

Distributed Data Management in Workflow Environments

Gustavo Alonso
Institute of Information Systems
ETH Zentrum (IFW C 47.2)
CH-8092 Zürich, Switzerland
alonso@inf.ethz.ch

Berthold Reinwald
IBM Almaden Research Center
650 Harry Road (K55/B1)
San Jose, CA 95120-6099, USA
reinwald@almaden.ibm.com

C. Mohan
IBM Almaden Research Center
650 Harry Road (K55/B1)
San Jose, CA 95120-6099, USA
mohan@almaden.ibm.com

Abstract

Most existing workflow management systems (WFMSs) are based on a client/server architecture. This architecture simplifies the overall design but it does not match the distributed nature of workflow applications and imposes severe limitations in terms of scalability and reliability. Moreover, workflow engines are not very sophisticated in terms of data management, forgetting the fact that workflow is, to a great extent, data flow. In this paper, we propose a novel architecture to address the issue of data management in a WFMS. This architecture is based on a fully distributed workflow engine for control flow, plus a set of loosely synchronized replicated databases for data flow. The resulting system offers greater robustness and reliability as well as much better data handling capabilities than existing approaches. To better illustrate this novel architecture and its implications, two commercial systems are employed in this paper: FlowMark, as the workflow engine, and the replication capabilities of Lotus Notes, as the support system for distributed data management.

1 Introduction

Workflow Management Systems, WFMSs, are the first viable proposal to effectively coordinate multiple, heterogeneous information resources and monitor the overall progress in the execution of a business process. Precisely because of the emphasis on monitoring, most existing WFMSs tend to be based on a client/server architecture with a centralized repository. In practice, however, this

approach is inefficient and introduces severe limitations in terms of robustness, reliability, and scalability. Similarly, since the execution of a business process is usually based on a process model describing the control flow between workflow participants, many workflow engines have concentrated only on implementing the control flow, neglecting the data flow aspects. FlowMark [14], for instance, is one of the few workflow products that provides some built-in capabilities for passing data among activities. However, the *data repository* is intended only for storing control data, i.e., application data employed to evaluate the transition conditions governing the control flow. Users, however, lacking a good underlying data management system, opt for storing all the application data in those repositories, which often results in serious performance and scalability problems.

To address both issues, centralization and the lack of suitable data management capabilities, a different architecture is required. Preliminary steps in this direction were taken by the proposals in INCAS [2] and Exotica/FMQM [1] but without regard for the data management problem. In this paper, we address this limitation by extending the architecture of Exotica/FMQM to incorporate data management capabilities. The approach taken is to base the data handling mechanisms on a set of loosely synchronized replicated databases. These replicated databases are used as the common distributed repository for all the sites participating in the execution of a business process. The advantage of this approach is that the burden of data handling is no longer on the workflow engine, thereby augmenting its capacity as a coordination tool, and that the data manager can support additional functionality that would be very difficult to incorporate into a workflow engine.

In terms of a practical implementation, we have resorted

to existing commercial systems. The functionality these systems provide is fairly common and found in a variety of products, thereby guaranteeing that the results proposed can be also achieved on designs based on different components. The distributed environment is provided by Exotica/FMQM (FlowMark on Message Queue Manager) which offers FlowMark functionality modified to work in a set of autonomous servers that cooperate to complete the execution of a process. Persistent messaging is implemented using MQSeries [3, 21]. The data management functionality is based on Lotus Notes Release 4, a document store which supports replication [12, 15, 16]. Lotus Notes has been chosen as the data manager because it provides a sophisticated document management architecture as well as a powerful replication mechanism. Technical details aside, the contribution of the paper is twofold. First, the problem addressed, although very common in distributed workflow environments, has been largely ignored in the research literature. To our knowledge this is the first attempt to formalize the coupling of a workflow management system and corresponding data handling capabilities. Second, and to the extent allowed by the space provided, the paper provides an overview of how issues such as scalability and performance are being addressed in practice in large corporations.

The paper is organized as follows: in the next section, an example application is discussed to motivate the rest of the paper. Section 3 briefly describes Exotica/FMQM. Section 4 describes in more detail the role of the data manager in a workflow system and its functionality in terms of document replication. Section 5 presents the overall architecture incorporating the distributed WFMS and the data manager. Section 6 points out some limitations and open problems in the approach presented. Section 7 discusses related work and Section 8 concludes the paper.

2 Motivation and Example

As an example of the problems addressed in the paper, consider the following business process. The process is a rough approximation of the procedures followed during the processing of an application for an international patent. A patent application is a complex document encompassing a great deal of legal and technical information. Upon submission, the first step is to ensure that all the necessary documentation has been provided. Once the application has been found to be complete, it is assigned to a particular area (software, mechanical engineering, chemistry, electronics, and so forth). Then the claim is forwarded to special department in which the patent is examined and any relevant literature (books, articles, journal or conference papers, technical reports) as well as existing related patents are added to the application as part of the overall case. Once this search is completed, the application plus the added documentation

are sent for evaluation. The evaluation often involves a team of experts that analyzes the application and existing patents (evaluators do not search for relevant documentation, they work with what is provided). Frequently, the applicants are asked for clarifications and rewriting of the application to avoid conflicts with existing patents and laws. This might require several rounds until finally the evaluation team gives a recommendation.

The importance of this example is in its data handling requirements. Each patent application involves a significant number of documents referencing other information sources as well as other patents. Each existing patent is in itself a large collection of documents. Additional information such as articles and scientific papers are smaller in size but can be quite numerous. Besides the application itself, related patents and relevant articles are also part of the necessary documentation. Hence, transferring the "case" from the initial stages to the evaluation stages involves moving a great deal of information around (often from country to country as it is the case with the European Patent Office).

As this examples shows, any attempts to use a workflow system in such an environment will be only as successful as the data management capabilities it provides. In practice, and as proven by many workflow installations, it is the ability to pass data among the participants the main factor determining the applicability of a given WFMS. Most existing commercial workflow engines do not provide any support to deal with data management, although it is obvious that the transferring of the information is an integral part of the workflow process. The best solutions combine an imaging system with the workflow engine but the integration is often poor and the workflow engine has no real control over the flow of data, which complicates the task of adjusting it to the flow of control. In what follows, a distributed architecture is proposed as a first step to integrate the flow of control and data taking advantage of systems that have been optimized for only one of the two tasks. We believe that the result is more flexible and efficient than what has been so far proposed in the literature or implemented in practice.

3 Exotica/FMQM: Distributed Workflow

This section briefly summarizes the distributed architecture of *Exotica/FMQM* - FlowMark on Message Queue Manager. For a more detailed description see [1]. First, the FlowMark model is reviewed, and then we extend the centralized FlowMark design to a distributed architecture. The choice of FlowMark as the base architecture does not result in a loss of generality as FlowMark closely follows the WfMC reference model (Workflow Management Coalition, WfMC [9]).

3.1 Workflow Model

A business process is, in general, an acyclic directed graph in which nodes represent steps of execution and edges represent the flow of control and data. In FlowMark [14, 10, 11], the main components of the *workflow model* are: *processes*, *activities*, *control connectors*, *data connectors*, and *conditions*. Activities are the nodes in the process graph and they represent the steps to be completed. Control connectors are used to specify the order of execution between activities and are represented as directed edges in the process graph. Data connectors specify the flow of information from one activity to another in the form of mappings between the data containers of the activities and are also expressed as directed edges in the process graph. Each activity has one *input data container* and one *output data container*. Finally, conditions specify when certain events can happen. There are three types of conditions: *transition conditions* associated with control connectors determine whether the connector evaluates to true or false; *start conditions*, which specify when an activity will be started; and *exit conditions*, which are used to specify when an activity is considered to have terminated.

3.2 A distributed architecture

Exotica/FMQM is a distributed version of FlowMark based on a generic queuing system with recoverable queues. The main idea behind its design is that workflow models are partitioned into execution scripts that are distributed to the nodes where the actual execution of these parts will take place. Note that in workflow systems it is possible to predict much of the execution by analyzing the model and assigning the activities and tasks in advance, so the actual partition of the model is not a complex algorithm. Communication between the different nodes takes place through persistent queues, thereby guaranteeing the atomicity of operations, enhancing the overall reliability, and allowing asynchronous communication between the different components. The overall architecture of Exotica/FMQM is geared towards control flow and there is no significant support for data management, as was pointed out above. This architecture, however, solves many of the problems being encountered by existing workflow products. In particular, it offers better resilience to failures and better scalability than existing configurations. To take advantage of these features, Exotica/FMQM is used as the basic architecture for the system. In what follows, a brief description is provided regarding the most relevant details of its implementation.

Persistent queues guarantee that once a message is placed on a queue and the corresponding transaction is committed, the message becomes persistent until it is retrieved. Messages are added and retrieved from a queue us-

ing *PUT* and *GET* calls, which can be issued both on local and remote queues. Using these generic primitives, Exotica/FMQM implements a distributed architecture in which process execution is handled as follows [1]. When a user creates a process, the process is first compiled and then divided into several parts, which are distributed to the appropriate node(s). A node corresponds to a physical machine. It is possible to see this node as a “server” to other nodes in a multilayer architecture, but this is irrelevant for our purposes here. In what follows we will use the term node to refer to a remote location where workflow navigation takes place. Upon receiving its part of the process, a node creates a *process table* to store this information and starts a process thread which handles the execution of instances of that process. The process table must be saved in persistent storage as it describes which actions the node must take at each step in the execution of a process. This table is, in general, an “append only” structure. For efficiency reasons a copy can be kept in main memory. After creating this table, the process thread creates a queue for communicating with other nodes all information relevant to instances of the process. There will be a queue per process type. During execution, the information regarding the process instances will be stored in an *instance table* handled by activity threads. These threads are responsible for the execution of individual activities within an instance and are created when an activity instance becomes ready for execution. All *PUT* and *GET* calls to the queue are executed in a transactional manner, i.e., their effects are not permanent until the transaction commits.

When a process instance is started, messages are sent to the nodes where the starting activities are located. This is done by *PUT*ting the messages in the appropriate queues. At each node, the arrival of a message triggers the process thread which will analyze the message, check in the process table for the steps that must be taken and start the appropriate activity threads. Messages are not “retrieved” but “browsed”, which is important in case of failures. Browsing leaves the message in the queue, preserving its persistent. Once started by the process thread, the activity thread first checks the process table to find the data pertaining to that activity. With this data, it creates an entry for that activity in the instance table where it will record the progress of the activity. It then checks the start condition of the activity. If it cannot be evaluated, because not all incoming control connectors have been evaluated, it makes a note in the instance table about what control connector has been evaluated and goes to sleep until the process thread awakens it again with new messages. If the start condition is false, then the activity is considered terminated. Dead path elimination messages are sent along the outgoing control connectors using *PUT* calls to the corresponding queues; upon receiving such messages, the corresponding entries are removed from

the instance table and any related messages retrieved from the queue are discarded. All the operations performed upon termination of the activity are one atomic action. This can be accomplished by using the transactional properties of the GET and PUT calls.

If the start condition evaluates to true, then the corresponding application is invoked, passing to it the data in the input template. When the application terminates, the output data returned is stored in the instance table. Then the activity thread evaluates the exit condition of the activity. If it is false, the application is invoked again. If it is true, the activity is considered terminated. Upon successful termination, the outgoing control connectors' transition conditions are evaluated and messages are sent to the appropriate nodes with PUT calls to their queues. The output data is sent to the nodes specified in the process table as outgoing data connectors. Then the entry is removed from the instance table and all messages corresponding to the activity are retrieved from the process queue. Note that each message corresponds to one activity, once the activity successfully terminates there is no need to keep the messages. These operations are performed as an atomic transaction as explained above.

4 Distributed Data Flow

In a distributed environment, application data associated with workflow processes need to "travel" along with the process. However, the application data should not be stored in the workflow management system itself, for reasons of efficiency, and hence a data manager is required. In this section, we describe the relevant aspects of the data flow starting with general document management in Lotus Notes [15, 16] and then describing in some detail its replication capabilities.

4.1 Document and form management

Notes is a document data store with a cross-platform architecture. It has its own specialized storage and indexing subsystem, which is optimized for groupware applications. For our purposes here, a Notes database consists of *documents*, *forms*, and *views*. Documents can be used to store multimedia data types like graphics, voice, attachments, tables, and embedded OLE objects, among others. Forms in Notes include integrated form management capabilities. They are used to create and edit documents. Views in Notes apply selection criteria on the documents in a database, and allow browsing of the documents in a database.

A Notes database contains a set of application documents, and provides functionality to create, list, edit, and delete documents. Each document (basically a record) contains some header information and a set of items. The

header information in a document associates the document with a form. This form is used to interpret, display, and edit the document. Many document management systems are tightly coupled with form management systems, as is also the case in Notes. Different documents in the same database can have different sets of items and can be associated with different forms. *Items* in documents are triples consisting of a name, a type, and a data value. This item structure makes documents self-defining. To cross-reference other information sources, doclinks between documents are defined. In the patent example, all application documents will be stored in Notes databases. Each patent application would be a document, *viewable* through different forms (for the person in charge of searching for relevant documentation, for the patent reviewer, for the applicant, for the patent office lawyers, etc.). All references to other documents (like relevant literature or related patents) are stored as doclinks.

4.2 Replication Mechanism

When, as in the patent example, the application runs in a distributed environment and with a significant load, additional mechanisms are needed to provide reasonable performance. If a centralized repository is used, the repository is liable to become a bottleneck and a single point of failure. Alternative solutions include using an imaging system or a distributed database. An imaging system has the problem that documents literally move from one place to the next. This creates very complex problems if a document needs to be used simultaneously at different locations. On the other hand, distributed databases tend to be fairly restrictive. For instance, replication in commercial databases is, in the best cases, limited to primary copy schemas [4], which does not allow to update a document at all locations. To solve this impasse, we propose to use the semantic knowledge embedded in the workflow engine. By exploiting this knowledge, a much "looser" (asynchronous) distributed document management policy can be implemented. For this purpose the replication mechanisms provided in Lotus Notes are used.

Notes' philosophy of replication can be thought of as a lazy exchange of modifications between replicas. Users at different sites can access and modify the same documents in different replicas. Replicas are kept in sync, but only in the course of time. Replicas share a common id, called *replica id*, which is the database id of the original database.¹ Notes allows us to specify the *direction* of replication (uni- or bi-directional), *when* to replicate, and the *contents* of replicas (subset of documents in a database). Replication can occur automatically according to a specified time schedule, or manually through server or client commands. Time scheduled replication is useful for server-to-server replication,

¹ The replication id distinguishes a replica from a casual database copy, which has its own different database id.

whereas manually triggered on-demand replication is very appropriate for client/server replication, where the client is a mobile user. Three different types of replicas can be distinguished, depending on the contents of the replica: full replica, partial replica, and replica stub. A *full replica* contains all of a database's documents and design features (such as forms and views), whereas a *partial replica* only contains selected documents specified in a selection criteria.² *Replica stubs* contain only the database design but no documents. Replica stubs are only useful to exchange database templates. In addition, Notes Release 4 introduced *field-level replication* which increases the performance of the replication process as only fields that have changed in a document are exchanged when copies are synchronized. As bi-directional replication is supported and concurrent changes are allowed in the replicas, *replication conflicts* can occur. A replication conflict happens when two or more users edit and change the same document in different replicas between replications.³ Replication conflicts are resolved by determining the document with the most changes as the main document and introducing the other documents as replication conflict documents (responses) to the main document. The replication conflict document can be used to consolidate the different document versions. If a document is edited in one replica and deleted in another, the deletion takes precedence.

Although these concepts are essentially database replication ideas, with a certain choice of the replication parameters the approach can be exploited to implement distributed data flow. This will be illustrated in the next section.

5 Distributed Workflow Data Management

This section describes the overall system combining Exotica/FMQM for control flow and a Lotus Notes-based system for data flow.

5.1 Architecture

The main idea for the integration of the control and data flow consists in establishing the mechanisms necessary to ensure that a message between nodes (on Exotica/FMQM) corresponding to the triggering of an activity also triggers

²The selection criteria for partial replica are specified in Notes replication formulae. Replication formulae are basically queries in Notes macro language against the database to retrieve the data of interest for replication. *Selective replication* saves local disk space as well as increases the performance of the replication process as only a subset of the data of interest for a site are replicated.

³Replication conflicts are comparable to *save conflicts* in Notes. A save conflict happens when two or more users edit the same document in a database on a server at the same time. Notes creates conflict resolution documents to support merging the different versions of a document.

the underlying data replication mechanisms at the data management level (Lotus Notes). Similar ideas have been introduced in the ActMan approach [23], although on top of a distributed relational database system prototype. The final goal of the approach is that any data required for the execution of the activity will be locally available while still being available at other sites. A *control node* represents the part of the system corresponding to Exotica/FMQM. Hence, it is the part of the system responsible for navigation, persistence of the messages, and interfacing with users and applications. A *data node* is the part of the system corresponding to Notes, i.e., that is in charge of providing access to the data. From an abstract point of view it is easier to treat them as separate entities, but in practice a control node and a data node can reside in the same memory space. Note that it is likely that several control nodes will be working off a single data node. The activities manipulated by the control node do not contain any data per se except a few simple variables (integers, booleans, strings) used to evaluate control flow conditions. Part of these variables will be *references* (documents id's) to data stored in the data node. Thus, control nodes only need to transfer simple variables. These ideas are summarized in Figure 1.

The separation between the control and data nodes presents us with several advantages. First, the data nodes are more likely to be compatible with the existing data management strategies of a company. As a result, the introduction of a workflow management system will not force changes on existing policies. Notes, however, requires the use of its own internal databases. But it must be noted that the data nodes do not need to be solely based on Notes. We have chosen Notes to simplify the explanation and provide details of a particular system. Second, separating the data from the control allows very interesting optimizations and greater flexibility in the handling of complex data. So far, workflow systems have limited themselves to documents but there is no reason to refrain from expanding their capabilities toward multimedia applications. This is only feasible if the control flow is independent of the data which is not the case in most of the existing commercial products. Finally, separating control and data nodes, along with the notion of clusters, allows a much more flexible architecture that will certainly have a significant impact on issues such as scalability, availability and fault tolerance.

5.2 Replication

We have already seen how control flow takes place and how a control node will interact with a data node. It remains to describe how data flow is implemented. In order to do this, it must be first established *when* the data flow should take place. The entire range of possible alternatives can be covered by describing two extreme scenarios: (1)

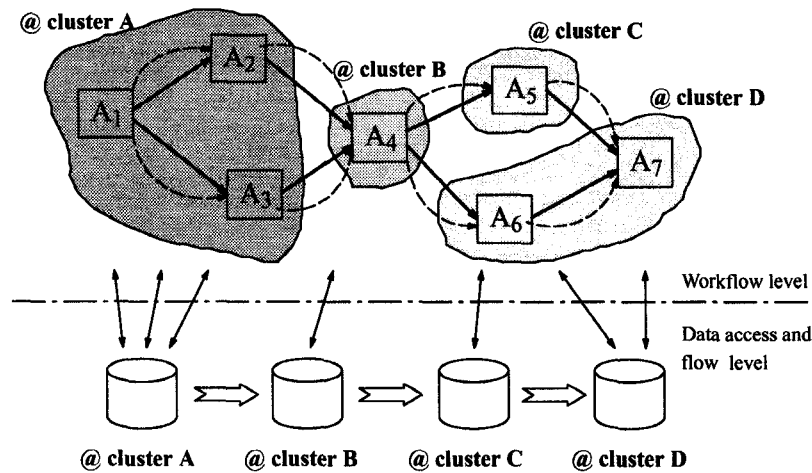


Figure 1. Interaction between a control node and a data node.

data flow takes place simultaneously with the control flow (as a control node sends a message about one activity to a second control node, the corresponding data node forwards the data to the second data node), and (2) data flow happens as soon as the document is accessed on the destination site (that is, the forwarding of the data only takes place upon request). In discussing the two cases, it needs to be kept in mind that an activity becoming ready for execution at a control node does not imply that the activity will be processed right away. Manual activities, for instance, end up in a work list and will not be executed until the user decides to do so, maybe days later. To propagate the data along with the activity is “overprotective” for manual activities. On the other hand, automatic activities (those executed without human intervention) would be unnecessarily delayed if they have to request all the data items one by one. Moreover, the performance of the data system can easily degrade if a series of single documents instead of sets of documents are exchanged between data nodes. These ideas are summarized in Figure 2.

We have opted for an intermediate solution that tries to accommodate the requirements of both types of activities. Since it is known at compile time whether an activity is manual or automatic, when a control node forwards control to another node for an automatic activity, it will also trigger the forwarding of the data. In the case of Notes, this forwarding takes place through replication. When a control node follows a control connector leading to a second control node, it will also prompt the data node to replicate its data at the second data node. We use a manually triggered (by the control node) uni-directional partial replica of the data. The reason for using uni-directional replication is that it matches the way data flows in a workflow environment (from output container to input container). For manual activities, repli-

cation also takes place via a uni-directional partial replica, however the actual transfer can be made manually (if it has not yet taken place at the time the user selects an activity for execution) or automatically as part of *document bundles* exchanged among data servers. The transfer of document bundles is triggered through Notes scheduled agents [15], e.g., once a day. In this way, data exchange can be made more effective by the replication of large amounts of data in one single operation. The size of the data bundles and their frequency are both adjustable parameters of the system. A further advantage of this approach is that if clusters of “equivalent” users exist, as pointed out above, the replication can be delayed until a user selects the activity, thus avoiding sending the data to all the clusters when only one of them will actually use it. To avoid that a user accesses an outdated document during activity execution, we code a timestamp in the work items, when the document on the source site was updated. We compare this timestamp with the timestamp of a possibly existing replica document. If the timestamps are the same, then there is no need for replication. In the other case, we need to replicate the document from the source site before we access it.

What we have presented so far with regard to data replication is fairly generic. The use of Lotus Notes as a concrete data node provides us with additional optimization opportunities. Notes replication, for instance, supports field-level replication which dramatically reduces the amount of data exchanged between node and improves the overall performance. If a user modifies only one field in a 1 Mbyte patent description, Notes transports only the one modified field to all the replicas. In addition, we exploit doclinks supported by Notes in our patent sample application. Patent applications refer to a lot of background information stored in archive databases at remote sites. This information does not

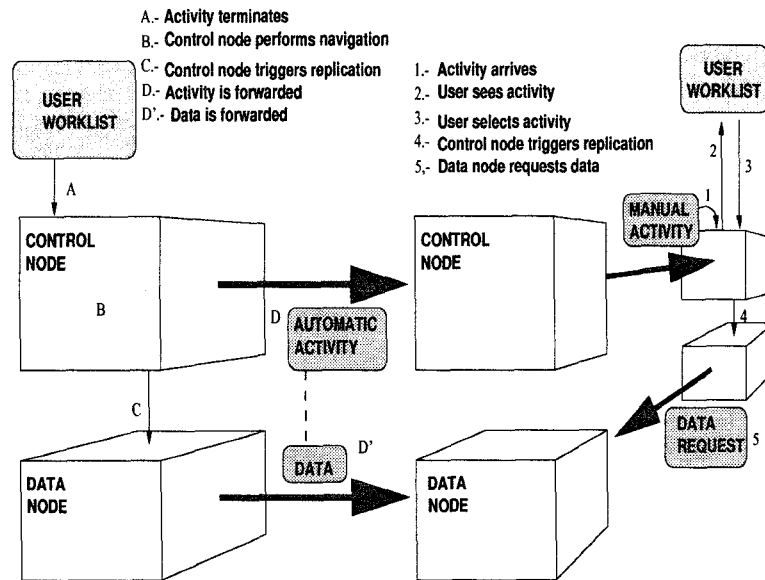


Figure 2. Coordination of distributed workflow and data management.

need to “flow” with the workflow as it is only occasionally accessed on request (following the doclink) by the user. The performance penalty for retrieving this kind of documents from a remote site as opposed to have it locally available is justified by improving the overall workflow performance as less data need to travel with the workflow.

6 Discussion and Future Work

The ideas described above can be divided in two major areas, distributed workflow execution and data management for distributed workflow applications. In both cases, this paper describes a possible architecture based on existing systems but similar ideas can be implemented with other systems. Additionally, in both cases there are a variety of issues that have not been discussed in detail in the paper for reasons of space and not being central to the design. In what follows some of these issues are briefly examined.

There are certain assumptions that must be made when distributing the execution of a workflow system. For instance, the notion of process owner must be changed as there is no centralized entity where the authority of a process owner can be exercised. Similarly, several operations that are trivial in a centralized environment become somewhat complex when implemented in a distributed system. As an example, the problem of detecting process termination could raise some complicated issues similar to those of snapshots and termination detection [5, 19]. To avoid these problems, we assume that each process has a unique starting activity and a unique final activity that can be used to detect where and who starts the process as well as when it

completes its execution.

Other interesting aspects of the distributed architecture are the management of work lists and the synchronization problem of ensuring that only one user actually executes an activity, which appears simultaneously on several work lists. In a centralized system, work lists are relatively easy to maintain, and the server handles race conditions so that only the first user performs an activity. These two features are complex to implement in a distributed environment. Our first approach has been to use remote GET calls to allow an activity to appear in several work lists. The first node to GET the message from that queue will be the one to which the activity is assigned. However, today, IBM’s MQI does not support remote GET calls and this complicates work list management.

Resilience to failures is ensured by using transactional PUT and GET calls, plus the ability to browse a queue without retrieving messages. When a message is received regarding an activity, it is not retrieved from the queue until the activity has terminated. Messages are kept until they are not needed anymore, and removed from the queue in an atomic operation. In this way, in case of failures which result in the loss of the instance table, a recovering node will restart the process thread which will find the messages still intact in the queue and, based on them, spawn again the appropriate activity threads which will reconstruct the instance table. This is the reason why instance tables do not need to be stored in stable storage and why process tables should be kept in stable storage. For the sake of completeness, it must be mentioned that an audit log is necessary to keep track of which activities have completed execution

and to store messages, in case the user decides to restart a finished activity. This audit log can be another queue, checkpointed every so often.

The described distributed data management approach exploiting workflow semantics promises tremendous performance improvements. The document databases are synchronized on a lazy basis depending on the workflow requirements. On the other hand, these databases might be difficult to use for existing non-workflow applications, as, from their perspective, it is not clear how current the data is. This will be a concern in particular, once there are loops in a workflow which result in a loop in the data flow. Problems also arise, once a document is required and updated by different process instances. Although Notes supports the merge of the documents to some extent, but the semantics for the end user might get fuzzy. Services are required, which spread out in a distributed system and find or compose the most current replica of a document in a distributed environment.

7 Related Work

There are many areas that have influenced or are related to workflow management. In Electronic Document Management (EDM), for instance commercial systems like Saros' Document Manager, Intergraph's DM2, Documentum' Documentum Server, IBM's VisualInfo, are concerned with the storage and life cycle of electronic documents. Form management systems can be traced back to early systems like TLA [26] where *form procedures* were proposed to organize the flow of forms. Electronic mail systems like LENS [17] provide means to distribute, forward and track information among individuals. Office automation prototype systems such as SCOOP [28] and DOMINO [13] actively support the automation of office procedures through the exchange of messages and distribution of data. Barbara et al. [2] proposed INCAS for distributed workflow management. In this model, each execution of a process is associated with an *Information Carrier*, which is an object that contains all the necessary information for the execution as well as propagation of the object among the relevant processing nodes. Transaction based applications [25], [18], [24], like the model based on extended nested sagas [8] for coordinating multi-transaction activities, or the Contract model, which provides reliable control flow between transactional activities [27]. In [6] and [7], a specification language and a transactional model for organizing long running activities is developed. Although intended for workflow systems, the primary emphasis is on long running activities in a transactional framework using triggers and nested transactions. The authors present a preliminary design using recoverable queues, which have similar semantics to the message queue interface (MQI) used in this paper.

Most of these early efforts did not address business processes in their full generality, most of them are centralized and fail to address issues such as high availability, failure resilience or scalability. Moreover, they tend to be transactional in nature and too centered around databases, which contrast with the reference model created by the workflow management coalition. For instance, systems like X-Workflow from Olivetti [22], use a central mediator to coordinate the control and data flow of a business process by distributing semi-structured e-mail messages to all users eligible to execute an activity. Once, a user selects an activity, an execution request is sent to the mediator which assigns the activity to the user and sends him/her the context data (input parameters) needed to execute the activity. Also, the mediator informs other users of this fact by sending a message which causes the intelligent mail system to remove the activity from the user's mail folder. This is a highly centralized approach characteristic of most existing commercial systems.

8 Conclusions

This paper has presented an architecture for managing the control and data flow aspects of a workflow system. The main contribution is the combined architecture based on separate systems optimized for different purposes: the workflow engine for the control flow, the data management system for the data flow. We see this architecture as offering significant advantages over existing solutions, as it provides a much better functionality without necessarily penalizing performance. In current commercial workflow systems, data flow is either done externally, with poor coordination and little flexibility, or embedded in the control flow, which has a serious impact on performance. The solution we propose in this paper minimizes the impact on the control flow, i.e. in the workflow engine, while allowing very sophisticated coordination between the activities and the data flow. As part of future work, we plan to enhance the data management features to include complex data types and optimize the data flow in large distributed environments (over WANs instead of over LANs).

Acknowledgments

Many of the ideas described in this paper are the result of the research efforts carried out within the Exotica project (<http://www.almaden.ibm.com/cs/exotica/>, [20]). We would like to thank Divyakant Agrawal, Amr El Abbadi, Mohan Kamath and Roger Günthör for their help in developing the Exotica/FMQM architecture.

References

- [1] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi, and M. Kamath. Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *IFIP WG8.1 Working Conference on Information System Development for Decentralised Organizations*, pages 1–18, Trondheim, Norway, Aug. 1995. Accessible via <http://www.almaden.ibm.com/cs/exotica>.
- [2] D. Barbara, S. Mehrota, and M. Rusinkiewicz. INCAS: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1):5–15, Winter 1996.
- [3] B. Blakely, H. Harris, and L. J.R.T. *Messaging and Queuing using the MQI: Concepts and Analysis*. McGraw-Hill, 1995.
- [4] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill, 1984.
- [5] K. Chandy and L. Lamport. Distributed Snapshots: determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3(1):63–75, Feb. 1985.
- [6] U. Dayal, M. Hsu, and R. Ladin. Organizing Long-running Activities with Triggers and Transactions. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 204–214, June 1990.
- [7] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-running Activities. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, pages 113–122, Aug. 1991.
- [8] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating Multi-transaction Activities. Technical Report CS-TR-247-90, Department of Computer Science, Princeton University, 1990.
- [9] D. Hollingsworth. Workflow management coalition: The workflow reference model. Document TC00-1003, Workflow Management Coalition, Dec. 1994. Accessible via <http://www.aiai.ed.ac.uk/WfMC/>.
- [10] IBM. *FlowMark - Managing Your Workflow, Version 2.1*. IBM, Mar. 1995. Document No. SH19-8243-00.
- [11] IBM. *FlowMark - Modeling Workflow, Version 2.1*. IBM, Mar. 1995. Document No. SH19-8241-00.
- [12] L. Kawell, S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Greif. Replicated document management in a group communication system. In *Proc. of the Conf. on Computer-Supported Cooperative Work, CSCW (Portland, Oregon)*, 1988.
- [13] T. Kreifelts and G. Woetzel. Distribution and Exception Handling in an Office Procedure System. In *Office Systems: Methods and Tools, Proc. IFIP WG 8.4 Work. Conf. on Methods and Tools for Office Systems*, pages 197–208, 1986. October, 22-24, Pisa, Italy.
- [14] F. Leymann and W. Altenhuber. Managing Business Processes as an Information Resource. *IBM Systems Journal*, 33(2):326–348, 1994.
- [15] Lotus Notes, Cambridge, MA. *Lotus Notes Release 4 Application Developer's Guide*, 1995.
- [16] Lotus Notes, Cambridge, MA. *Lotus Notes Release 4 Database Manager's Guide*, 1995.
- [17] T. Malone, K. Grant, K. Lai, R. Rao, and D. Rosenblitt. Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Office Information Systems*, 5(2):115–131, 1987.
- [18] D. McCarthy and S. Sarin. Workflow and Transactions in InConcert. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993. IEEE Computer Society.
- [19] J. Misra. Detecting termination of distributed Computations Using Markers. In *ACM Proceedings of the Symposium on Principles of Distributed Computing*, pages 290–294, 1983.
- [20] C. Mohan, G. Alonso, R. Günthör, M. Kamath, and B. Reinwald. An Overview of the Exotica Research Project on Workflow Management Systems. In *Proc. of the Sixth International High Performance Transaction Systems Workshop (HPTS)*, Asilomar, CA, 1995. Accessible via <http://www.almaden.ibm.com/cs/exotica>.
- [21] C. Mohan and R. Dievendorff. Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing. *Bulletin of the Technical Committee on Data Engineering*, 17(1):22–28, Mar. 1994. IEEE Computer Society.
- [22] Olivetti Systems & Networks GmbH. *Ibisys X.Workflow-Vorgangssteuerung auf der Basis von X.400*, 1994. Produktbeschreibung.
- [23] B. Reinwald and H. Wedekind. Automation of Control and Data flow in Distributed Application Systems. In *Database and Expert Systems Applications (DEXA), Proc. of the Int. Conf. in Valencia, Spain*, pages 475–481, Berlin, 1992. Springer-Verlag.
- [24] A. Sheth. On Multi-system Applications and Transactional Workflows, Bellcore's projects PROMP and METEOR, 1994. Collection of papers and reports from Bellcore.
- [25] C. Tomlison, P. Attie, P. Cannata, G. Meredith, A. Sheth, M. Singh, and D. Woelk. Workflow Support in Carnot. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993. IEEE Computer Society.
- [26] D. Tschritzis. Form Management. *Communications of the ACM*, 25(7):453–478, July 1982.
- [27] H. Waechter and A. Reuter. The ConTract Model. In A. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7, pages 219–263. Morgan Kaufmann Publishers, San Mateo, 1992.
- [28] M. Zisman. Representation, specification, and automation of office procedures. Ph.d. thesis, University of Pennsylvania, 1977.