Functionality and Limitations of Current Workflow Management Systems

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

D. Agrawal **A. El Abbadi**
Department of Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
{agrawal,amr}@cs.ucsb.edu

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

Abstract

Workflow systems hold the promise of facilitating the everyday operation of many enterprises and work environments. As a result, many commercial workflow management systems have been developed. These systems, although useful, do not scale well, have limited fault-tolerance, and are inflexible in terms of interoperating with other workflow systems. In this paper, we discuss the limitations of contemporary workflow management systems, and then elaborate on various directions for research and potential future extensions to the design and modeling of workflow management systems.

1 Introduction

Workflow management is one of the areas that, in recent years, has attracted the attention of many researchers, developers and users. For the users, it has finally made commercially available tools and functionality for which there has been an important demand for quite some time. Concepts such as *computer supported cooperative work*, *paperless office*, *form processing*, *cooperative systems*, and *office automation*, have been delayed decades, in some cases, for

the technology and know-how required to implement real systems. The technology has been provided by advances in networking, distribution and ever faster and cheaper computers and the know-how by the much advertised business process re-engineering techniques. And while these concepts were becoming a reality, the demand for solutions capable of integrating all the information resources of organizations has been increasing at a surprising pace. If there is any proper characterization of the information resources of any modern corporation, it is as a collection of widely heterogeneous, largely distributed and loosely coupled computing environments. The decentralization of the corporation, the decentralization of the decision making, the need for very detailed information about every day activities as well as the emphasis on client/server architectures, the relevance of federated systems and the increasing availability of distributed processing technology (WWW, CORBA, OLE, Java) are all trends that indicate that the days of monolithic, centralized information processing are over. But to make this a reality first there must be a way to implement large and heterogeneous distributed execution environments where sets of interrelated tasks can be carried out in an efficient and closely supervised fashion. This is where workflow management systems come in to the picture.

Workflow management systems (WFMS) are used to coordinate and streamline *business processes*. Typical business processes are loan approvals, insurance claims processing, and billing. These business processes are represented as *workflows*, i.e., computerized models of the business process, which specify all the parameters involved in the completion of these processes. Such parameters range from defining the individual steps (entering customer information, consulting a database, getting a signature), to establishing the order and conditions in which the steps must be executed including aspects such as data flow between steps, who is responsible for each step, and the applications (databases, editors, spreadsheets) to use with each activity. A WFMS is thus the set of tools used to design and define workflow processes, the environment in which these processes are executed, and the set of interfaces to the users and applications involved in the workflow process. The workflow concept has been so successful that in a few years several hundred products have been launched into the market and all analysts agree that in the near future this market will enjoy a substantial growth rate.

It is clear why users and developers are interested in the topic but, what about researchers? After all, these ideas will not sound unfamiliar to many. Until now, the main

tools of enterprise computing, databases and TP-monitors, have been successfully used to solve similar problems. However and in spite of their popularity, workflow systems are far from providing the functionality, reliability and robustness characteristic of existing database systems, all key elements to become the backbone of corporate computing. In particular, there are many instances in which the expectations from the users and the actual features provided by the systems are not well correlated. There are many reasons for this, the main ones being the novelty of the application area and the lack of maturity of the first generation of workflow products. But it is also a widely acknowledged fact that the requirements of a workflow system in terms of scalability and system wide reliability exceed those of database and transaction processing technology. Hence the need for further research in the area.

In this paper, we first describe the basic concepts of workflow management in Section 2. In Section 3, we discuss the limitations of existing systems and Section 4 presents a discussion of various areas on current research for enhancing the capabilities of current workflow management systems.

2 Large Scale Workflow Systems

The basic concepts of workflow management can be best introduced using the definitions provided by the *Reference Model* of the *Workflow Management Coalition*, *WfMC*, an international organization leading the efforts to standardize workflow management products [wfmcM]. Concrete architectural details are based on FlowMark, IBM's workflow product.

2.1 Types of Workflow Systems

There are many parameters involved in the specification of a workflow system. In spite of the efforts of the Workflow Management Coalition, the term workflow is still very fuzzy and used in many different contexts. Moreover, it is generally associated with the concept of business processes, which is also not very precise. Probably for these reasons, most of the existing classifications are based on the intended used or on the underlying technology.

A widely accepted taxonomy distinguishes between *administrative*, *ad hoc*, *collaborative*, and *production* workflows. The basic parameters of this classification are the similarities among the business processes involved and their value to the associated enterprises. However, it is also possible to organize them according to the task complexity and the task structure. Figure 1 summarizes both approaches.

In general, administrative workflows refer to bureaucratic processes where the steps to follow are well established and there is a set of rules known by everyone involved. Examples are the registration for courses in a university, applying for a degree after finishing the dissertation, registration of a vehicle, and almost any other process in which there is a set of forms to be filled and routed through a series of steps. Note that this type of workflows leads almost naturally to the idea of form processing, a new term for the older concept of the paperless office, and is also associated with large scale systems where the number of processes involved tend to be very high. For instance, a typical billing application may involve several million processes a year.

Ad Hoc workflows are similar to administrative workflows except for the fact that they tend to be created to deal with exceptions or unique situations. This depends on the users involved. While for a university the process of applying for a degree is an administrative procedure, for a student it is something that happens only once and therefore ad hoc from that point of view. If the process is of sufficient complexity, it is possible to define a workflow to help with its coordination and management. It may also be the case that the situation is not exceptional but each particular instance is unique. For example, each journal follows a different protocol for the submission process. Authors, especially given the length in time of these processes, may want to leave the coordination of the different steps in the hands of an ad hoc workflow system. This brings an important aspect of ad hoc workflows. While the actual process may be unique, the user will be in general be involved in a variety of these processes. The reason for using a workflow system with these characteristics is not the difficulty of tracking each separate process, but the problem of keeping track of all of them simultaneously.

The third class of workflows, collaborative, is mainly characterized by the number of participants involved and the interactions between them. Unlike other type of workflows, which are based on the premise that there is always forward progress, a collaborative workflow may involve several iterations over the same step until some form of agreement has been reached or it may even involve going back to an earlier stage. A good example is the writing of a paper by several authors. It would be very difficult to model such a process using tools that are not geared for collaboration since it is almost impossible to predefine the steps to follow. Note that this steps should not be mistaken with milestones, which can be predefined. Moreover, collaborative workflows tend to be very dynamic in the sense that

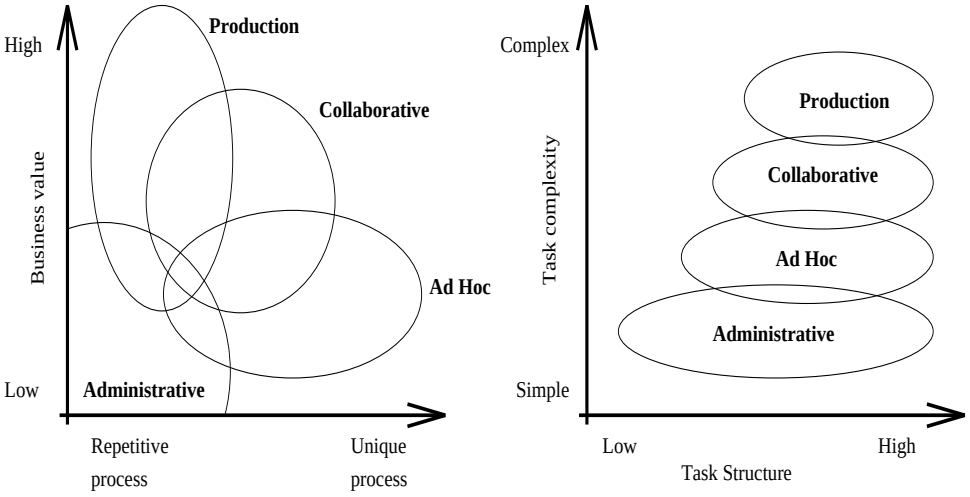

Figure 1: A rough classification of workflow management systems

they are defined as they progress. Taken to the extreme, it may be questionable whether these type of processes follow within the category of workflow systems since most of the coordination is done by humans with the system limited to the role of providing a good interface to recorded interactions, usually by e-mail. There is quite a number of products advertised as workflows that follow into this category.

Production workflows are the high end of these systems. They can be characterized as the implementation of critical business processes, that is, those that are directly related to the function of the organization. Credit and loan applications and insurance claims are the typical examples, but note that the difference between administrative and production workflows is sometimes a matter of perspective. Usually, when talking about production workflows, the main points to consider are the large scale, the complexity and heterogeneity of the environment where they are executed, the variety of people and organizations involved, and the nature of the tasks. In particular, production workflows tend to be executed over heterogeneous systems, frequently legacy applications, and it is very important to have monitoring tools to allow the statistical analysis of the execution of these processes. The ideas discussed below apply mainly to production workflows.

Another classification often found in the literature is according to the underlying technology: *mail-centric*, *document-centric* and *process-centric*. Mail-centric systems are based on electronic mail and can be roughly associated with collaborative and ad hoc workflows. Given the characteristics of the communication media used, e-mail, these systems are not suitable

for production workflows or environments with a large number of processes. Document-centric systems are based on the idea of routing documents and the ability to interact with external applications is limited. Many administrative workflows, those based on forms, can be implemented using document centered systems. Process-based systems correspond to production workflows. They generally implement their own communication mechanisms, are built on top of databases and provide a wide range of interfaces to allow interaction with legacy and new applications. This is the type of systems addressed here.

2.2 Workflow Model

The core of any workflow system is formed by business processes. The reference model defines a business process as "a procedure where documents, information or tasks are passed between participants according to defined sets of rules to achieve, or contribute to, an overall business goal" [wfmcM]. A *workflow* is a representation of the business process in a machine readable format. Hence, a *workflow management system*, WFMS, is "a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic" [wfmcM].

A workflow model is an acyclic directed graph in which nodes represent steps of execution and edges represent the flow of control and data among the different steps. The components described below follow the meta-model proposed by the Workflow Management Coalition [wfmcM]. This model is only an abstraction and does not provide implementation details. These are described based on FlowMark's model, depicted in Figure 2:

- **Process**, a description of the sequence of steps to be completed to accomplish some goal. It should have a name, version number, start and termination conditions and additional data for security, audit and control. A process consists of *activities* and *relevant data*.

- **Activity**, or each step within a process. Activities have a name, a type, pre- and post-conditions and scheduling constraints. They can be *program activities* or *process activities*. A *program activity* has a program assigned to it that is executed when the activity is executed. An activity is executed by assigning it to users who are capable of executing them. Each user has a *worklist* of activities that need to be executed. A *process activity* has another process associated to it, so an entire process is executed

when the activity is executed. Process activities are used for nesting and modular design. Each activity has an input data container and an output data container.

- **Flow of Control**: specified by *control connectors* between activities, is the order in which activities are executed.
- **Input Container**: a sequence of typed variables and structures that are used as input to the invoked application.
- **Output Container**: a sequence of typed variables and structures in which the output of the invoked application is stored.
- **Flow of Data**: specified through *data connectors* between activities, is a series of mappings between output data containers and input data containers to allow activities exchange information.
- **Conditions**, which specify the circumstances under which certain events will happen. There are three basic types of conditions. *Transition conditions* are associated with control connectors and specify whether the connector evaluates to true or false. A control connector that evaluates to false will not trigger the execution of the activity at its end. *Start conditions* specify when an activity will be started: for example, either when all incoming control connectors evaluate to true - (*and* condition) - or when one of them evaluates to true - (*or* condition). *Exit conditions* specify when an activity is considered to have terminated. After the execution of an activity the exit condition is checked. If true the activity has terminated, if false, the activity is rescheduled for execution.

An activity can be in one of the following states: *ready*, before the execution of an activity starts, *running*, during the execution of an activity, *finished* when the execution has completed, and *terminated* when execution has completed and the exit condition is satisfied. Activities can be started from the ready state either manually or automatically. Within a process, those activities without incoming control connectors are considered to be the *starting activities* of the process, and are set to the ready state when the process is started. Once an activity finishes, its exit condition is evaluated. If it is false, then the activity is reset to the ready state. Otherwise the activity is set to terminated and all the

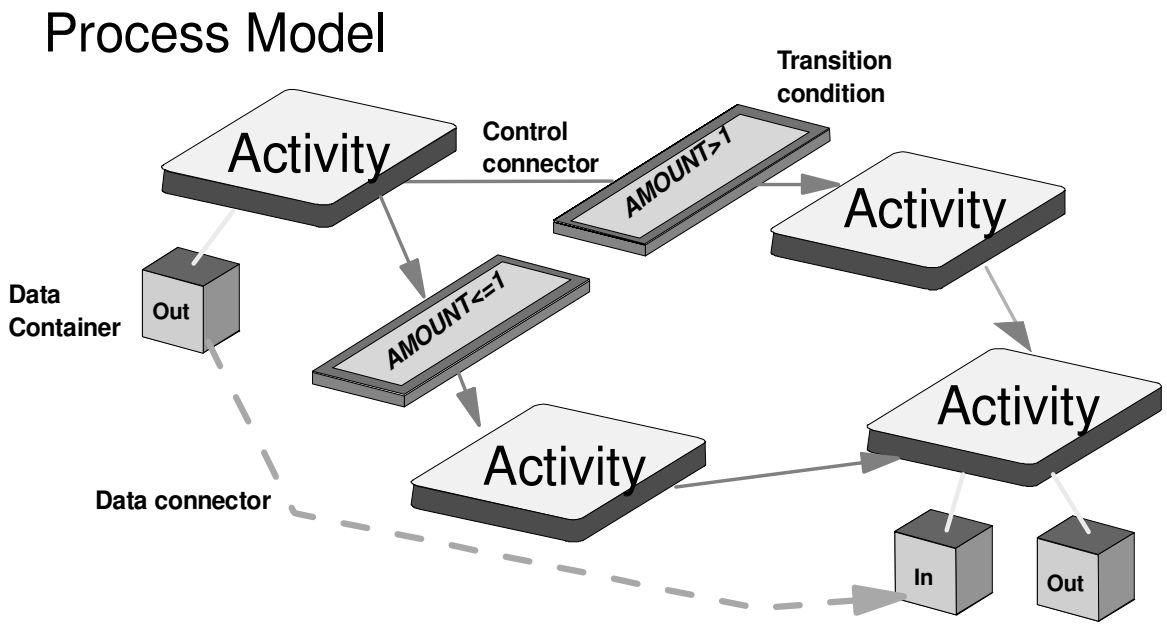

Figure 2: *Main components of FlowMark's model for control and data flow*

outgoing control connectors from that activity are evaluated. When the start condition for an activity is met, the activity is set to ready. If an activity will never be executed because its start condition evaluates to false, the activity is marked as terminated and all the outgoing control connectors from that activity are evaluated to false. This procedure is called *dead path elimination*. The process is considered finished when all its activities are in the terminated state.

A key aspect of workflow systems is the various conditions associated with connectors and activities since they are the basis for the scheduling of activities. The logic behind the business process is embedded in them. These conditions can be based on three different types of information:

- **Application Data**: which provides input related to the applications and allows to describe the flow of control in terms of the work done by those applications. Typical examples are: "salary > $50.000 AND position = permanent_employee", or "user = student OR user = faculty". It is generally provided through API calls.

- **Execution Data**: which provides information on whether activities have been successful in their execution. Note that this is different from application data. Execution data are usually return codes. For instance, in the case of transactions, whether they

are committed or aborted. In the case of programs this can be their return code indicating whether errors have occurred. This information is usually provided by the underlying system (operating system, distributed execution environment, etc.).

- **External Events**: which allow to synchronize the execution of the workflow process with the occurrence of events in the external world such as the arrival of a message, the time or date, and so forth. In general this entails some form of triggering or querying mechanism that connects the actual event with a condition in the workflow management system.

These three types of inputs are generally treated in different ways since they originate from different sources. Although it is possible to combine them within the same condition, in practice each of these three inputs will be more useful in a particular type of conditions. Application data is generally used to decide which path to take and which activities must be executed, execution data is usually used to determine which path to take (as in the case of failures) and when activities have successfully executed. External events are most commonly used to trigger the execution of a particular process or activity.

Conditions can be unevaluated, partially evaluated and evaluated. Depending on the type of information they use, these states can be temporary or permanent, therefore it is important to understand what is meant by each condition. Conditions based on external events can change their status, i.e., they are *dynamic* (something is true at a given time, but false some time later). Conditions based on application data and execution data can only go from unevaluated to partially evaluated to evaluated, i.e., they are *static*. Once they have reached the evaluated state, the evaluation can not change, they are either true or false. As a result, it is easier to deal with application and execution data not only from the workflow process design point of view, but also from the point of view of the implementation of a workflow engine. External events, depending on their nature, may require the inclusion of some temporal reasoning into the system as well as the ability to cope with changing conditions. In these cases, the semantics of the conditions are difficult to define, as an activity may be executing when the conditions that triggered its execution become false. However, external events may also be the key to workflow synchronization, as conditions that include an external event can be seen as synchronization points. Existing systems provide only a limited form of conditions. In most cases there are no external events, and very few systems allow include application data to be included as part of the flow of control.

These three types of conditions are one of the major differences between workflow management systems and transaction processing. In general, transaction processing is based solely on execution data, following the premise that the semantics and the consistency of the transaction are the programmer's concern, not the system's. This is also true of advanced transaction models [Elm92], which tend to be based on formalisms developed on execution data.

2.3 Architecture

A WFMS provides support in three functional areas: *Buildtime*, *Runtime control* and *Runtime interactions*. The Buildtime functions support the definition and modeling of workflow processes. The Runtime control functions handle the execution of a process. The Runtime interactions provide interfaces with users and applications. Of these, Buildtime and Runtime control are likely to be centralized. The former because it will be accessible only to a small set of workflow designers, the latter because it is common to all users and usually has high demands in terms of storage capacity.

Runtime control has two aspects to it: *persistent storage* and *process navigation*. Persistent storage allows the system to recover from failures without losing data and also provides the means to maintain an audit trail of the execution of processes. The navigational logic controls the execution of processes. Thus, we consider two components within runtime control, the *storage server* and the *navigation server*. These are referred to as the Workflow control data and the WFM Engine in the reference model. Similarly, runtime interactions are of two types: interactions with the users and interactions with invoked applications. The former is the interface with the end users and consist mainly of the worklist assigned to a given user. The latter is the interface to the applications being executed as part of a workflow. We consider them as separate components, the *User Interface* and the *Application Interface*. These appear in the reference model as *Worklist* and *Invoked Applications*.

2.4 Products

Workflow concepts are not new. Many of the ideas can be traced to areas like office automation, image processing or computer supported cooperative work. Nowadays there several hundred commercial products that claim to be workflow tools. Of these, only a handful are true workflow engines. It is also important to mention that there are a multitude of other

products being developed as third party applications on top of distributed platforms such as LotusNotes. Such products play a role similar to that of many third party tools used to interface with a database management system (SQL forms, for instance) and are not true workflow engines.

At the beginning of the 90's, a handful of software companies started to offer workflow products: Action Technologies, Lotus, Reach, and those in imaging systems, such as Recognition International, Sigma Imaging Systems, and FileNet, to mention a few. Nowadays there are hundreds of workflow products. To many, the most immediate ancestors of commercial workflow systems are imaging systems used for document processing. It is a natural step, after a document has been scanned and it is available in digital form, to provide tools to circulate this document to the persons for which it is relevant. One of the pioneers in this area which has also become a strong contender in the workflow arena is FileNet's WorkFlo. But there are many other influences, as proven by the Action Technologies which already in the 80's had a product called The Coordinator (the rights to this product were sold off, and it is now being commercialized by Da Vinci Corporation, which also produces Da Vinci Mail) showing many of the characteristics of a workflow management system. This makes it difficult to provide a list of products since there is a great variety of systems which, in many cases, have little in common. The following is a brief list of some of the most relevant products. Note that most of them provide a suite of components with equivalent functionality but which are not necessarily available in the same platforms as the actual workflow servers.

- **ActionWorkflow System**, of Action technologies is currently available in two versions, for Microsoft SQL server and Lotus Notes. It contains three basic components. The ActionWorkflow Management System, for integrating and controlling workflow transactions. The Analyst, a specialized editing tool to design workflow processes. And the Application Builder which translates the definition into an executable process. Additional facilities are provided by a Reporter tool that allows querying the progress and status of the workflow processes.

- **FlowMark** is IBM's leading workflow product. It runs on OS/2, Windows and AIX and it is based on ObjectStore, an object oriented database from ODI. Its main components are Servers, Buildtime Clients, Runtime Clients and Program Execution Clients. The servers provide the interaction with the databases and are in charge of the coordination of the workflow execution. The buildtime clients provide a graphical interface

for the design of workflow processes. The runtime clients provide the interface to the users through a work list, while the program execution clients provide the interface to the applications through a series of API calls and standard interfaces.

- **WorkFlo Business Systems** of FileNet runs on SunOS, UNIX, AIX, HP-UX, Macintosh and OS/2 and it is build on top of an Oracle database. It consists of a suit of products: Workforce Desktop, for Window based PCs; WorkShop, for designing interfaces; WorkFlo, which coordinates the interaction with mainframes, networks and other applications; FolderView, for less structured workflow applications; WorkFlow Application Libraries, a set of standardized APIs; and Image Management Services, for database management.

- **InConcert**, produced by XSoft, a division of Xerox Corp, runs on SunOS, AIX, DOS and HP-UX, and can use several databases: Informix OnLine, Oracle or Sybase. It provides Desktop Application, a GUI-based tool set for accessing InConcert capabilities. It is object oriented and provides several hundred application programming interfaces to ensure that almost any application can be integrated into the system. It also provides a set of reporting functions to monitor the progress of the workflow.

- **OmniDesk**, of Sigma Imaging Systems Inc., runs under OS/2 with clients under OS/2 and Windows and allows using ODBC-compliant databases. It consist of a RouteManager, for workflow management and load balancing; RouteBuilder, for defining the routing logic; and FormBuilder, to create the interfaces to the workflow. Although based on image processing ideas, OmniDesk is also suitable for workflows not based on images.

- **ProcessIT**, of AT&T Global Information Solutions (formerly NCR), is UNIX based with clients running on Windows and built on top of SQL databases. It is transaction based and consist of four products: MapBuilder, a Windows based interface to define processes; Process Activity Manager, the workflow engine; WorkView, the worklist interface; and ProcessIT's Status Monitor, used to capture the state of the system to identify bottlenecks.

- **Staffware**, of Staffware Corporation, is UNIX based with Windows clients and does not use a database but a file system. It is divided in three components: Staffware Unix

Server which runs on over 20 platforms; Staffware Windows Client; and Graphical Workflow Definer, which provides the interface for the definition of workflow processes. It uses the protections of the underlying file system to provide an added level of security.

- **Regatta**, of Fujitsu, runs under Solaris, Windows NT and SunOS, with clients in Windows and X Windows and using either SQL Server, Sybase or Oracle databases. It is based on a Visual Process Language used to create and edit processes through Graphical Planner, a GUI tool. Incremental automation is a very important aspect of this system to allow several ranges of workflow, from an improved e-mail system to fully automated processing of activities.

- **OPEN/workflow**, a WANG's product, runs under AIX or HP-UX and is based on its own database engine. The system is divided into Database Services, which provide the basic integrity, security, concurrency control, recovery and administration capabilities; Graphical Procedure Builder, a tool for process definition; Integration Toolkit, with the API calls and communication services required to interact with other applications; and Reporting Tools such as Query Builder and Report Builder to access the information about process execution.

Besides these products, there are many others that offer workflow capabilities: WIT (Application Partners), FlowPath (part of Bull's Image Works), Plexus FloWare (Recognition International), TeamFlow (ICL), ViewStar (ViewStar Corp.), and Quality at Work (Quality Decision Management). There are also a number of products that are intermediate between workflow and e-mail systems: Aster*X (Applix Inc.), BeyondMail (Beyond Inc.), WE-Mail (Professional Programming Services), and Microsoft Mail (Microsoft). One step ahead, but not yet workflow systems, are the group scheduling and group collaboration software: Synchronize (Cross Wind Technologies Inc.), AV ONGO Office (Data General Corp.), Futurus Team Windows (Futurus Corp.), Goldmine (Elan Software Group), WorkMAN (Reach Software), and WordPerfect Office (WordPerfect Corp.). In the image and document management arena some existing products are GroupFile for Windows (LaserData Inc.), Kewyfile (Keyfile Corp.), Interleaf (Interleaf), VisualInfo (IBM), and Advanced Professional System (I-Concepts Inc.).

3 Limitations of Existing Systems

The state of the art in workflow management has been determined so far by the functionality provided in commercial systems [AS96]. Paradoxically, this has been the major source of limitations. Many products were developed without a clear understanding of the user requirements and, as any serious workflow practitioner can testify, these products were quite unprepared to meet the demands placed upon them by eager users. To understand this, it is necessary to understand the background of workflow management. The direct ancestors of commercial workflow systems can be traced back to work done in areas such as office automation, image processing or computer supported cooperative work. In these environments, the main problems to solve were those of *sharing* and *cooperation* (largely still unsolved, by the way). Issues such as performance, scalability or reliability are hardly ever considered in these areas, an unfortunate characteristic inherited by workflow products. No commercial workflow products are based on OLTP (On-Line Transaction Processing) or database technology. Although many of them use databases as the underlying repository and some incorporate ideas that can be related to functionality found in commercial transaction monitors, workflow systems were not conceived to face the daunting tasks faced by very large databases or sophisticated TP-monitor installations. As a consequence, the robustness and technological maturity reached in these areas is all but lacking in workflow systems. The following are some of the most glaring limitations of existing systems.

Existing systems are almost totally incompatible. The situation is similar to that of databases before the widespread acceptance of the relational model and SQL. In spite of the efforts of the Workflow Management Coalition [wfmcM], current products incorporate in the design very concrete and exclusive interpretations of the world that make practically impossible to federate different systems. These incompatibilities are not just the syntax or the platform, but the very interpretation of workflow execution. In most cases, the system is so tied to the underlying support system that it is not feasible to extend its functionality to accommodate other workflow interpretations. Moreover, systems are too dependent on the modeling paradigm (Petri-nets, state charts, transactional dependencies, to name a few) and there is no clear understanding of the execution model of workflow processes. As a result, corporations are forced to use a unique system and to abide by its modeling idiosyncrasies.

Initially conceived as cooperative tools, workflow engines have been designed for small groups. When users, realizing the potential offered, have tried to use them in large scale en-

vironments, all the inherent restrictions in the designs have surfaced (it is not surprising that some of the major products have been recently or are currently being entirely redesigned). The architectural limitations (single database, poor communication support, lack of foresight in the designs, the problems posed by heterogeneous designs) have prevented existing systems from being able to cope with a fraction of the expected load. In the best possible scenarios, commercial systems support up to 100 users and no more than a few thousand processes running concurrently. This is far from the figures encountered in large systems (see above).

Finally, one of the major limitations of existing systems is their lack of robustness and very limited availability. The degree of resilience to failures of current systems is minimal. Current products have a single point of failure (the database) and no mechanism for backup or efficient recovery. This is not as much a flaw as a design decision, since these products were initially intended for small groups and small loads. Very large workflow management systems will involve several thousand users, hundreds of thousands of concurrently running processes and several thousand sites distributed over wide area networks. They will be critical systems and, as such, their continuous availability is crucial, in the same way that continuous availability is the key to many banking and corporate database applications. It is not reasonable to expect corporations to rely on a workflow management if a single database failure can bring the entire system to a halt. Moreover, since workflow systems will operate in large distributed and heterogeneous environments, there will be a variety of components involved in the execution of a process. Any of this components can fail and, nowadays, there is not much that can be done about it. Exiting systems lack the redundancy and flexibility necessary to replace failed components without having to interrupt the functioning of the system.

4 Enhancing Workflow Systems

In this section, we discuss research areas that can enhance workflow systems. In particular, we point out the need for a better understanding of workflow management to ensure scalability, reliability, flexibility and high availability of the workflow system itself, as well as the need for enhanced expressiveness of workflow models.

4.1 Distribution for Scalability and Reliability

A very interesting research area is that of distributed execution of workflow processes. In designing current systems, there is a trend towards client/server architectures in which a dedicated server provides most of the functionality of the system while the computing potential at the clients is barely used. There are a number of reasons for this choice: lightweight clients, centralize monitoring and auditing, simpler synchronization mechanisms, and overall design simplicity. However, as pointed out above, an architecture based on a centralized server is vulnerable to server failures and offers limited scalability due to the potential performance bottleneck caused by the centralized server. To avoid these limitations, workflow distributed architectures have started to appear (so far as research prototypes). One of the pioneers was INCAS, from Matshushita Laboratories [BMR94]. 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. The execution of a process takes place as the information carrier moves from location to location. Hewlett-Packard Laboratories has also done some work in the area in the form of a a specification language and a transactional model for organizing long running activities [DHL91]. Although intended for workflow systems, the primary emphasis is on long running activities in a transactional framework using triggers and nested transactions. This design is based on recoverable queues, as it is that of EXOTICA/FMQM, FlowMark on Message Queue Manager [AAEM96]. The goal of Exotica/FMQM was to study the effects of complete decentralization on the design of a workflow system and the feasibility of replacing a centralized database by persistent messaging. In the resulting system, each node functions independently, the only interaction between nodes being through persistent messages. The advantage of this approach is that the performance bottleneck of having to communicate with a single server during the execution of a process is avoided. Moreover, the resulting architecture is more resilient to failures since the crash of a single node does not stop the execution of all active processes.

Key to most distributed architectures, is the ability to work in a completely asynchronous environment. In general, many applications require an asynchronous communication mechanism across heterogeneous protocol independent platforms. These services are, preferably, connectionless and accessible through API calls. The most common mechanism is to provide a local queue where applications can place and retrieve messages. Once left in the local

outgoing queue, the communication system will take care of delivering it to the appropriate incoming queue in a remote machine. These queues can be made persistent so messages survive crashes, making asynchronous communication possible between applications that run at different points in time. Additionally, interactions with the queue are transactional, which provides greater flexibility when dealing with failures.

Using these queues, each node executes its part of a workflow process. Once it terminates all the activities that are to be executed in that node, all the relevant information is placed in a queue to be sent to the next node which will execute the next part of the process. This simple idea raises some interesting research issues that are still open. For instance, with a centralized server each process has an owner, who can start process instances, abort their execution, and is notified of their termination. When the process is distributed among many nodes, the notion of a process owner is much less obvious. Similarly, detecting when a process terminates may not be easy since no node is aware of the entire process. Another interesting aspect of the distributed architecture is the management of worklists. Worklists are lists of workitems belonging to one user. In a centralized system, this is very easy to maintain since users only need to logon to the central server to retrieve their worklist. Once retrieved, the server can update it by sending the updates to the runtime client where the user is currently logged on. Moreover, an activity may appear on several worklists simultaneously, but only one user will be allowed to execute it. The synchronization problem of ensuring that only one user actually executes the activity is solved by having the server select the user who contacts it first. These two features are complex to implement in a fully distributed environment since activities are associated with nodes and queueing systems do not provide the appropriate primitives (namely, the ability to retrieve messages from a remote queue). Finally, monitoring and logging is slightly more involved than in a centralized system. Once these problems have been solved, distributed architectures will probably become more common, specially in combination with technology such as CORBA or Java.

4.2 Mobility for Flexible Interaction

Business process re-engineering, along with its technological counterpart, workflow management systems, offer great opportunities to improve the efficiency of an organization by taking over the more menial tasks of coordination and monitoring. At the same time, *disconnected operation*, identified as one of the main ways in which computers will be used in the future,

is also appearing as a way to solve key problems in todays' organizations. With mechanisms to support disconnected operation, users within an organization can work independently of the main computer facility. It seems obvious to try to combine both trends, workflow and disconnected operation: users can work from a remote location and the coordination is performed by the workflow system. However, disconnected computing and workflow management systems have contradictory goals. A workflow management system is a tool for cooperation and collaborative work that requires constant monitoring . On the other hand, disconnected computing is geared towards supporting users working in isolation from others. The question is how to allow cooperation while still respecting the autonomy of the disconnected clients.

Some work has been done in this area and there are some promising approaches like using the *world wide web* as the user interface. So far, and to our knowledge, there are no commercial products that support disconnected operation. An interesting solution among those proposed so far [AGKA96], relies on giving enough autonomy to the clients to allow them to perform work without having to be connected to the rest of the system while still maintain the overall correctness and consistency of the processes being executed. The gap between disconnection and coordination is closed by establishing a set of basic rules for both worlds: users must "commit" themselves to perform certain tasks before disconnecting from the system and the system guarantees that there are no synchronization problems with other users.

The entire sequence of operations involved in the execution of every activity within a process in a static WFMS is based on the fact that all components are always connected to the server and, therefore, to the database, which simplifies synchronization and the design of the clients. This permanent connection is used to monitor the progress of the activity, to provide feedback to the user, to allow external applications to access data from the workflow system and so forth. Hence, support for disconnected operation can be provided in two ways, one is to have the clients working in a "batch" mode, where a set of activities is assigned to them and all the relevant information is downloaded to the clients prior to their disconnection so there is no need to contact the workflow server. The other is to allow the clients to perform navigation themselves by transferring entire parts of a process to the clients, effectively duplicating at the clients much of the functionality of the servers. This turns out to add significant overhead and, therefore, the "batch" mode seems a more viable

option. This mode is now discussed in more detail.

During disconnected operation we will assume that both the worklist and the applications interface are local, while all other components are remote. Worklists are usually a mere interfaces for the user to specify actions such as *start activity*. Consequently, their role does not change much in disconnected mode except for the fact that instead of sending the commands to the server, these are now sent to the application interface. The applications interface, on the other hand, acts according to the messages received from the worklist as opposed to reacting to the messages sent from the server. Since it cannot connect to the database to provide additional information requested by the application through API calls, it must also provide its own persistent storage for the information that may be requested by the application. Similarly, it must also persistently store the results of the application's execution until they can be sent to the server. These steps are organized in three phases. The first is a *synchronization* phase in which, prior to disconnection, a user declares the intention to reserve an activity for execution during disconnection. If it is an activity that can be executed by several users, then the other users are notified that they are no longer eligible to execute the activity. This phase also involves transferring all the information pertaining to the activity from the server to the program execution client. The second phase is the disconnected operation *per se*, in which the user works on the reserved activities without any control from the server. The third phase is the *reconnection* to the server, in which the worklist of the user is updated, and the results of the executions of the activities are reported back to the server for storage in the database.

The key aspects of disconnected operation are the *locking* and *preloading* of activities that will be available at the client while being disconnected. Locking is necessary due to the fact that the same activities may appear in several worklists simultaneously. Under normal circumstances, the centralized database serializes all changes to an activity and, hence, even if two users attempt to start the same activity concurrently, only one of them will be able to register in the database as the user to which the activity has been assigned. To prevent other users from working concurrently on the same activity, before a user can disconnect from the server, all activities they intend to work on must be *locked* by the user. When a user locks an activity, this implies an explicit commitment to work on that activity, regardless of whether the user works on the activities while connected to the server or disconnected from it. A locked activity is permanently assigned to a user until the user completes it or unlocks it.

During disconnected operation only locked activities will appear in the worklist of the user. Similarly, the locking of an activity signals the server to retrieve all the information pertaining to the execution of the activity and to send this information to the client to store for use during disconnection. This is the step of preloading the activity. Of course, both operations are geared towards maintaining the "look and feel" of the interface. From the user's point of view there should not be any difference between normal and disconnected operation, beyond the limitation that during disconnected operation the worklist contains only locked activities. It must be pointed out, however, that there are many trade-offs to consider, specially in the case of portable computers. A database can add significant overhead in terms of the footprint of the program execution client. On the other hand, if many activities are locked simultaneously, some form of indexing and organized data repository needs to be provided to guarantee fast access to the locked activities. All this parameters need to be kept in mind to design a client with a reduced footprint.

Disconnected and mobile operations are rapidly gaining in importance. As WFMS are also more prevalently deployed in various organizations, they must support disconnected operations. The Exotica approach is promising and leaves many implementation issues open. We believe this to be an important area for future research.

4.3 Transactions for Enhanced Expressiveness

It is a generally acknowledged that traditional databases are not capable of supporting a variety of applications. To extend their functionality, several advanced transaction models [Elm92] exist but very few have ever been used in commercial products. One of the reasons for such a limited success is the inadequacy of advanced transaction models. Advanced transaction models are too centered on database concepts, which limits their possibilities and scope as many computer tools are not transactional. It has also been pointed out that, since they tend to remain theoretical models, a number of important design issues are yet to be resolved. Interestingly enough, there is an important demand for tools to support applications very similar in nature to those envisioned by the designers of advanced transaction models. Workflow systems are one of the by-products of this demand. In fact, *Workflow Management Systems*, WFMSs, bear a strong resemblance to advanced transaction models, although addressing a much different and often richer set of requirements.

Transaction models have a significant number of advantages. Among them the use of

the ACID properties (Atomicity, Consistency, Isolation and Durability), which advanced transaction models have tried to relax to adapt them to more sophisticated applications. For instance, to relax the notion of atomicity is important to avoid the blocking phenomenon typical of standard atomicity. But even when non-transactional units of work are considered there is always the notion that a collection of activities must successfully terminate. In this context, the concept of relaxed atomicity acquires a new and rich meaning, since "successful termination" can have multiple interpretations and, in general, will be embedded within the semantics of the activities. Hence, it is important to have a framework such as that of advanced transaction models to reason about the order of execution, data dependencies, subtransaction characteristics and alternative executions. On the other hand, we believe that only by addressing the requirements of real applications such as those of workflow environments, i.e., being interpreted in a much broader context, will these models reach their technological maturity.

Workflow systems, for their part, are learning some of the lessons taught by transaction models the hard way. In spite of the complex environments they target, few or none of the current products have incorporated transactional concepts such as atomicity, isolation, or alternative execution. It yet remains to be seen which of these concepts are useful in these environments, a topic hotly debated by researchers, but there are undoubtedly many ideas from the transactional world that can be translated and successfully applied in a workflow environment. As an example, recent work [AAEK96] has shown how to incorporate the notion of relaxed atomicity into a workflow specification. This has been done by implementing flexible transactions on top of a workflow system. Flexible transactions provide the means to specify alternative execution paths in the case of failures while still preserving the overall atomicity, a very desirable property required to provide adequate exception handling capabilities. This is a first step in the cross-fertilization between advanced transaction models and workflow environments, but additional research is needed to formalize workflow specifications and identify transactional concepts of value in these environments.

4.4 Replication for Interoperation and Availability

One of the key aspects of WFMSs is their availability. If a company is to rely on a WFMS to coordinate and monitor its business processes, it must be first convinced of its high availability. It is not difficult to imagine environments where one cannot afford to stop

ongoing business processes because of system failures (or system updates, administration, configuration changes, etc.). This is especially true of installations with a large number of process instances running simultaneously, where any down-time introduces significant delays. In spite of its importance, availability of WFMSs is a topic that has been largely neglected by commercial systems and only recently has been addressed by the research community [KAGM96].

Most existing systems are built on top a centralized database that acts as a single point of failure: when the database fails no process can continue executing. Even if several databases are used to minimize the impact of failures (by running different processes off different databases) existing designs will stop executing all the processes associated with the failed database. It can be argued, however, that availability is a known problem that has been solved in databases using different techniques. Since WFMSs are built on top of databases, it should be possible to apply these techniques to the underlying database to provide higher availability. The most common technique to provide high availability is replication, by which a mirror system is kept synchronized with the main system. When the main system fails the mirror takes over. If the mirror is an exact replica of the main system (all updates to the main are also performed at the mirror), the technique is known as *hot standby*. This usually requires a Two Phase Commit protocol between the main and the mirror, but it allows the mirror to take over almost immediately in the event of a failure. The cost can be reduced by allowing the mirror to stay slightly out-of-date instead of completely synchronized. It is also possible for the mirror to provide *cold standby* by just storing the updates, without applying them, until the moment in which it actually has to take over. There are, however, some differences between databases and workflow environments . First, databases assume that the primary and the backup are the same database. This would tie a WFMS to the platforms where the database runs. Second, database backups are managed at a very low level (pages or log records, for instance) and replication takes place regardless of the semantics of the application. In a WFMS it is possible to use the application semantics to optimize the replication by only maintaining copies of those events that are relevant to the overall execution.

To address these issues, current proposals [KAGM96] have suggested an approach in which there is no dedicated backup and different processes can have different guarantees. The reason not to have a dedicated backup is that the distributed and heterogeneous char-

acteristics of the architecture would require either a backup for every individual system or a single remote backup for the entire system, which is distributed over a wide area network. Such an approach would incur in too high a cost and would need to cope with the heterogeneity of the primary databases. Instead, databases are used both as primaries and backups. For some servers the database acts as the primary, for others it acts as a backup. This increases the load at the database but is a feasible solution. In part to reduce the overhead at the backup, in part to accommodate the many different requirements of workflow applications, processes are organized according to three categories. *Critical* processes are those for which execution must be immediately resumed in case of failures. Hence, they are replicated using a hot standby approach. All changes performed at the primary are forwarded to the secondary where they are immediately applied. Both transactions, at the primary and the backup, are committed using 2PC. *Important processes* are those which should be eventually resumed in the event of failures, but some delay is acceptable. This allows to minimize the impact on performance as 2PC is no longer necessary and the backup does not perform any updates, it simply stores the changes in case they are necessary to restore the process state. When a failure occurs, all the stored changes need to be applied at the backup before execution can be resumed. Finally, *normal* processes rely only on forward recovery to deal with failures. They are not replicated at all and the only guarantee is that, in case of failures, once the failure is repaired, execution will be resumed where it was left. To assign a process to one of these categories is left to the designer of the workflow.

The most interesting aspects of these backup schema is the fact that it is based on the application semantics and that it can be performed over heterogeneous databases. In a heterogeneous database, a data mapping mechanism is used so information from a database can be used in another. This data mapping uses a canonical representation based on the workflow specification so inter-database communication takes place at the level of workflow concepts (activities, processes, data containers, control connectors, etc.). This same canonical representation is used to avoid the problem of having to deal with internal database representations. Low level items such as objects, tuples, attributes or pages are not replicated, rather the state of workflow entities (activity x has started, process y has terminated, etc.). Since the number of entities is very small, the mapping is not complicated and does not add a significant overhead.

The issues of replication over different database systems as well as large scale distribution

of WFMSs in general are closely related to the bigger problem of interoperability across heterogeneous WFMS. We believe that the development of a canonical representation along with a modelling standard as proposed by the Workflow Coalition [wfmcM] could be the basis for interoperability across heterogeneous workflow systems. This would facilitate both the scalability and the incorporation of various fault-tolerant levels in WFMSs.

5 Conclusions

In this paper we presented a brief description of the state of the art in workflow systems. An analysis of current commercial WFMSs lead us to conclude that current systems are inflexible, lack any standardization across products and do not handle failures in large distributed systems. As part of the Exotica project, we explored many of these problems and proposed several solutions. In particular, we proposed the use of message queues for fault-tolerant reliable communication, a mechanism for supporting disconnected operations and how to incorporate replication to improve availability. From a modeling point of view, workflow systems provide an interesting alternative to current attempts in relaxing the standard transaction management properties. In fact, we were able to demonstrate that a workflow system can be easily used to implement various advance transaction models. We believe that these research and development is needed towards building scalable and reliable distributed workflow management systems.

Acknowledgements

Part of this work has been done in the context of the Exotica project. This project started in 1994, at IBM Almaden Research Center and with funding from IBM Hursley (Networking Software Division) and IBM Vienna (Software Solutions Division). A. El Abbadi and D. Agrawal participated in the project while on a sabbatical visit to IBM Almaden. G. Alonso worked on the project as a visiting scientist. We are grateful to R. Günthör and M. Kamath for their help in formulating some of the ideas presented in this paper. Even though we refer to specific IBM products in this paper, no conclusions should be drawn about future IBM product plans based on this paper's contents. The opinions expressed here are our own.

Useful pointers

The following are some URLs where additional information and further references can be found regarding workflow management systems:
http://optimus.cs.uga.edu:5080/activities/NSF-workflow/

http://www.do.isst.fhg.de/workflow/pages/Workflow_Index_Englisch.html
http://www.ifi.unizh.ch/groups/dbtg/Workflow/workflow_sites.html
http://wwwis.cs.utwente.nl:8080/~joosten/workflow.html
http://www.almaden.ibm.com/cs/exotica/

References

[AAEK96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, C. Mohan. *Advanced Transaction Models in Workflow Contexts*, In Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA Feb. 26 - March 1, 1996.

[AGKA96] G.Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, C. Mohan. *Exotica/FMDC: A Workflow Management System for Mobile and Disconnected Clients*, International Journal of Distributed and Parallel Databases (to appear).

[AAEM96] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Günthör, M. Kamath. *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management*, Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organizations. Trondheim, Norway, August, 1995.

[AS96] G. Alonso, H.-J. Schek. *Database Technology in Workflow Environments*, INFORMATIK-INFORMATIQUE (Journal of the Swiss Computer Science Society), April, 1996.

[BMR94] Barbara, D., Mehrota, S., and Rusinkiewicz, M. (1994). INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments. Technical report, Matsushita Information Technology Laboratory.

[DHL91] 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, August 1991.

[Elm92] A.K. Elmagarmid (ed.) Transaction Models for Advanced Database Applications Morgan-Kaufmann, 1992

[wfmcM] D. Hollinsworth. *The Workflow Reference Model*, Workflow Management Coalition, TC00-1003, December 1994.

[Hsu93] M. Hsu. *Special Issues on Workflow and Extended Transaction Systems*, Bulletin of the IEEE Technical Committee on Data Engineering vol. 16, no. 2, June 1993; and vol. 18, no. 1, March 1995.

[KAGM96] M. Kamath, G. Alonso, R. Günthör, C. Mohan. *Providing High Availability in Very Large Worklflow Management Systems*, In Proceedings of the Fifth International Conference on Extending Database Technology (EDBT'96), Avignon, France, March 25-29, 1996.