <td> Element_Structure Child[1]</td> <td>NULL</td> </tr> </table>	SCUid	1	SCUname	StudentInfo	Element_Structure		ElementName	StudentInfo	TemplateId	1	Element_Structure Child[1]		ElementName	Address	Element_Structure Child[1]	NULL	Element_Structure Child[2]		ElementName	Address	Element_Structure Child[1]	NULL	Element_Structure Child[3]		ElementName	Qualification	Element_Structure Child[1]	NULL
	SCUid	1																											
	SCUname	StudentInfo																											
	Element_Structure																												
	ElementName	StudentInfo																											
	TemplateId	1																											
	Element_Structure Child[1]																												
	ElementName	Address																											
	Element_Structure Child[1]	NULL																											
	Element_Structure Child[2]																												
	ElementName	Address																											
	Element_Structure Child[1]	NULL																											
Element_Structure Child[3]																													
ElementName	Qualification																												
Element_Structure Child[1]	NULL																												

Read Unit: Read Units are non-modifiable information units. These units are defined across elements of one or more templates, and reflect the data stored in the elements, in instances of the templates at time of user access. These units may differ from one

user to another. The Read Unit is specified as class, containing *element_ids* and corresponding *template_ids*.

Smallest Concurrency Unit (SCU): An SCU is the atomic updatable unit of a document. Simultaneous update access by multiple users/agents to an SCU is not permitted. However, simultaneous update of template data by different users/agents accessing different SCUs is possible therefore facilitating parallel execution of workflow activities and increasing the throughput. The data updated in the SCU is stored in the template instance. Each SCU contains a set of modifiable elements. An SCU can either be defined over a single template or across different templates. The class specification of an SCU is given in Table 2. In this specification, the element structure contains the element name, which is the name of the root element, and a field for template id. This field is filled only if the whole root node is taken from one template, else the field is set to null. The children are again nodes of type Element Structure, and they can be taken from some template, or specified by the WFMS-administrator. The SCU object, shown in Table 2, is instantiated at the time of View Creation.

View : A View is a collection of SCUs and RUs seen by an External Agent/user (EA) (described later). A view is generated dynamically from its base templates during workflow execution. For each of the Agent Roles in the XDoC-WFMS, a View is defined. Views can also be defined for individual EAs, but by default the view of the EA is the view of its Role. The EAs view the workflow system through these Views. Each EA is permitted to access to only a part of the system. Therefore, each EA is provided with a View that consists of elements which (s)he needs for performing an activity and has permission to. A view can contain one or more SCUs. In addition to the elements of SCUs, a view can also have other elements that are used only for information/display purposes. These constitute the read units of the view.

Table 1 gives the specification that helps the WFMS-administrator to describe a view. In this specification, *viewid* uniquely identifies the view, *viewname* is any name given to the view by the WFMS-administrator, *agentrolename* specifies to which agentrole this particular one serves as a view. The *SCUs* array specifies the different SCUs that belong to this view. *Read_Elements* array specifies the details of RUs in the View.

Workflow: A Workflow describes the business process. It represents the final business goal that has to be attained. It has a start state, an end state and some error and exception states. The start state represents the beginning of the workflow, the end state represents the attainment of the goal, and error states indicate exceptions.

The workflow decomposer, shown in Fig 1, decomposes the workflow specified by the WFMS-administrator into activities.

Sub-workflow: Sub-workflows represent sub-goals of the workflow. Their interaction is modeled using ECA rules [2,3]. A sub-workflow can be further subdivided into activities. An Activity is the atomic unit of work, and the interaction among sub-workflows is represented as follows. The concept of sub-workflows only helps in facilitating the specification of the workflow, while the execution is modeled at the activity level.

1. An activity is a (sub)-workflow S_i .
2. A conjunction of sub-workflows is a workflow. $W = S_1 \wedge S_2 \wedge \dots \wedge S_n$, where, W represents the Workflow and S_1, S_2, \dots, S_n the sub-workflows. Here the workflow is said to be executed only if all the sub-workflows are executed.

3. A disjunction of its sub-workflows is a workflow. $W = S_1 \vee S_2 \vee \dots \vee S_n$ the workflow W is executed only if one or more of S_i 's execute successfully.
4. A sequence of sub-workflows is a workflow. $W = \text{Seq}(S_1, S_2, \dots, S_n)$ the workflow W is executed only if sub-workflows execute successfully in the following sequence S_1, S_2, \dots, S_n .
5. A finite number of applications of above set of rules on a set of sub-workflows is also a workflow. For example, $W = \text{Seq}(S_1, (S_2 \wedge S_3), (S_4 \vee S_5))$

The workflow administrator decomposes workflows into sub-workflows, and sub-workflows into activities, using the decomposition algorithm [1].

Activity: An Activity is an atomic unit of work and is performed by a single agent.

An activity involves:

1. Interacting with Human Users through Views
2. Interacting with External software or hardware systems.

External Agent: An External Agent (EA) is a human, hardware or a software agent that is the recipient of some information in the workflow. The External agent interacts with the workflow by performing agent-specific functionality that contributes to the goal of the workflow. The functionalities of the EA are domain specific and are specified by the administrator. An EA's interaction with the workflow is modeled in the form of an activity.

An agent is capable of doing certain activities in the workflow. This agent is then said to possess *capabilities* [1] for performing the above activities. Therefore, EA's capabilities are used in assigning activities to it. The different EAs in the system and their characteristics are specified by using C++ classes. An EA is capable of doing many activities and an activity can be done by several EAs. Table 1 gives the class specification for an External Agent.

Agent Role: An Agent Role is an abstraction that is used to specify the basic set of permissions and capabilities for a set of similarly capable Agents.

Micro Agents: Micro Agents are internal agents of the XDoC-WFMS, which perform system specific tasks of the workflow management system. The micro-agents are semi-autonomous, reactive, pro-active entities having social ability [13].

A workflow execution involves process specific and process independent activities. Process specific activities depend on the business process and the method used for modeling it. Process independent activities, are specific to the workflow management system and define its working. Some examples of the process independent activities are data-transfer, event handling, database management, view management, exception handling and interfacing with external devices. Micro-agents perform the process independent activities of the WFMS. Some micro-agent functionalities are:

- Presenting the current data corresponding to the element from the database when the external agent initially accesses the view.
- Micro-agents support event handling at activity level.
- A micro-agent can access multiple databases to retrieve and input relevant information to the XML view being processed. For example, in the Post Graduate Admission Workflow, after the application has passed the completeness check, depending on the stream chosen by the student, a micro-agent could choose the faculty member that is capable of reviewing that particular student.

- Exception handling through user notification, user input, and automatic processing.

The properties of the micro-agents specified above support finer control of activity execution and speedier action through faster execution of activities of a workflow in XDoC-WFMS.

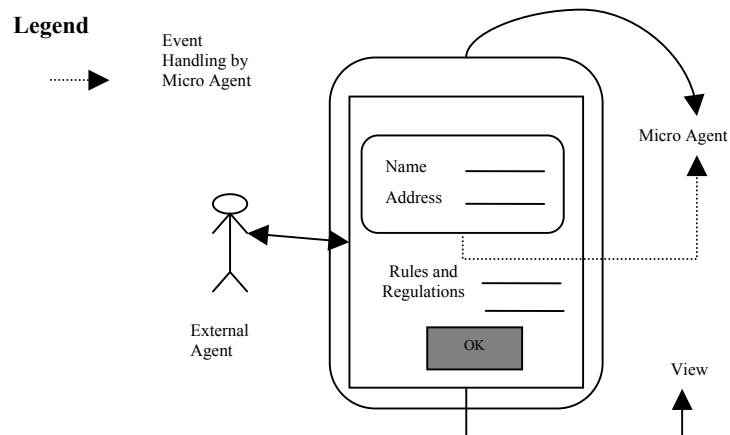


Fig. 1. Illustration of role of micro-agents in ExternalAgent-View Interaction.

Fig 2 depicts the interaction EA interaction with the view and the role of the micro-agent in it. A view is encapsulated in a micro-agent. The micro-agent handles the events generated while the view is being accessed. For example, if the RUs are updated by some other EA when this EA is viewing, the update will be notified to the current EA by the micro-agent of this view.

Administrator: The administrator of the workflow is a special External Agent, with pre-defined capabilities. The administrator plays two important roles

1. The administrator converts the business model, into a process to be executed by the WFMS, by specifying various workflow elements. (S)he specifies the Templates, SCUs, RUs, Views, Sub Workflows, Activities, ECA Rules and Agent Capabilities for a particular workflow.
2. The consistency of the above information is checked by the WFMS. The administrator monitors workflow execution to improve system efficiency and to take action in case of failures. Administrator deals with configuration management of the WFMS and failure recovery, which are outside the scope of this paper.

3. Architecture of XDoC-WFMS

The different modules of XDoC-WFMS system include Document Specification Language Processor, Workflow Decomposer, View Specification Processor, Workflow Generator, Micro-Agent Manager, Workflow Coordinator, Activity Log

Manager and Recovery Manager. Fig.1 illustrates the architecture of the XDoC-WFMS. The key modules of the architecture and their roles are explained below:

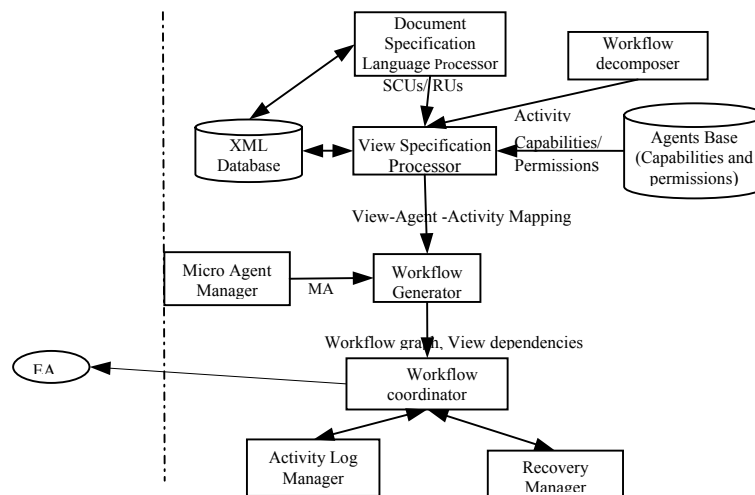


Fig. 2. Architecture of X-DoC WFMS

1. Document Specification Language Processor: This module allows the WFMS-administrator to specify the templates, SCUs and RUs of the workflow. The elements of the Template are stored in the XML database.
2. Workflow Decomposer: The workflow decomposer interacts with the administrator to decompose the workflow into sub-workflows and activities. This is done using the methodology of CapBasED-AMS [1] of decomposing workflows.
3. View Specification Processor: This module maps the views with their corresponding SCUs and RUs. This module also associates the Activities with the views, and the External Agents with the Activities that they can perform. It outputs a View-Agent-Activity Mapping. In case of Activities without corresponding views (i.e., Activities having control functionalities), the View Specification Processor outputs only the Agent-Activity Mapping.
4. Workflow Generator: The workflow generator takes a set of View-Agent-Activity and/or Agent-Activity Mappings and outputs a workflow graph. It interacts with the Micro-Agent Generator, which provides the micro-agents to handle view related activities and exception handling. This module also determines the order in which the views are accessed and therefore outputs the workflow graph. The graph represents the order in which the views are activated.
5. Micro-Agent Manager: With the workflow specification as the input, this module generates different micro-agents required by the workflow and associates them with the views. Apart from generating the micro-agents, this module is responsible for the various functionalities provided by the micro-agents. The different functions provided by the micro-agents are dealt with in detail in section 4.4.
6. Workflow coordinator: The workflow coordinator takes the input as the workflow graph and executes the workflow at run-time. The objects for templates, SCUs, views, etc. are handled by the Run-time object environment.

7. Activity Log Manager/ Recovery Manager In case of run-time exceptions, control is handed over to the activity log manager and recovery manager. The micro-agents interact with them to restore the system to a consistent state.

4. Example

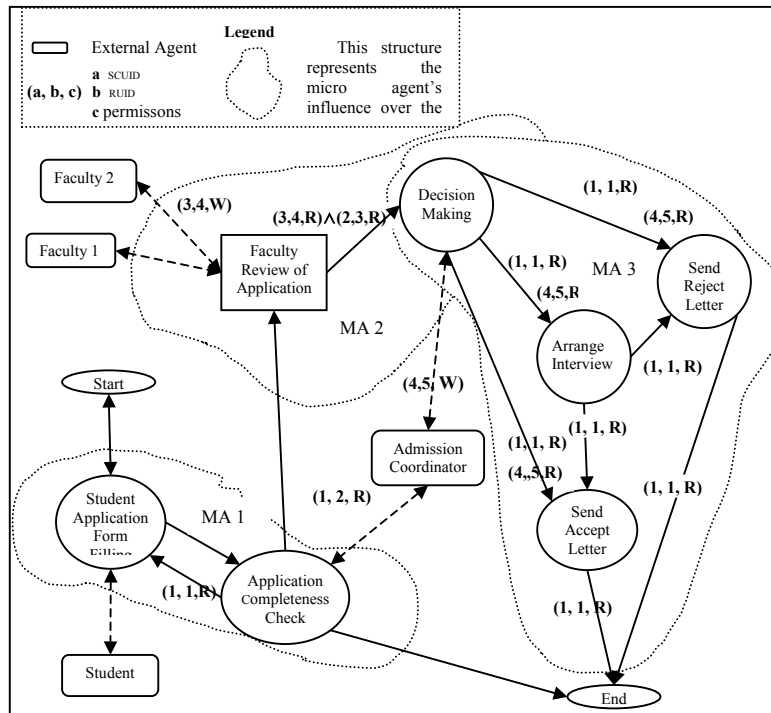


Fig. 3. Post Graduate Admission Workflow showing all the activities involved, and the agents which perform these activities.

Table 3. SCUs and RUs involved in Post Graduate Admission Workflow

<i>Student Template SCUs</i>			<i>Units involved in document flow</i>			
SCU ID	Elements in the SCU	EAs with write access	SCU name	SCU id	Read Units of this SCU	RUID
1	Name, phone, address, qualification, recommendation letters.	Student	Student	1	Name, address, phone	1
2	Comments & decision of Faculty1	Faculty1	Student	1	Name, qualifications, recommendation letters.	2
3	Comments & decision of Faculty2	Faculty2	Faculty1	2	Name, address, phone, qualifications	3
4	Coordinator decision	Admission Coordinator	Faculty2	3	Comments & decision	4
			Coordinator Decision	4	Coordinator Decision	6

The working example that is used for XDoC-WFMS, is the Graduate Admission Process in [1]. The graduate admission process starts with a student filling up an admission form. This admission form is sent to the Graduate Admission Coordinator who performs the completeness check. If the document is complete, it is sent to the faculty members for evaluation. Each faculty member puts forth his comments and decisions regarding the candidate. The Graduate Admission Coordinator then decides to accept, reject or interview the candidate. The student is notified of the decision. A candidate who is called for the interview is rejected or accepted based on the interview. There are time constraints in the system. If the student admission form is incomplete, he/she has to fill it and resubmit before the last date for application forms. Similarly the interview call notification has to be acknowledged before the last date. Also the faculty members who are evaluating the applications must do so before a particular date. In case of default, the student's application is rejected and his application is removed. The activity diagram of the above example is shown in Fig. 3, along with the SCUs, RUs, Micro-agents, and Views involved in it.

There are two different templates in the Graduate Admission workflow: Student Template (has both SCUs and RUs) and Notification Template (has only RUs). An additional read-only template could be a Course template whose elements represent information about the list of courses offered, and the course details are read units in the student template. Any element of the course template is not modified in the process of student admission. Whenever a student applies for admission, an instance of the student template is created. This template contains all the information about the student admission at any point in the workflow execution. The different SCUs involved in this workflow, the various elements each SCU contains and the EAs who are given write access to these SCUs are shown in Table 3. Also the different RUs involved in the workflow are shown in Table 3. (i,j,p) in the Fig 3 represents a read unit with id j , which is a part of an SCU with id i and is given a permission p . When a student fills up an instance of the student template, each of the SCUs in this template follows a different path along the workflow execution. After the student fills in the SCU with id 1 (Name, phone, address, qualification, recommendation letters), it is sent to the Admission Coordinator as read unit 3 in Table 3. The micro-agent associated with the CompletenessCheck activity of Admission Coordinator (MA 1 in Fig 3) does this data transfer. The Admission Coordinator who has a sound knowledge of the admission procedure (KAP[1]) and also has the ability to approve student admission (CTA[1]) checks if all the details are furnished by the applicant. If a student application is complete, the Coordinator, who has knowledge of the faculty research areas (KFA[1]) sends the student details: Name, qualifications and recommendation letters (RU id 2 of Table 3) to two faculty members through a micro-agent. After the deadline of the reviews, the Coordinator views the faculty comments and decision about the student under consideration, (SCUs 2 and 3 in Table 3) and gives his final decision of approving or rejecting the student by modifying the *Coordinator Decision field* (SCU 4 in Table 3). The Admission Coordinator can call a student for an interview to decide on selecting or rejecting the student. Once the decision is taken, the student information (RU id 1) and the Coordinator decision, accept/reject (RU id 6) need to be sent to the student. These Read Units are passed as parameters, to the micro-agent associated with the notification activity. The notification micro-agent treats these as read units and instantiates a notification

template with the values from the read units 1 and 6. The notification micro-agent interfaces with the e-mail client, is responsible for notifying the students appropriately. Even in case of an interview, the interview notification is similarly sent to the student and after the interview is conducted, the final decision of accept/reject is taken and the student is appropriately notified. If the decision is to admit the student, the student's information is entered into the table of admitted students; else, the student record is removed.

5 Execution Model

5.1 Architecture of Workflow Coordinator

The execution model describes how a workflow instance gets executed, and describes the steps in the workflow instance life cycle. The workflow coordinator handles the workflow execution. The architecture of workflow coordinator is given in Fig. 4. The components of the Workflow Coordinator are:

- **Workflow Engine:** The workflow engine is the key component of the workflow coordinator. Its functionalities are
 - It incorporates an Event Capture Module.
 - It invokes micro agents.
 - It performs ECA Rule look up and initiates the corresponding action.
- **Micro Agent:** A micro agent performs the following kinds of functions
 - It interfaces with external software or hardware systems.
 - It creates and handles views.
 - It manages user worklists.

The micro agent lifecycle is from creation of the micro agent at start of an activity execution to the completion of the activity.

- **The ECA Rule Base and Run Time Knowledge Base:** The ECA Rule base contains the ECA rules specific to a workflow. The Run Time Knowledge Base maintains instances of various workflow elements like Templates, SCUs, Views, Activities etc. associated with the different workflow instances executing at that point of time.

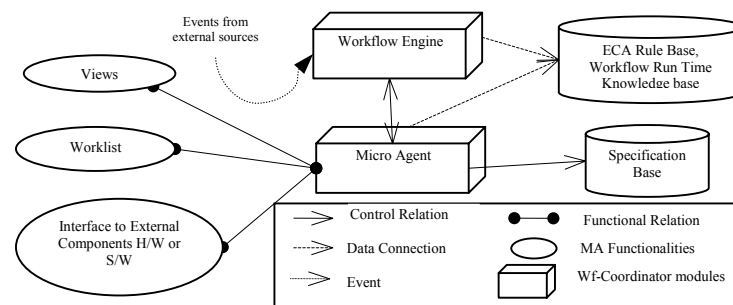


Fig. 4. Architecture of the Workflow Coordinator Module

5.2 The Workflow Execution Algorithm

Table 4. Pseudo Code for Execution Algorithm.

<p>Pseudo Code for the Execution Algorithm</p> <ol style="list-style-type: none"> 1. EventSet Ei 2. Capture_Start_Event(Ei) 3. Assign_Execution_Instance_Id(instance_id) 4. Action Ai = LookUpInECARule(Ei) 5. OutputEventSet Oi = PerformAction(Ai, instance_id) 6. Oi supersetof InputEvenSetForActivity() 7. μ = GenerateMicroAgentForActivity(instance_id) 8. Ei = PerformActivity(μ); 9. Goto Step iv
<p>Pseudo Code of Perform Activity</p> <ol style="list-style-type: none"> 1. GetActivityDescription 2. if InterfaceRelatedActivity 3. ExecuteLibraryRoutine 4. if UserRelatedActivity 5. CreateUserView 6. if WorklistRelatedActivity 7. UpdateUserWorklist

Example: The execution of the example given in Section 4 can be modeled in terms of the Execution Algorithm.

1. The student logs on to the WFMS. This generates the workflow start event.
2. The workflow engine creates a workflow instance and assigns it an id.
3. The workflow engine invokes and assigns a micro agent to the ‘Student Form Filling’ activity. This micro agent handles the Student Form Filling view events. After the successful completion of the activity, the micro agent updates the student template and notifies the workflow engine.
4. The workflow engine instantiates a micro agent for ‘Admission Completeness Check Activity’.
5. This micro agent handles the events associated with the admission coordinator view. The completion of this activity generates the incomplete_application or complete_application event.
6. The wf engine performs an ECA Rule lookup for the above events
 - The incomplete_application event is the input event to the ‘Incomplete Application Notification’ Activity. This activity involves interfacing with the e-mail client and sending the notification message as e-mail to the student. The interfacing is facilitated by library routines corresponding to that particular e-mail client. The student template information is retained till the timeout on receiving application occurs.
 - The complete_application event is the input event for faculty review activity. Each of the two faculty members evaluating the student application performs an instance of this activity. Therefore, a micro agent is invoked for each faculty member involved in the activity. These agents are invoked simultaneously.

After a faculty member finishes reviewing, the micro agent updates the student template and notifies the workflow engine about the activity completion.

7. The workflow engine invokes the next activity - 'Admission Coordinator Decision' only after both the faculty reviews are available. It creates a micro agent associated with this activity, which in turn handles the events associated with the Decision view. The output event of this activity is 'Accepted', 'Rejected' or 'Interview'.
8. The workflow engine performs an ECA Rule lookup for the above events. All the three events involve sending notifications. The micro-agent interfaces with the e-mail client as described in step 6. The corresponding action for the 'Reject' event would also involve removing the student template from the knowledge base.

5.3 Discussion

XDoC-WFMS, was an attempt, to develop a document centric workflow management system that makes use of the inherent structural semantics of XML, and the flexibility of the agents. The concepts of SCUs, Templates, Views and micro-agents have been introduced. We have used XML as our document description language, because XML facilitates the division of documents into semantically well-structured sub documents. Templates and Views, are comparable to base tables and relational views in the relational model. SCUs were introduced to define sub-documents that facilitate concurrent access to the templates. Views were introduced to provide secure access to the system. Apart from this, they also allow for customization of user preferences. Workflows have embedded document flows in them, but this usually deals with document level movement, or message passing. But XML also facilitates sub-document flows and a refresh capability to view the current state of the database. The execution of a WFMS requires processing entities that maintain the connectivity of activities in workflow. These entities form an inherent part of the system, while ensuring consistency, and efficiency in execution. Menial WFMS tasks can be entrusted to micro agents as user performed activities tend to be slow. To increase the speed of execution of these activities, micro agents can be used to perform some amount of pre-processing and post-processing pertaining to an activity. And micro agents can be customized by changing their execution parameters. Further details about role of micro agents in XDoC-WFMS are currently being worked on.

6. Related Work

The CapBasED-AMS [1] has an architecture built around task management and user interaction with tasks. The tasks there are general and also include system specific tasks. System specific tasks are modeled as interactions with hardware or software systems and are specified by the administrator. The micro-agents in XDoC-WFMS handle this system specific activities and interface with the Views provided to the External Agents, who are the users, either human, software or hardware interacting with the system. OASIS system [5] models their WFMS around MOAPs (Micro Organizational Activity Processor). The OASIS WFMS is a network of communicating MOAPs, which are associated with knowledge bases at various levels – user, group and organizational. A task in OASIS, encapsulates both data and

processing objects. XDoC-WFMS totally separates data and processing units. The processing units are micro-agents, and data is specified in terms of Templates. This allows the administrator to specify document flows separately, without having to specify micro-agents. In [4] on secure workflows, the concept of Least Privilege Security is dealt with. Least Privilege Security gives a user minimum access to tasks and information in the organization; the privileges are withdrawn from the user once the task is completed. This is encapsulated in XDoC-WFMS by the inclusion of Views. The access to these views is restricted. Only an External Agent who has the right privileges can access the view and his permissions are withdrawn once the activity is performed.

The MARIFlow system [6] describes the architectural details required for automating and monitoring the flow of control and documents over the Internet among different organizations. This is achieved by using MARCAs (or MARIFlow Cooperating Agents). The MARCAs are automatically initialized at each of the sites involved in the workflow and they provide for coordination with the other agents in the system by routing the documents in electronic form according to the process description. The security constraints like organizational firewall are also handled by these MARCAs. The notion of orthogonal and yet related handling of document flows, activity control flows and WFMS functionality in a seamless integrated manner by using templates, views and micro-agents is novel and contributing aspect of this work.

Event handling during a workflow execution can be made easy by integrating event handling system into the database. This can be done by using Database Alerting Techniques [8]. This work deals with an office automation model using active databases. The updates made to the office database can trigger events based on user defined rules. For example, a rule can be specified to make an Alerter Module, monitor a particular type of an Update Operation. The Activity Management System in this model consists of an Alerting subsystem, Office manager and Activity agents. The alerting subsystem screens database updates to detect if any of the rules have satisfied so that events can be triggered. The office manager coordinates activities and the activity agent is an office specialist capable of performing some defined office activity. The user interacts through a user interface called *Intelligent Coupler* which sends appropriate messages to various modules of the AMS. In our work, we introduce the concept of micro-agents which serve this purpose.

The system OA-DBMS (Office Automation DBMS) [9] emphasizes on an integrated database support, such that data can be shared among applications. It supports the use of four data modeling concepts: aggregate hierarchies (a document is a set of hierarchy related entities which are aggregations of subtypes), generalization (abstraction of common properties of different entities), cross references (expresses subordinate relationships of entities of similar or different types) and long fields (Unstructured and multimedia data are stored as long fields).

METUFlow system[10] supports the idea of distributed execution environment. In this distributed environment, the scheduler, the history manager and worklist manager are fully distributed, giving rise to failure resiliency and increased performance. The workflow process here is defined as a collection of blocks, tasks and sub-processes and each of them is referred to as an activity.

The WASA Approach [11] deals with a flexible and platform-independent workflow support for scientific applications such as geo-processing, molecular biology and

laboratory environments. In these domains, there is a need for dynamic modifications of the workflow model, while the workflow is being executed. These changes are categorized as anticipated dynamic changes and ad-hoc dynamic changes. The WASA model explains how the two types of changes are handled.

Enterprise-Wide Workflow Management Based on State and Activity Charts [12] deals with the specification, verification and distributed execution of Workflows based on State and Activity Charts. Activity Charts specify the dataflow between activities in the form of a directed graph, whereas State Charts specify the control flow between the activities and the control flow is driven by the ECA Rules. State and Activity Charts help in automatic verification of critical workflow properties by methods of model checking. Distributed execution is achieved through partitioning of the centralized workflow specification, automatically, using these graphs.

7. Conclusion

Document flow is an important aspect of workflow driven execution of business processes. But document flow and processing is not the complete workflow. Therefore, there needs to be a document centric, and yet full workflow functionality to support execution of workflows for business processes. Further, within documents there are interdependencies among sub-documents, sharing and concurrent updates to documents, and restricted access to documents. In order to facilitate the combined workflow and document flow oriented business processing, an XDoC-WFMS has been proposed in this paper.

The salient features of this system are:

- Document Centric Work flow modeling, which allows for distinct document flows to be embedded within workflows.
- View based document interfacing, which allows the user to access only those documents (s)he is restricted to view.
- Micro Agents to handle document transfers and entries into the database and also to handle exception conditions that arise during the workflow execution.
- A workflow management system that facilitates execution of document-processing activities and other activities in a seamless manner. A full functionality of a workflow management system based on capability modeling and event-driven execution is provided for this.

In our on-going work, we are implementing the XDoC-WFMS and are conducting case-studies to automate the administrative business processes in the institute.

References

- [1] K. Karlapalem, H. P. Yeung, P. C.K.Hung: "CapBasED-AMS- A Framework for Capability-Based and Event-Driven Activity Management System", CoopIS 1995: pages 205-219.
- [2] S. Chakravarthy, V. Krishnaprasad, Z. Tamizuddin, R. H. Badani: ECA Rule Integration into an OODBMS: Architecture and Implementation, International Conference on Data Engineering 1995: 341-348

- [3] S. Chakravarthy, R. Lee, R. Dasari: "ECA-Rule Processing in Distributed and Heterogeneous Environments", *Distributed Objects and Applications* 1999: 330-339.
- [4] P. C. K. Hung : "Secure Workflow Management Systems", Ph.D. thesis, Department of Computer Science, The Hong Kong University of Science and Technology, July 2001 .
- [5] C. Martens, F. H. Lochovsky: "OASIS: A Programming Environment for Implementing Distributed Organizational Support Systems" , Conference proceedings on Organizational computing systems, 1991, Pages 29 - 42
- [6] A. Dogac, Y. Tumbag, A. Tumer, M. Ezbiderli, Nesime Tatbul, N. Hamali, C. Icdem, Catriel Beerli: "A Workflow System through Cooperating Agents for Control and Document Flow over the Internet", *CoopIS 2000* : 138-143.
- [7] K. Karlapalem, A.R. Dani, P. Radhakrishna: "A Frame Work for Modeling Electronic Contracts", to appear in ER-2001, Japan, 2001.
- [8] Jo-Mei Chang and Shi-Kuo Chang: "Database Alerting Techniques for Office Activities Management", *IEEE Transactions on Communications*, Vol. Com-30, No.1, Jan 1982 : Pages 74-81
- [9] M. Bever and D. Ruland: "Aggregation and generalization hierarchies in office automation", Conference Sponsored by ACM SIGOIS and IEEECS TC-OA on Office information systems, 1988 : Pages 250 - 264
- [10] A. Dogac, E. Gokkoca , S. Arpinar, P. Koksai, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, M. Altinel : "Design and Implementation of a Distributed Workflow Management System: METUFlow" , NATO ASI Series, Computer and System Sciences, Vol.164: Pages 61-91.
- [11]G. Vossen, M. Weske: "The WASA Approach to Workflow Management for Scientific Applications", NATO ASI Series, Computer and System Sciences, Vol.164: Pages 145-164.
- [12] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz-Dittrich: "Enterprisewide Workflow Management Based on State and Activity Charts", NATO ASI Series, Computer and System Sciences, Vol.164: Pages: 281-303.
- [13] G. Joeris, C. Klauck; O. Herzog: "Dynamical and Distributed Process Management based on Agent Technology", in Proc. of the 6 th Scandinavian Conference on Artificial Intelligence (SCAI'97), August '97, Helsinki, Finland, 1997: Pages 187-198.