Implementing Distributed Workflow Systems from XML Specifications *

Anand R. Tripathi, Tanvir Ahmed, Vineet Kakani and Shremattie Jaman
{tripathi, tahmed, kakani, jaman}@cs.umn.edu
Department of Computer Science
University of Minnesota, Minneapolis MN 55455

ABSTRACT

In enterprise-wide distributed workflow, multiple users cooperate towards some common goal following predefined task specifications. This paper presents an approach for constructing a distributed workflow system starting from its XML specification in terms of various participants' roles, access rights based on roles, shared objects, operations, and workflow constraints. The specifications are integrated with an agent-based distributed middleware for workflow implementation. This middleware enforces coordination and security constraints specified in the XML description of the plan. Moreover, shared objects in the workflow environment are distributed to the participants according to the role-specific views of the plan. We illustrate our approach through an example workflow system for collaborative authoring.

1 Introduction

A workflow management system can be considered as a computer automated infrastructure where a group of people participate together to achieve a common goal following some predefined rules and task assignments. Most of the existing workflow management systems try to devise a generic workflow model supported by some middleware [14, 2]. Moreover, these systems provide some specification languages and workflow design tools to support workflow activities ranging from state manipulation to automatic recovery [4].

The main contribution of this paper is in developing a methodology for building a distributed workflow environment by starting with its specification in a high level form in XML [22], and then interfacing such a specification with a mobile agent based middleware to build the desired workflow environment. The XML specification needs to describe shared objects, coordination operations, roles of various people involved in a workflow, security policies based on roles, and workflow constraints. The middleware enforces the coordination and security constraints specified in the XML specification of a plan. An important aspect of our work is the fact that the designer of a workflow needs to specify only the workflow plan without being concerned about the management of the mobile coordination agents and the security issues.

There are several motivations for implementing distributed workflow systems using a mobile agent based middleware. Mobile agents can be used to encapsulate application specific coordination protocols. Through their use in a workflow, a group coordinator can ensure that the members participate only by using the prescribed protocols. Moreover, with the use of mobile agents as user-interface objects, the administrator or coordinator of an environment can dynamically upgrade its agents installed at the participant's nodes to alter the workflow policies. It can also grant and change appropriate privileges to a member based on his role in the collaboration. It is possible to exploit the mobility of a shared object, when implemented as a mobile agent, by moving it from one participant to another at various stages of a workflow. By implementing a user's interaction environment in a workflow as a collection of mobile agents, it is possible for a user to physically move to a different node by simply directing its agents to migrate to that node. Moreover,the mobile agent paradigm can be exploited to implement workflow systems in disconnected environments. This can be achieved as mobile agents can carry along all the application specific data and code, eliminating the need for a shared file system.

We present the details of this approach through a workflow authoring system, which we implemented as a proof-of- concept experiment, using the Ajanta mobile agent system [20]. The overview of our approach is described in Section 2. Section 3 presents a schema for specifying workflow environments and shows how the example authoring system can be specified as a workflow plan. Section 4 describes how this plan is used to build a distributed workflow system using the facilities of the Ajanta mobile agent programming system. Section 5 presents related work, and section 6 discusses the conclusions and future directions of our work in this area.

*This work was supported by National Science Foundation grants ANIR 9813703 and EIA 9818338.

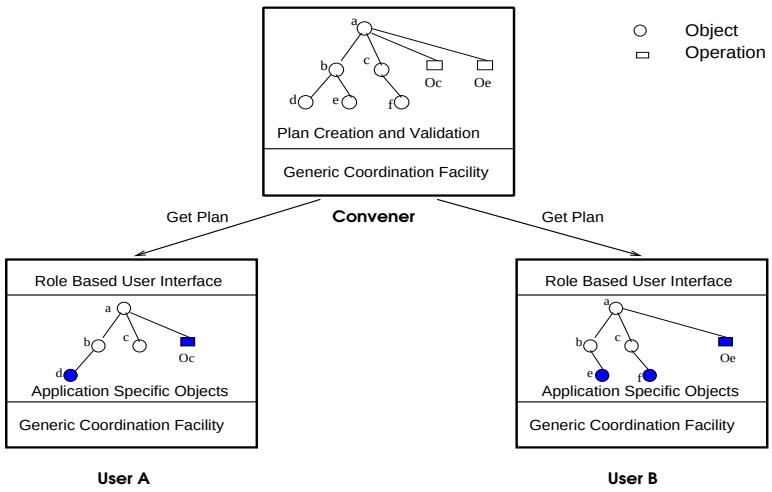


Figure 1: System level view of agent-based workflow

2 Overview of the Approach

In the proposed approach, building a distributed workflow involves three steps. The first step is devising a schema in the form of XML Document Type Definition (DTD) to specify a workflow environment. Such a schema provides constructs for defining roles, shared objects, and operations associated with a workflow task. It also provides constructs for associating privileges with roles and participants, coordination actions and workflow constraints with operations. The semantics of objects and their actual definitions are not described but left to the applications.

The second step is the description of a workflow plan, using XML, in conformance with the DTD. The designer of the workflow plan, whom we refer to as the *convener*, is responsible for preparing this description. This step involves defining the following elements in a workflow:

Shared Objects: A workflow involves management and use of a set of shared objects. These objects define the shared workspace of the participants. Object specific operations are performed by the participants based on their roles.

Participant Roles: A role defines a set of responsibilities and tasks for a participant towards the goal of the workflow activity. The participants and other entities involved in a workflow can be identified by their roles. These roles are assigned to people at the time of the realization of a plan.

Privileges: The security policies in a workflow are based on participants' roles. Privileges are associated with each role in order for the participant assigned to that role to execute the required tasks.

Coordination operations: The coordination operations define the actions to be executed when certain events occur. These are the user level operations which are executed whenever a stage of workflow is accomplished. An operation execution may result either in copying the corresponding object to other participants or in invocation of certain methods on objects in other participants' object-space.

The third step in our approach is illustrated by Figure 1, which shows the typical structure for realizing a distributed workflow using a generic coordination facility. The XML specification of the workflow is interfaced with a generic coordination facility built using the Ajanta agent execution environment. The workflow plan is securely distributed in a tamper-proof manner to each users' computer. This plan is trimmed on the basis of the user's role in the workflow lifecycle so that the user does not get parts of the plan which he will not access. We refer to this plan as the *role specific plan* for that user. Each user participates in the workflow by executing an agent-based coordinator on his system. We refer to this as the *User Coordination Interface (UCI)*. A UCI maintains with it copies of the shared objects that are required as per the user's roles. It provides suitable interfaces to its user to facilitate execution of operations on the shared objects.

When installed and initialized on a user's computer, a UCI downloads the role specific plan from the convener. This plan is composed of a collections of objects and their associated operations in the form of a XML tree. We refer to the local copies of the objects at the UCI as the user's *object-space*. Some of the objects in the shared workspace may not be present in the participant's object-space, if disallowed by the security policy. In Figure 1, object *d* is present only at at user A's object-space, and objects *e* and *f* are present only at user B's object-space. This figure is explained in more detail in section 4.

3 Specification of a Workflow

This section presents an XML schema that we have developed for specifying a workflow plan. First, we describe an

example workflow authoring system and then introduce our schema in DTD for generic workflow environments. Using the authoring system as an example, we illustrate our approach.

3.1 A Collaborative Authoring Workflow

We use a collaborative authoring workflow as an example to experiment with the central concepts discussed in this paper. The collaborative authoring environment considered in this paper supports the activities of a group of people jointly developing a document as shown in Figure 2. The Figure introduces the overall model of our authoring workflow example where the convener generates a plan in XML which encapsulates the central entities of the workflow like a shared document, roles in workflow and workflow constrains. In this example, the participants perform tasks related to writing, reviewing, and editing different parts of the document to be developed. Participants are assigned a variety of roles to perform these tasks.

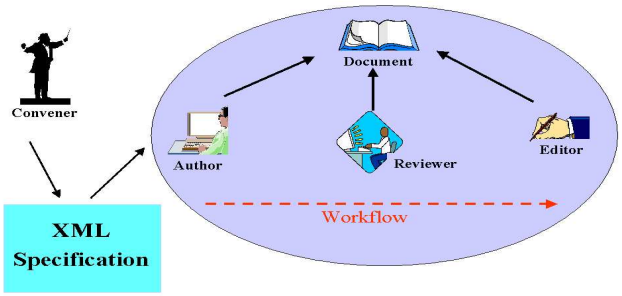


Figure 2: Steps in building an agent-based distributed workflow

The central entity in the shared workspace is the document object to be produced. Three roles are defined for the authoring system: *author*, *editor*, and *reviewer*. The document object contains one or more chapter objects. A chapter contains three objects that are shared: the chapter's content, the reviewer's comments and the editor's comments.

Following are some of the coordination operations defined for this authoring system. The first operation in the collaboration occurs at the chapter level. When the author of a chapter completes a draft of the chapter's content, it is published to the chapter's reviewer and editor by invoking the "publish" operation. This results in the coordination actions of making the chapter's content available to the participants in these two roles. This operation also enables a reviewer to write a review of the chapter's text. The reviewer of the chapter cannot compose the review until the chapter content has been written and received. The next operation occurs at the reviewer level. Upon completion of the review, the reviewer publishes the review to make it available to the author and the editor. The editor of the chapter composes his comments based on the chapter's content and the review. The editor publishes the comments to the author. The author can then read the review and the editor's comments, and modify the chapter accordingly.

3.2 Workflow Environment Description and an Example Plan

Our XML DTD defines a generic specification of workflow environments to be supported by an agent based middleware. Figure 3 defines a workflow environment under the tag PLAN, which is composed of one or many roles, objects, and operations. This definition of a workflow environment can be used to build any workflow plan; it is not limited to the authoring example presented here. Figures 3, 4, 5, 6, and 7 show, in five parts, the DTD specification for a generalized workflow environment. These figures include the corresponding examples of an XML plan for the document preparation system.

As shown in Figure 3, a role has a unique id, a name, an object reference, and a role-interface. The object reference is specified if a role is associated with an object and then the object defines the scope for the role. The role-interface can be a Java class supplied by the convener or implied by the workflow system. This role specific interface can provide different views of the same object to different participants. Each role is assigned to one or more participants, who are uniquely identified by Uniform Resource Names (URN) [19]. The Ajanta system uses the URN scheme for naming all global objects and users. In the example in Figure 3, the author role has a participant *urn:ans:A* and the reviewer role has two participants: *urn:ans:B* and *urn:ans:C*. The Java class of the author's interface is `ajanta.UserInterface`, which is specified by the convener.

Role based access control [17, 3, 5] provides centrally controlled access rights and is flexible to enforce policy guidelines for enterprise workflow systems. In our XML DTD, we mapped the Java ACL (Access Control List) interface and mapped its group interface to role. As shown in Figure 4, each ACL has a name, an owner if specified, and multiple entries containing access rights. Each entry is a mapping between a principal or a group and a set of permissions. Moreover, each entry type can be either positive or negative, and the default value is positive. In this specification, a group is a role reference. Access rights are defined in term of three generic operations: *execute*, *update*, and *read*. It is possible to include other types of permissions based on method level access control. In the example of Figure 4, the author role has *read* and *update* permissions, the reviewer role has *read* permission, but the principal *urn:ans:C* is explicitly denied *read* permission.

```
<!DOCTYPE PLAN [
<!ELEMENT PLAN (ROLE+, OBJECT+, OPERATION+)>

<!ELEMENT ROLE (PRINCIPAL+)>
<!ATTLIST ROLE
  ROLE_ID ID #REQUIRED
  ROLE_NAME CDATA #REQUIRED
  OBJECT_REF IDREF #REQUIRED
  ROLE_INTERFACE NMTOKEN #IMPLIED
>
<!ELEMENT PRINCIPAL EMPTY>
<!ATTLIST PRINCIPAL
  URN CDATA #REQUIRED
>
......
]>
```

```
<PLAN>
  <ROLE ROLE_ID="doc:author"
        ROLE_NAME="author"
        OBJECT_REF="doc"
        ROLE_INTERFACE="ajanta.UserInterface">
    <PRINCIPAL URN="URN:ans:A"/>
  </ROLE>
  <ROLE ROLE_ID="doc:reviewer"
        ROLE_NAME="reviewer"
        OBJECT_REF="doc"
        ROLE_INTERFACE="ajanta.UserInterface">
    <PRINCIPAL URN="URN:ans:B"/>
    <PRINCIPAL URN="URN:ans:C"/>
  </ROLE>

.........
</PLAN>
```

Figure 3: Schema and example for a Workflow Plan and Role

```
<!ELEMENT ACL (ACL_ENTRY*)>
<!ATTLIST ACL
  ACL_NAME CDATA #IMPLIED
  ACL_OWNER CDATA #IMPLIED
>
<!ELEMENT ACL_ENTRY ((PRINCI-
PAL|GROUP), PERMISSION*)>
<!ATTLIST ACL_ENTRY
  TYPE (positive|negative) "positive"
>
<!ELEMENT GROUP EMPTY>
<!ATTLIST GROUP
ROLE_REF IDREF #REQUIRED
>
<!ELEMENT PERMISSION EMPTY>
<!ATTLIST PERMISSION
  NAME (execute|update|read) "execute"
>
```

```
<ACL>
  <ACL_ENTRY>
    <GROUP ROLE_REF="doc:author"/>
    <PERMISSION NAME="read"/>
    <PERMISSION NAME="update"/>
  </ACL_ENTRY>
  <ACL_ENTRY>
    <GROUP ROLE_REF="doc:reviewer"/>
    <PERMISSION NAME="read"/>
  </ACL_ENTRY>
  <ACL_ENTRY TYPE="negative">
    <PRINCIPAL URN="URN:ans:C"/>
    <PERMISSION NAME="read"/>
  </ACL_ENTRY>
</ACL>
```

Figure 4: Schema and example for Role based Access Control

According to the DTD specification in Figure 5, an object must have a unique id, a name, and a type specified using MIME format. Moreover, an object can have a parent object reference, data, and a codebase which specifies the class implementing this object. Access control is imposed on each object. Object status, which is a list of operations performed on the object, is maintained with each object. An object can have methods and can be composed of other nested objects. The shared object in this example is a document with a single chapter. The review and editor comments of the chapter are omitted from the example as the given chapter specification is sufficient for the basic illustration of our approach. The type of the document and the chapter are declared using MIME type `multipart/alternative`, as the document may contain several chapter objects and each chapter is composed of contents, reviews, and editor's comments. The access control for a chapter's content is shown separately in Figure 4.

Figure 6 shows the specification of an operation whose execution is controlled by the UCI. Each operation has a name, access control entries, precondition, and either a clone object or one or more agent-actions. The access control entries of an operation specify for various roles the permissions to execute this operation. A precondition is a list of operations, which must precede the given operation. The precondition ensures operation constraints. The CLONE_OBJECT and AGENT_ACTION tags define the coordination actions that need to be performed when a given operation is executed. The clone operation is provided to encapsulate the generic operation of object migration in the workflow system. Figure 6 presents the plan for an operation specification. The author role has the access right to perform the `Content_Publish` operation on the `Contents` object. The corresponding clone object `Contents` of the operation `Content_Publish` contains a list of roles as targets.

In Figure 7, agent actions are the remote actions performed by an agent by visiting other users' UCIs. An agent-action is composed of a remote method and one or more targets which are references to roles. The remote method is uniquely identified by a reference to the desired object, a method name,

```
<!ELEMENT OBJECT (ACL, STATUS, METHOD*,
                  OBJECT*) >
<!ATTLIST OBJECT
  OBJ_ID ID #REQUIRED
  PARENT_OBJ_REF IDREF #IMPLIED
  OBJ_NAME CDATA #REQUIRED
  OBJ_TYPE CDATA #REQUIRED
  CODE_BASE NMTOKEN "null"
  OBJ_DATA CDATA "null"
>
<!ELEMENT METHOD ( ACL, PARAMETER*)>
<!ATTLIST METHOD
  METHOD_NAME NMTOKEN #REQUIRED
>
<!ELEMENT STATUS (OPERATION_PERFORMED*)>
<!ELEMENT OPERATION_PERFORMED EMPTY>
<!ATTLIST OPERATION_PERFORMED
  OP_REF IDREF #REQUIRED
  OPERATOR CDATA #IMPLIED
>
```

```
<OBJECT OBJ_ID="doc"
        OBJ_NAME="Document"
        OBJ_TYPE="multipart/alternative">
  <ACL/>
  <STATUS/>

  <OBJECT OBJ_ID="doc:ch1"
          OBJ_NAME="Chapter1"
          OBJ_TYPE="multipart/alternative">
    <ACL/>

    <STATUS/>
    <OBJECT OBJ_ID="doc:ch1:content"
            OBJ_NAME="Contents"
            OBJ_TYPE="text/plain">

    ....

</OBJECT>
```

Figure 5: Schema and example for Object

```
<!ELEMENT OPERATION (ACL, PRE_CONDITION*,
 (CLONE_OBJECT | AGENT_ACTION+))>
<!ATTLIST OPERATION
  OP_ID ID #REQUIRED
  OBJECT_REF IDREFS #REQUIRED
  OPERATION_NAME NMTOKEN #REQUIRED
>
<!ELEMENT PRE_CONDITION EMPTY>
<!ATTLIST PRE_CONDITION
  OP_REF IDREFS #REQUIRED
>
<!ELEMENT CLONE_OBJECT (TARGET*)>
<!ELEMENT TARGET EMPTY>
<!ATTLIST TARGET
  ROLE_REF IDREFS #REQUIRED
>
```

```
<OPERATION OP_ID ="doc:ch1:con:publish"
           OBJECT_REF="doc:ch1:content"
           OPERATION_NAME="Content_Publish">
  <ACL>
    <ACL_ENTRY>
      <GROUP ROLE_REF="doc:author"/>
      <PERMISSION/>
    </ACL_ENTRY>
  </ACL>
  <CLONE_OBJECT>
     <TARGET ROLE_REF="doc:reviewer"/>
  </CLONE_OBJECT>
</OPERATION>
```

Figure 6: Schema and example for Operation

and a parameter list. In general, these are the methods of the shared objects that an agent would invoke when visiting the UCIs of the users corresponding to the target roles. In our current implementation, the underlying agent based middleware supports operation constraints, and maintains and manipulates operation status in each object's node. In a similar plan for an authoring system, we separated this task on an application level object `status`. As an example of agent action, Figure 7 shows how the `status` object is manipulated by the `UpdateStatus` operation. When this operation is invoked, agents are launched to corresponding participants of target roles. These agents execute the `updateOp` method on the remote copies of the `status` object. Here, the remote method requires no parameters. For simplicity, the details of the parameter specifications are not included in the schema.

4 Implementation of Workflow using Mobile Agents

In this section we describe how the XML description of a workflow is integrated into a generic facility which supports coordination to achieve the desired workflow. The generic facility is built using the components of the Ajanta agent programming system because this framework is readily available and provides security and other necessary components. Section 4.1 presents a brief overview of the Ajanta's agent programming model and facilities. Using the authoring system as an example, section 4.2 describes how a plan is interfaced with the agent-based middleware to build UCIs.

4.1 Mobile Agents in Ajanta

Ajanta [20] is a Java-based framework for programming mobile agents in distributed systems. In Ajanta, mobile agents are *mobile objects*, which can migrate autonomously in distributed environments. Agents encapsulate code and execution context along with data and are executed on behalf of a user. The Ajanta system provides facilities to build customizable *servers* to host mobile agents, a set of primitives for the creation and management of agents, and a global naming service. Programming abstractions are provided for remote execution by agent migration. Security is an integral part of Ajanta design and Ajanta provides components for authentication, public key maintenance, access control, host resource protection, and cryptographic services. Interested

```
<!ELEMENT AGENT_ACTION (TARGET*, REMOTE_METHOD)>
<!ELEMENT REMOTE_METHOD (PARAMETER*)>
<!ATTLIST REMOTE_METHOD
  OBJECT_REF IDREFS #REQUIRED
  METHOD_NAME NMTOKEN #REQUIRED
>
```

```
<OPERATION OP_ID="doc:status"
           OBJECT_REF="status"
           OPERATION_NAME="UpdateStatus">
 <ACL/>
   <AGENT_ACTION>
     <TARGET ROLE_REF="doc:reviewer"/>
     <REMOTE_METHOD OBJECT_REF="status"
                    METHOD_NAME="updateOp"/>
   </AGENT_ACTION>
 ......
```

Figure 7: Schema and example for Agent Action

readers can refer to [20, 9] for further details.

4.2 Agent Based Implementation Environment

In this section, we present the implementation details of the example authoring system which is interfaced with the agent based middleware. We do this by identifying the various stages involved in the workflow:

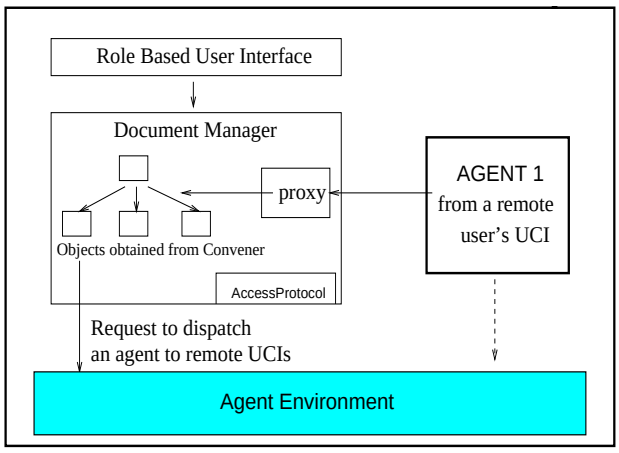
Plan Creation and Consistency Checking: The convener prepares a plan from the XML specifications which conforms with the DTD. While preparing the plan, it only needs to provide the values for the REQUIRED attributes of each entity. The IMPLIED attributes of different entities are determined by the UCI, if not specified as a part of the plan. The convener then parses and validates this XML plan against the DTD to generate a Document Object Model(DOM) [22] tree representation of the XML entities. The next step is to check the operation and access control consistency. For example, the plan is inconsistent when a user wants to clone an object to a target but the update permissions are not given to that user. Once a consistent plan is available, the convener waits for the plan requests from the entities participating in the workflow.

Plan Distribution with Authentication: On receiving a user request for a plan, authentication is performed using the Ajanta naming service. The naming service maintains the public key of each principal in Ajanta. Once the user is authenticated, a plan is downloaded to the user's UCI based on his roles in the workflow. As mentioned in section 2, this role specific plan is a trimmed version of the instantiated DOM tree in the convener's UCI. To construct a role specific plan, the concept of *view-node* access is used, which specifies that the user can see the attribute values for an object node but not its data. Based on the following rules, the convener constructs the role specific plan : if a user has any access on a method, he has *view-node* access on the method's object; if a user has any access on an operation, he has *view-node* access on the corresponding object; for each object with *view-node* access, a user has *view-node* access on all its parent object nodes. In Figure 1, user A has access on object *d* and operation *Oc*; here, the operation *Oc* is associated with object *c*. Hence, the user A has *view-node* access on objects *a*,*b*,*c* and *d*. This step ensures that the user sees only a view of the plan on which he has access privileges. While customizing the plan for a particular user, the convener bundles all the objects to be given to the user along with the related operations, as a part of the plan. Currently, the convener also distributes the role definitions as a part of the plan.

Creation of UCI and Role Based User Interface: Each user performs his tasks in the workflow through the UCI process executing on the user's desktop. Our approach is to implement a UCI by extending Ajanta's `AgentServer` class, which provides a secure execution environment for hosting mobile agents. Figure 8 shows the structure of a UCI for the example authoring system. It contains a *document manager* object and *user interface* objects. The document manager deals with the XML plan obtained from the convener and maintains the shared object space. It also maintains operation nodes corresponding to the OPERATION tag and these operation nodes contain various specifications needed for operation execution. Its interface enables the user and the visiting agents to read, write, edit, and publish any part of the document.

The user performs these tasks through the role specific user interfaces, which are either obtained from the convener and instantiated or by default bundled with the generic UCI, if not specified by the convener. By providing a customized role based user interface for each user, the convener can ensure that the users participate in the workflow according to the prescribed protocol only. The generic user interface obtains the role specific plan from the document manager and displays the various objects and operations present in the user's object space. Figure 9 shows a generic user interface which displays role specific plan for a user in author role.

The user interface provides the default applications for manipulating the objects based on their type. In case of text objects, it simply provides a text editor to compose the text every time the object is accessed through the user interface. In the case of complex types, depending upon the MIME types, it can be extended to launch relevant applications which allow manipulation of those objects.



UCI Implementation as an Agent Server

Figure 8: Steps in building an agent-based distributed collaboration

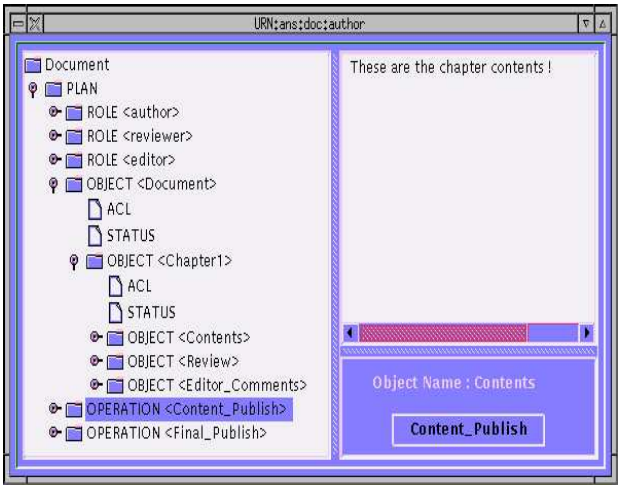


Figure 9: User Interface for the Author Role

Execution of Coordination Operations: The operation execution takes place through the user interface component of the UCI. The user interface enforces access control and synchronization on operation execution. Access rights are checked against the ACL, which is maintained on each operation node. In the current implementation, the execution of an operation results in the launching of mobile agents, which either clone the corresponding object or invoke specified remote methods on the remote copy of the object related to the operation.

An operation is performed only if its precondition is satisfied. Precondition checks are performed at two stages in workflow execution. First, the user interface performs this check and displays appropriate error messages. Second, when an agent reaches a remote target UCI as a part of the operation's remote action. If the precondition of the operation at the remote site is not satisfied, the agent waits for the precondition to be true. Proper synchronization primitives are implemented in the system to signal the agent when the precondition of the operation is satisfied.

When an operation is executed, the corresponding operation node with the document manager is examined. If any cloning is specified as a part of this operation, the document manager creates an agent to clone the object referenced by this operation to the target roles' UCIs. All the child objects and related operations of this object are packaged together and cloned at the remote sites. If any agent-action is specified as a part of this operation, the document manager creates an agent to execute the methods at the target UCIs. Figure 8 shows that to publish the chapter contents to remote hosts, an agent is created and dispatched. Here, an agent server, in our context the UCI, provides a secure execution environment for the hosting agents by a proxy based access-control mechanism [21].

Each agent carries with it credentials, which include the identity of the agent's owner. When an agent arrives at a host, the agent-server checks its credentials and verifies the agent's identity. Upon success, the agent-server creates an exclusive proxy of the document manager for this agent and embeds the identity of the agent's owner in the proxy. Later, when the agent invokes a method in the proxy, the proxy transparently invokes the corresponding method in the document manager with the agent's owner identity. The proxy mechanism enables the document manager to impose access control based on the agent's owner identity.

5 Related Work

The work presented here relates to many existing ideas from a number of areas such as computer-supported workflow, workflow specifications, workflow management, role-based security, and mobile agent systems. Our work integrates these ideas into an XML schema for workflow specification and interfaces this schema with Ajanta mobile agent based middleware for implementing distributed workflow systems.

Several systems have been designed as general purpose systems to build desired collaboration and workflow environments using library modules and interfaces [11, 16]. However, in these systems coordination policies still need to be programmed using procedural languages. COCA [10] is a framework for collaborative objects coordination, separating specification of coordination from computation. Our work is related to COCA in that it also uses roles to specify policies for workflow. However, our approach differs from COCA in that we use XML for specification of various coordination policies and description of roles unlike a first order logic language used in COCA. Our model makes the specification and the underlying implementation framework more decoupled. XML is also more readily understood by both human users and software systems.

Coordination specifications indicate how the dependencies between activities should be managed in a workflow. In workflow design, a process language allows one to define the activities [18] [1]. A number of groups have proposed process definition meta models. These include OMG's XMI (XML Meta Data Interchange)[12], UML [15], and Wfmc [24] process definition meta-language. Moreover, workflow specification languages [23, 2, 13] are studied based on state and activity charts. State and activity charts can be easily incorporated into our XML schema. In our approach, the XML based plan description corresponds to a process level description, and the agent-based middleware is the execution infrastructure. Our work is fundamentally different from Wfmc [24], which is concerned with interoperability of workflow systems. It mainly focuses on data exchange related issues.

Many other workflow [14, 2] systems have introduced a generic middleware for implementing workflow. In [7], a component based Java Beans supported suite is introduced for building synchronous collaborative applications. This requires a middleware to be implemented using the component suite, which is limited to current Java Beans tools [7]. In our model, XML specification can easily tie existing middleware and other components to realize a workflow system. The recent development of internet based workflow requires asynchronous workflow model which can be easily abstracted in agent based workflow systems.

The use of mobile agents in workflow systems has also been investigated by others researchers [6]. In [8], agent enhanced workflow is discussed, where an agent layer is wrapped around existing workflow applications. In contrast, our approach uses mobile agents at the implementation level for supporting workflow in distributed collaborations.

6 Conclusion

The main contribution of this paper is in developing a methodology for building a distributed workflow systems using a high level specification in XML and then interfacing this specification with a mobile agent based middleware, to realize the desired environment. The workflow plan is specified in XML in terms of shared objects, user roles, role-based security requirements, operations, and workflow requirements. The agent based middleware interprets the XML description of the environment, and transparently launches agents to other users' nodes when operations need to be performed to ensure workflow requirements.

We have presented our approach using a proof-of-concept implementation of an example workflow authoring system using Ajanta. Especially noteworthy is the fact that we are able to leverage an agent based middleware for supporting the security requirements prescribed in the workflow specifications as Ajanta provides security related components like authentication, public key maintenance, access control, host resource protection, and cryptographic services.

There are several areas that we plan to investigate further in our research. We plan to add more dynamic behavior to the system such as dynamic delegation of access rights and enabling participants to dynamically add entities to workflow. Moreover, we plan to integrate state and activity charts in our schema and extend the security policy so that it can incorporate workflow states. Also, an extension of our user interface will support manipulation and display of various kinds of objects depending on their MIME types.

REFERENCES

1. BUSSLER, C. Enterprise-Wide Workflow Management . *IEEE Concurrency 7*, 3 (July-September 1999), 32–43.

2. D. WODTKE, J. WEISSENFELS, G. W., AND KOTZ-DITTRICH, A. Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In *International Conference on Data Engineering* (1996).

3. DEMURJIAN, S., TING, T., AND THURAISINGHAM, B. User-role based security for collaborative computing environments . *Multimedia Review 4*, 2 (Summer 1993), 40–47.

4. DENGI, C., AND NEFTCI, S. Dflow workflow management system. In *International Conference on Database and Expert Systems Applications (DEXA)* (1997), IEEE, pp. 16–21.

5. DEWAN, P., AND SHEN, H. Access Control for Collaborative Environments. In *Proceedings of the ACM Conference on CSCW* (1992), pp. 51–58.

6. GAIL KAISER, A. S., AND DOSSICK, S. A Mobile Agent Approach to Lightweight Process Workflow. Available at URL ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-021-99.pdf.

7. GURUDUTH BANAVAR, SRI DODDAPANENI, K. M., AND MUKHERJEE, B. Rapidly Building Synchronous Collaborative Applications By Direct Manipulation . In *CSCW 98* (98), pp. 139–148.

8. J.W. SHEPHERDSON, S. T., AND ODGERS, B. R. Cross Organisational Workflow Co-ordinated by Software Agents. Available at URL http://www.zurich.ibm.com/ hlu/WACCworkshop.

9. KARNIK, N., AND TRIPATHI, A. A Security Architecture for Mobile Agents in Ajanta. In *Proceedings of the International Conference on Distributed Computing Systems 2000* (April 2000).

10. LI, D., AND MUNTZ, R. COCA: Collaborative Objects Coordination Architecture. In *Proceedings of CSCW'98* (1998), pp. 179–188.

11. MCCANNE, S., BREWER, E., AND KATZ, R. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Seventh International Workshop on Network and Operating System Support for Digital Audio and Video* (May 1997).

12. OMG . XML Meta Data Interchange. http://www.omg.org.

13. P MUTH, D. WODTKE, J. W., WEIKUM, G., AND DITTRICH, A. K. Enterprise-wide workflow management based on state and activity charts. In *Workflow Management Systems and Interoperability*, T. O. A. Dogac, L. Kalinichenko and A. Sheth, Eds. Springer Verlag, 1998.

14. PETER MUTH, JEANINE WEISSENFELS, M. G., AND WEIKUM, G. Workflow history management in virtual enterprises using a light-weight workflow management system . In *Proceedings of the 9th International Workshop on Research Issues in Data Engineering* (1999).

15. RATIONAL SOFTWARE. UML Documentation . Available at URL http:www.rational.com/uml/resources.

16. ROSEMAN, M., AND GREENBERG, S. Building Real-time Groupware with GroupKit, A Groupware Toolkit. In *ACM SIGCHI'96* (March 1996).

17. SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. Role-Based Access Control Models. *IEEE Computer 29*, 2 (February 1996), 38–47.

18. SCHMIDT, M.-T. The Evolution of Workflow Standards . *IEEE Concurrency 7*, 3 (July-September 1999), 44–52.

19. SOLLINS, K., AND MASINTER, L. RFC 1737: Functional Requirements for Uniform Resource Names. Available at URL http://www.cis.ohio-state.edu/htbin/rfc/rfc1737.html, December 1994.

20. TRIPATHI, A., KARNIK, N., VORA, M., AHMED, T., AND SINGH, R. Mobile Agent Programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems* (May 1999).

21. TRIPATHI, A. R., AND KARNIK, N. M. Protected Resource Access for Mobile Agent-based Distributed Computing. In *Proceedings of the 1998 ICPP Workshop on Wireless Networks and Mobile Computing* (August 1998), IEEE Computer Society, pp. 144–153.

22. W3C. Extensible Markup Language (XML) 1.0, W3C Recommendation 10. Available at URL http://www.w3.org/TR/, February 1998.

23. WODTKE, D., AND WEIKUM, G. A Formal Foundation For Distributed Workflow Execution Based on State Charts. In *International Conference on Database Theory* (1997).

24. WORKFLOW MANAGEMENT COALITION. Interface 1 - Process Definition Interchange V 1.0 Final . http://www.wfmc.org, 1998.