1

WORKFLOW MANAGEMENT SYSTEMS: THE NEXT GENERATION OF DISTRIBUTED PROCESSING TOOLS

G. Alonso* and C. Mohan**

**Institute of Information Systems, ETH Zürich (IFW C 47.2), Zürich 8092, Switzerland*

***IBM Almaden Research Center, 650 Harry Road (K55-B1), San Jose, California 95120-6099, USA*

ABSTRACT

Workflow management systems have attracted a great deal of attention due to their ability to integrate heterogeneous, distributed applications into coherent business processing environments. In spite of their limitations, existing products are enjoying considerable success since they are the first practical implementation of functionality and concepts studied for many years. The goals may have changed from office automation and computer supported cooperative work to business processes and re-engineering, but the basic ideas and concepts have remained the same. It would be a mistake, however, not to try to see beyond current systems and applications. In today's computer environments, the trend towards using many small computers instead of a few big ones has revived the old dream of distributed computing. But there is a significant lack of tools for implementing, operating and maintaining such systems. Most existing solutions still belong to the mainframe world and are only slowly making their way to PCs and workstations. TP-monitors offer a perfect example of this trend. Not so long ago, TP-monitors were only used in large installations. Now they are becoming the basic glue for any practical system. Workflow management systems, on the other hand, do not have such a long history behind but, like TP-monitors, they do provide functionality that is essential in a distributed environment. It is precisely because of this functionality that the concepts and ideas associated with workflow management will play a crucial role in the future. This chapter describes in detail such functionality and provides some insight on how it can be used in environments other than business processing.

1 INTRODUCTION

One of the basic platforms in which to implement generic distributed systems is commodity hardware and software such as clusters of personal computers and workstations connected via a network. The continuous increase in computing power, storage capacity and communication speed have turned such configurations into viable and cost effective alternative architectures. As a matter of fact, much of the necessary infrastructure is already in place both in terms of hardware (clusters of personal computers connected by a Local Area Network) and software (the many existing applications). The only component missing is the necessary glue to make a coherent whole out of many autonomous, heterogeneous building blocks running over distributed, loosely coupled systems. Federated database systems [47], TP-monitors [26, 45], persistent queuing [6, 43], standards like CORBA [46], process centered software engineering [33] and workflow management systems [31] are among the best known examples.

The functionality needed in such environments can be roughly divided in four obvious categories: *interface definition*, *communication*, *execution guarantees*, and *development environment*. Existing systems usually aim to provide functionality in only one of the categories: TP-monitors for execution guarantees; CORBA as an interface definition; queuing systems as communication platforms; and workflow systems for developing distributed applications. Because of this narrow focus in only one category, related products are generally available as a stand-alone systems. Users or designers interested in getting two or more of the four categories have to resort to combine several heavy-weight solutions, which adversely affects performance and usability. Since the four categories are needed in any practical system, a more rational approach would be to build systems that incorporate the four categories of functionality in their initial design and avoid the redundancy present in today's solutions. The "transactional services" described by the CORBA standard are an excellent example of this. The most reasonable implementation for these services is a TP-monitor. TP-monitors, however, provide a great deal of services that are already implemented in CORBA environments since they are basic services of any distributed application. The result of joining the two systems is that a great deal of unnecessary redundancy is introduced. The number of such examples is limited only by the number of possible combinations between these systems. Most of these combinations are currently being implemented in one form or another.

The role workflow management will play in future computing environments immediately follows from this idea. One of the factors that have made workflow management so successful is the support they provide for developing complex applications over distributed systems using already existing tools. This same concept can be generalized in future systems, turning workflow management into one of the basic technologies for developing large scale distributed applications based on autonomous components. It is likely that workflow management will not evolve in the form it has taken in current workflow products. Instead, it will become one more component of larger, tightly integrated architectures. In order for this to happen, workflow management needs to be reinterpreted from a perspective going beyond current products and business processes reengineering. One way to do this is to consider workflow management as a very high level programming language linking, within a single control logic, heterogeneous applications residing over a wide geographic area. This direction has already been hinted at by several workflow designers [17, 37] for whom a workflow system is essentially a tool for programming at a very coarse granularity. With such an approach, the basic building blocks are entire applications like databases, spreadsheets, text editors, or scientific tools; workflow management is the programming tool to build distributed applications using these building blocks in a transparent manner. The goal of this chapter is to describe this idea in more detail emphasizing the functionality of workflow management that is of interest in generic applications. Thus, the infrastructure provided to support distributed execution of complex processes is discussed in Section 2. Section 3 presents the basic features of current systems as well as their limitations, suggesting several ways to address them. Section 4 concludes with an overview of areas other than business processing where workflow technology could be applied.

2 WORKFLOW MANAGEMENT

Workflow management is a relatively new term. The ideas and concepts associated with it, however, have been around for quite some time. Workflow functionality can be found in many early systems in the areas of office automation, transaction processing, document routing and image management. But the technology to develop full functional systems has become available only in the last few years. To certain extent, workflow management has found its window of opportunity in this decade thanks to organizational management trends such as business process reengineering [29]. As a result, it is uncommon to find a product that it is not directly associated with the reengineering

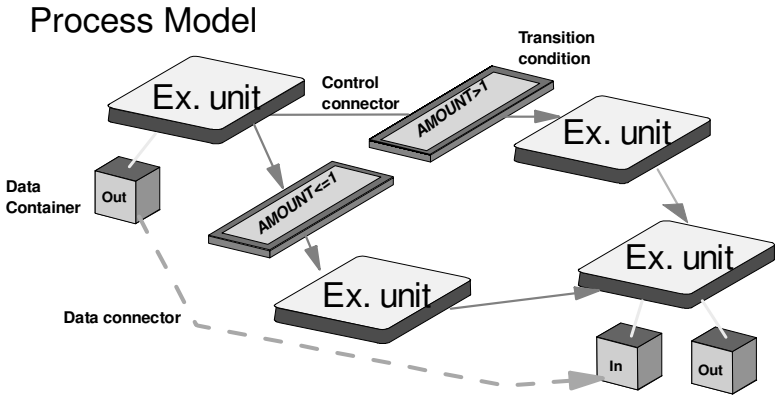

Figure 1 *Basic components of a workflow process*

world. But this is likely to change in the future as workflow systems diversify and incorporate ideas from other areas. This section discusses the main ideas behind workflow management, without tying them to business processes, as a first step towards generalizing their applicability to distributed environments.

2.1 Language

As the representation of an abstract procedure, a workflow process can be seen as an annotated directed graph in which nodes represent steps of execution, edges represent the flow of control and data among the different steps, and the annotations capture the execution logic. Other forms of representation are possible, based on rules [33], for instance, but the basic concepts are essentially the same regardless of the representation. This chapter follows the annotated directed graph approach used in most commercial products. Thus, the workflow language can be seen as a meta-programming language in which the basic instructions are procedure calls. Its main elements (shown in Figure 1) are:

Execution unit is the basic instruction of the workflow language. It can be compared with a procedure call in a programming language. Similarly to procedure calls, it can correspond to an internally defined procedure (a process), to a structured block of instructions (a block), or to a remote procedure call to an external application (an activity). Associated with each execution unit there is an input and an output data container used to store the inputs and outputs of the execution unit. A state is associated with each execution unit,

as well as two conditions, one to determine when the execution unit can start and another to determine when it has been completed successfully.

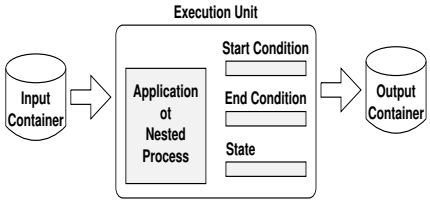

Figure 2 *The execution unit as the basic building block of a workflow model*

Process is the equivalent of a program. It specifies the execution logic by linking execution units via control and data connectors. To allow nesting, a process can be represented as an execution unit, in which case it becomes one more step within another process. The possible states of a process are shown in Figure 3.

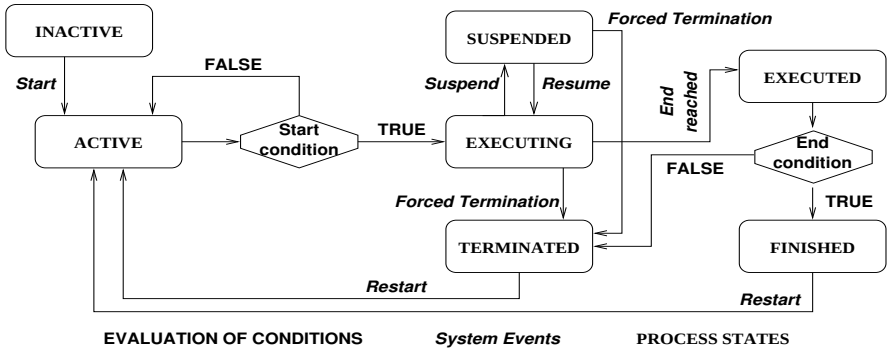

Figure 3 *State diagram of a process*

Blocks allow the modular descomposition of a process very much like in structured programming. A block is equivalent to a series of execution units bracketed by a *BEGIN ... END*. It is essentially another process except that it has no name and can not be reused. Contrary to sub-processes, which are bound to the parent process at run time, blocks are instantiated at compilation time. It is possible to associate certain semantics with blocks to denote specialized types of structures such as loops, case statements, and fork or parallel-do operations.

Activities correspond to the invocation of external applications. Processes and blocks are structuring constructs that have no effect outside the work-

flow system. Activities correspond instead to interactions with the external world. They can be *manual* if they require human intervention to be started, or *automatic* if they can be started without human intervention. In general, manual activities correspond to activities that require user involvement to be completed (filling a form, providing some information, making a decision). Automatic activities, on the other hand, usually do not require user participation (transactions over a database, index calculations, statistical calculations, etc.). Associated with each activity there is an application and a set of eligible users indicating which application is to be invoked and the users allowed to execute it. Figure 4 shows the possible states of a manual activity (automatic activities have a similar but slightly simpler state graph).

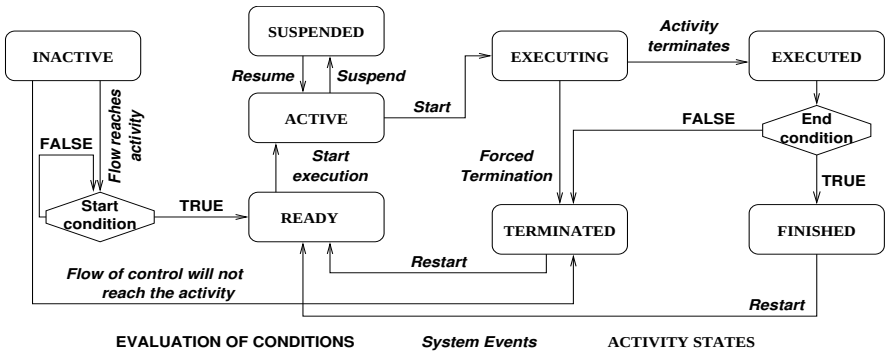

Figure 4 *State diagram of an activity*

Data containers provide a persistent repository for the input parameters and output of an execution unit. In the case of processes, the input data container collects input parameters for the entire process. When the process starts to be executed, these input parameters are distributed among the input containers of the execution units within the process. As these execution units terminate, their outputs are transferred from their own output data containers to the output container of the process. For activities, the input data container stores the parameters to use when invoking the application and the output data container stores the application's return values.

Data connectors are used to specify data flow between execution units. For instance, the input data container of a process is mapped to the different input data containers of the execution units within the process by indicating via data connectors which variable in the process container corresponds to which variable in an execution unit's container. The same mechanism is used to pass the results produced by an activity as inputs to another activity. Together, data containers

and data connectors, eliminate the need for global variables and allow each execution unit to define its own parameters. The use of data connectors forces the workflow programmer to explicitly state the data flow within the process and helps to optimize data migration in applications distributed over a wide geographic area.

Control connectors indicate the flow of control among execution units. In general, control connectors can only be used between execution units at the same level of nesting, which strengthes the modularity of the language. That is, it is not possible to add a control connector between activities of two different blocks, or between an activity external to a process and an activity within the process. Each control connector has a condition attached to it, which is used to determine when the control connector is to be followed.

Conditions are boolean expressions over data in the data containers. They indicate when certain actions should take place. In the case of execution units, there are two types of conditions to be considered: *start* and *end* conditions. The former specifies when an execution unit can start to execute (the exact meaning varies depending on whether the execution unit is a block, a process or an activity). The latter is used to determine when an execution unit has terminated successfully, usually by checking the return code provided in the corresponding output data container. In the case of control connectors, conditions indicate whether the connector should be followed or not. If the condition of a connector is evaluated to true, the execution unit at its end is taken out of the inactive state (the exact action depends on the nature of the execution unit). If the condition associated to a connector evaluates to false, it indicates that the connector will not be followed. Marking a control connector as false triggers the procedure of *dead path elimination* which marks off all connectors and execution units that will never be executed. This helps to determine when a process has terminated its execution.

Applications represent the external programs to be invoked as part of the execution of an activity. Applications are registered with the workflow system very much like applications being installed in an operating system. The registration process allows the workflow system to establish in which network addresses a given application can be found, access permissions associated with it, under which operating system it runs, associated paths, input parameters, and any other additional information necessary to invoke the application remotely. Once registered, applications are invoked by linking them to activities.

Staff represents users and sets of users. Similarly to applications, users must be registered with the workflow system. Users must be registered individually

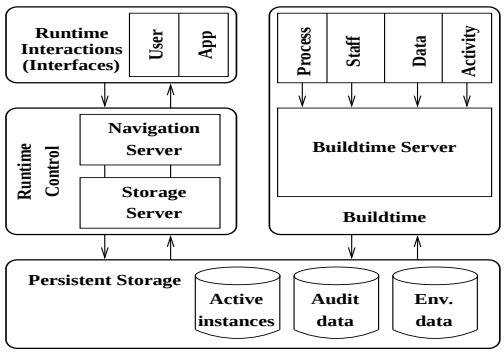

Figure 5 *Main components of a workflow management system*

and later on they can be grouped into more meaningful sets, usually known as *roles*. Roles allow the system to refer to groups (programmers, managers, engineers, sales representatives) when allocating work instead of having to deal with individual users. When an activity or a process is defined, part of the information specified is the users or group of users that are eligible to execute the activity or to start the process.

2.2 Architecture

Architectural details vary from product to product and are evolving very quickly as products try to cope with more demanding environments. It is possible, however, to distinguish a set of features common to most systems by looking at the functionality that needs to be provided.

Functional Description

The basic functionality of a workflow system can be divided in three major areas: *design and development*, *execution environment*, and *interfaces*. Usually, these three areas are also referred to as *Buildtime*, *Runtime control* and *Runtime interactions* respectively [30, 55].

For design and development, workflow systems provide a language along the lines described above as well as several tools to register users and applications. Programming, i.e., designing, a workflow process is usually done through a graphical interface in which execution units are represented as a variety of

selectable icons and connectors as directed links between these icons. This approach is perhaps the most user friendly but it has several drawbacks, the main one being that it becomes rather cumbersome to visualize and manipulate large and complex processes. Current systems usually provide a more textual language in which to specify processes but, in most cases, these languages are not adequate for large scale programming. It is likely that, in the future, more sophisticated languages will be supported. Additional tools are also provided for debugging and compiling the process description into object code that can be used for execution. Current systems provide only a primitive development environment but, given the key role it plays, it is likely that the buildtime component of future systems will be significantly enhanced [37, 48].

The execution environment can be divided in two parts: *persistent storage* and *process navigation*. Persistent storage provides a repository where all the necessary information about the system can be kept and retrieved at any time. Persistent storage is managed via a *storage server*. Since the information involved is often complex and it is necessary to support complex queries over it, most systems use a database management system for this purpose. The advantage of relying on persistent storage is that it makes possible to recover from failures without losing data (forward recovery) and also provides the means to maintain a record of the execution of processes. These two features open up many interesting possibilities when programming distributed applications. For instance, the fact that the execution is persistent implies that failures will not require to repeat the entire process, execution can be resumed from the point where it was left when the failure occurred. It is possible to subdivide the persistent storage in several areas according to the data stored: *audit trail*, *active instances*, and *environment information*. The audit trail contains information about already executed processes. In business environments this provides the information necessary to evaluate the organization's performance, system evolution, potential bottlenecks as well as supporting data mining and analysis techniques. Active instances correspond to the persistent state of processes being executed, which can be queried through monitoring tools provided by the user interface. The environment information corresponds to the staff and applications. It is used to locate applications and to determine the invocation method as well as to locate users and to determine their access rights. Process navigation is performed by the *navigation server* or *WFM Engine*. It mainly consists of evaluating the conditions specified for activities and control connectors, activating or deactivating control connectors and triggering status changes in execution units according to the events taking place in the system. Usually, all these operations are performed as transactions over the underlying storage server.

Finally, a workflow system supports two types of interfaces: users and application interfaces. Users interact with the workflow system through a *worklist* which acts as a repository for all the activities assigned to the user. This interface can be as simple as a list of manual activities waiting to be selected by the user or as sophisticated as a dynamic interface to the audit trail for querying information regarding already executed processes. The worklist is created when the user logs-in and updated every time a new activity becomes ready for execution (updates are sent using the environment information, which is also kept up-to-date regarding which users are connected to the system at any given time and from which location). Applications are handled on a location basis. Users will usually connect to the system from a PC or workstation. These locations will have an application interface so applications can be started when users decide to execute an activity. But it is also possible to have application interfaces in locations where no users are connected. This allows, for instance, to connect to mainframes, specialized workstations or execute automatic activities across wide area networks. Which type of connections are allowed and supported depends largely on the intended use of the product, i.e., whether it is a collaboration tool to be used in a LAN environment or a production tool to be used in conjunction with OLTP (On Line Transaction Processing) and OLAP (On Line Analytical Processing) systems.

Runtime Architecture

Current workflow management systems serve as platforms for executing distributed applications designed according to business rules. The same functionality they provide for business processes can be used in generic distributed applications. Thus, very much like in the case of TP-monitors [26], workflow systems are slowly evolving towards specialized, multi-platform distributed operating systems. As a generic example of existing architectures, Figure 6 shows the architecture of FlowMark [32, 37].

Most workflow systems are built on top of a database management system. In the case of FlowMark, the database is Object Store (represented in Figure 6 as *OSS* and *DB* which together act as the storage server). Most other systems are based on relational databases, for instance: ActionWorkflow is based on Microsoft SQL Server, WorkFlo of FileNet uses Oracle, and InConcert of XSoft can use Informix, Oracle or Sybase engines [48, 50]. The navigation server, represented in Figure 6 by the *FMS* component, is usually implemented as a client of the database since most navigation steps involve getting information in and out of the database.

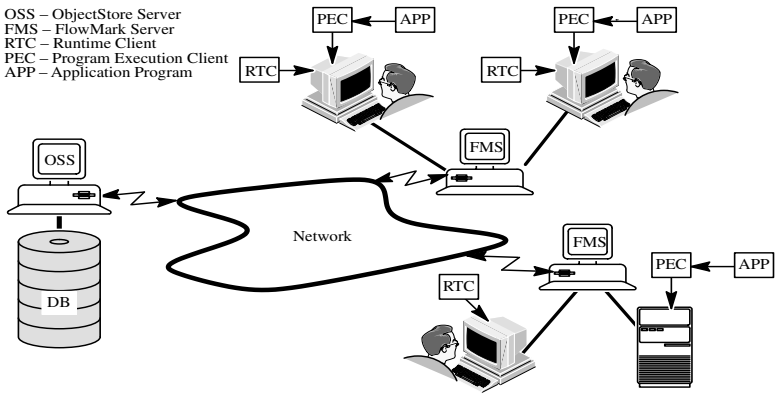

Figure 6 *Run time architecture of IBM FlowMark*

The rest of the system components used during the execution of a process are connected to the navigation servers, which can also be connected among themselves [8]. These connections do not need to be over a LAN, they can also take place through a WAN or even from mobile clients [7]. A common configuration is to have the application and user interface in the same location where the user accesses the system. This allows the user both to access the corresponding worklist and to execute activities locally (which, of course, also requires to have the application locally installed). In Figure 6 this is represented by the Runtime Client (*RTC*), the Program Execution Client (*PEC*), and the application (*APP*). These correspond to the user interface, application interface and application being invoked respectively. It is also possible to configure nodes to host only one application interface and specialized applications. Such configuration plays an important role when automatic activities are involved, for instance, when a series of transactions are executed over a database server.

2.3 Process Execution

The way execution proceeds in a workflow system is best illustrated with an example (this example follows the architecture and runtime interactions of FlowMark) [7]. An execution unit becomes ready for execution as a result of a navigation step. In the case of processes, when they reach the "active" state all of their starting activities are set to ready and any necessary input data transferred to the corresponding input data containers. In the case of activities, when they reach the "ready" state, the navigator performs role and staff resolution

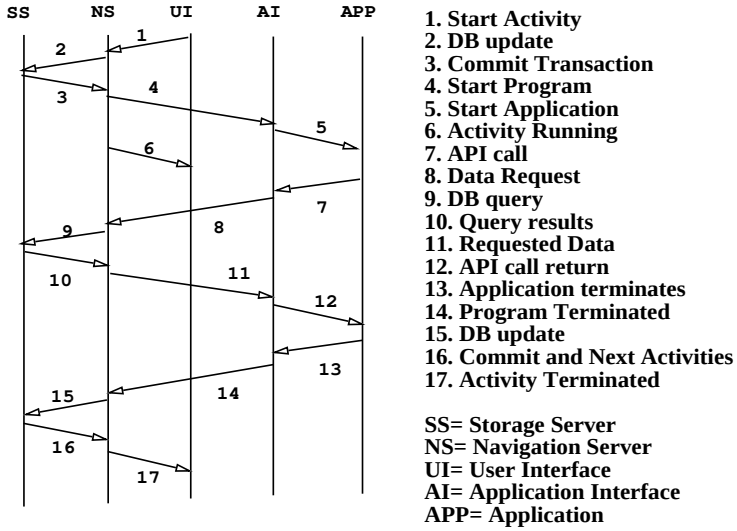

Figure 7 *Steps involved in the execution of an activity*

to determine all the users who are eligible to execute the activity and updates the worklists of all these users by including the activity as a new workitem. If the activity is an automatic activity, then it immediately changes to the "active" state during which the navigator locates a node where the activity can be executed. When the corresponding application is invoked, the activity then switches to "executing". Manual activities, on the other hand, must wait until a user selects the activity for execution. In this case, the exchange of messages between the different components is shown in Figure 7.

Manual activities appear in the worklist of all users eligible to execute it. When a user selects the activity, the user interface sends a *start activity* message to the navigator. The navigator reacts to this message by taking several steps. First, the activity is deleted from the worklists of all other users by sending a message to these worklists indicating that the activity is no longer available. Second, a transaction is started over the storage server to retrieve the information related to the corresponding application. This information allows the navigator to determine which application interface will be responsible for executing the activity. It is possible for an application to reside in many locations. If it requires interaction with the user, the application is usually invoked at the user's location, otherwise simple heuristics can be used to select the most appropriate location (load balancing, overhead, pre-established priorities, etc.). Once the

application interface has been selected, a *start program* message is sent to it. As the third and final step, the navigator sends an *activity running* message to the user interface from where the activity was selected so the status of the activity can be updated and the progress of its execution monitored from the user interface.

Any communication between the application and the workflow system takes place through API calls to the application interface. Application interfaces are multi-threaded so as to be able to cope with several applications being executed simultaneously at the same location. Thus, upon receiving a *start program* message, the application interface spawns a thread for the particular application and this thread will start the application. Any initial parameters to be passed to the application when it is invoked are sent to the application interface along with the *start program* message. The application may, however, request additional information from its input data container by issuing API calls to the application interface. These calls are received by the application interface which will forward a *data request* message to the navigator. The navigator, upon receiving such request, executes a transaction over the storage server and forwards the *requested data* to the application interface. The application interface then completes the API call by returning the data to the application. When the application terminates, the application interface sends a *program terminated* message to the navigator, along with any values returned by the application. At the navigator, this message triggers the execution of a transaction that will store the values returned by the application in the appropriate output data container. The navigator then proceeds to perform the corresponding navigation steps: check the end condition of the activity, if it is false the status of the activity is set to "terminated", if it is true the activity status is set to "finished" and then the outgoing control connectors evaluated, and so forth. As a final step, the navigator sends an *activity terminated* message to the user interface indicating that the selected activity has completed its execution. This message results in the activity being deleted from the worklist.

3 WORKFLOW MANAGEMENT SYSTEMS

There are three key features in any successful workflow product: *availability*, *scalability*, and *industrial strength design* [4, 25, 41]. Without availability, workflow systems will not be used for mission critical processes. Without scalability, they will not be used to support large organizations. Without industrial

strength, their applicability is greatly reduced. The problem with these obvious requirements is that they exceed those of current database and transaction processing technology, which can be considered the state-of-the-art in corporate computing. As a consequence, the robustness and technological maturity reached in the transaction processing area is all but lacking in workflow systems [22]. In spite of their initial success, current systems still need to be further developed along those three areas:

3.1 Availability

The goal of current systems is to become the central tool for the coordination of mission critical processes. The most likely candidates to use current workflow systems are large corporations in which the number of potential users can be in the tens of thousands, the number of concurrent process in the hundreds of thousands, and the number of sites connected to the WFMS in the thousands, distributed over a wide geographic area and based on heterogeneous systems [34]. In such environments, availability is a key feature. Fortunately, most failures in a workflow system can be masked using the redundancy inherent to the architecture. For instance, it is common to have the same application installed in several nodes. If one of them is not available, it may be possible to invoke the application at a different node. The same applies to all other components except to the storage server. A workflow system acts as an execution engine driven by the storage server, currently implemented in most systems as a centralized database. This centralized database becomes, sooner or later, a bottleneck and a single point of failure. It is certainly possible to rely on the underlying database to provide the necessary degree of availability. This approach has significant disadvantages, however. In the first place, database techniques are usually product based, i.e., the primary and the backup are the same database. In practice, this would tie the workflow architecture to a particular database and is in conflict with the distributed and heterogeneous nature of the system. It would also 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 solution would be fairly expensive and it does not provide a good way to cope with the heterogeneity of the storage servers (it is not reasonable to expect that all "workflow clusters" will use the same database as storage server). In the second place, the granularity used in database solutions is very fine, mainly pages or log records [42], and ignores the semantics of the application. The advantage of having a well defined application and a limited set of interactions would be lost. Finally, availability is always achieved at a

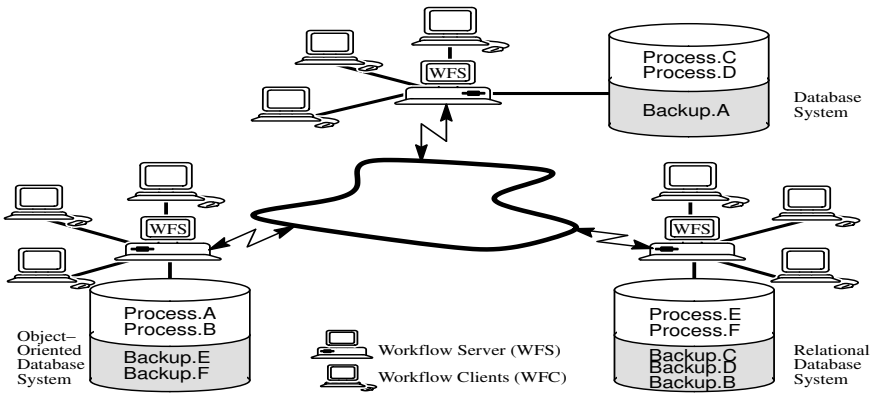

Figure 8 *A flexible backup architecture for workflow systems*

price. When and how to pay this price should be an adjustable parameter so as to make the system as flexible as possible.

One way to address these concerns is to provide a backup architecture that is database independent, uses knowledge of the semantics of workflow operations to optimize the exchange of information between the primary and the backup, and allows to adjust the degree of availability in the system [34]. For this purpose, standard database techniques such as hot-standby, cold-standby, 1-safe, and 2-safe, can be used [26]. These approaches can be combined to provide a flexible mechanism for high availability on workflow systems. Three process categories are defined: *normal*, *important* and *critical*. Critical processes use a 2-safe, hot standby policy, i.e., critical processes can resume execution almost immediately after a failure. Important processes use a 2-safe, cold standby approach, i.e., execution can be resumed after failures but only after some delay necessary to update the backup. Normal processes do not use any backup strategy, i.e., execution can only be resumed after the failure has been repaired but, in exchange, normal processes do not create any extra overhead in the workflow system.

Since the degree of availability is set at the process instance level, it is no longer possible to predetermine the primary and backup locations for a process. For this reason and to achieve database independence, there is no single backup for the system. Each storage server will act as both primary and as backup depending on the particular process instance as shown in Figure 8. Thus, the backup mechanism can be implemented as part of the standard communications between storage servers. The only difficulty being that the primary and the

backup may have different schemas (for instance, between a relational database and an object-oriented database). This problem can be solved by relying on semantic information about the workflow language, which is used to define a *canonical representation* in which each component of a workflow process is uniquely identified. When passing information between primary and backup, this is done using the canonical representation. In practice, this means that the primary only reports state changes to the components of a process which opens up the opportunity to optimize storage and communication overhead. In addition, this backup architecture also allows to perform load balancing in the system by moving the execution of a process from one location to another which effectively provides a way to migrate processes and sets the basis for scalable architectures.

3.2 Scalability

Due in part to the emphasis placed on cooperation by the first workflow products, most of them were designed with small groups in mind. In many ways, workflow systems have been victims of their own success since once users realized the potential of workflows, these engines were applied in large scale environments for which they were not designed [4, 25, 41, 48]. Other design issues aside, the main problem of current systems in terms of scalability is that they rely on a centralized database to implement the storage server, thereby introducing a serious bottleneck in the architecture. There are, of course, several advantages to the centralized approach: lightweight clients, centralized monitoring and auditing, simpler synchronization mechanisms, and overall design simplicity. But, in general, a centralized database results not only in scalability problems but also in performance limitations. The latter are not usually a concern in business processes but they are if the workflow system is used to execute automatic activities in a distributed system. Such problems can be addressed in several ways: using distributed execution instead of centralized control, and providing a way to tie together several workflow systems, each with its own storage server, into a bigger system. The former approach is still largely a research proposal, the latter a solution currently adopted by most products.

The idea of distributed execution was pioneered by the INCAS prototype [9]. In INCAS, the execution of a process takes place through an *Information Carrier*. The information carrier is an object that migrates from location to location as execution proceeds. It contains all the information relevant to the execution of the process so as to allow navigation to take place by consulting

the data in the information carrier. A similar approach is followed by EXOTICA/FMQM, FlowMark on Message Queue Manager [6]. In Exotica/FMQM, each node functions independently, the only interaction between nodes being through persistent messages used to trigger the next step in the execution. The basic idea is to partition the process definition into independent subsets that are distributed to the nodes were execution may take place. In contrast to the information carrier of INCAS, where all the information moves from node to node as navigation takes place, in Exotica/FMQM each node stores locally all the information it needs to perform navigation on a given process. Such an approach has also been followed by other prototype systems [54]. This greatly reduces the communication overhead between nodes and solves some additional problems related to monitoring and state detection. Independently of the form in which navigation takes place, the advantage of the distributed approach is that the need for a centralized database is avoided, which eliminates the performance and scalability bottleneck. Moreover, the resulting architecture also increases the resilience to failures since the crash of a single node does not stop the execution of other active processes. It is also possible to combine this distributed approach with a backup mechanism such as the one described above to provide both scalability and availability.

An alternative to distributed execution is to use several identical, independent systems. One primitive form of this approach has been successfully used in environments that tolerate *load partion*. If all processes are entirely independent of each other and the shared resources (corporate databases, for instance) are capable of supporting the accumulated load, it is possible to use several identical systems with each one executing part of the total load. This approach allows linear growth but it does not really address more fundamental problems as there is no way for the independent systems to communicate with each other. A more sophisticated solution is based on the same mechanisms described above for increasing the availability of the system. Both critical processes and important processes are replicated somewhere else in the system. Instead of using the copy for backup purposes, it is possible to use it to migrate the execution of processes from the primary to other locations as the load at the primary increases. In this way, the scalability problem becomes just a matter of providing enough locations in which processes can be run. All these locations will share the environment information, which can be easily replicated at all sites since it does not change often. The links between the different locations (necessary for the backup architecture) can also be used for communication between navigation servers so as to allow a navigation server to invoke a subprocess at a different location [8].

The idea of process migration and remote invocation requires to have reliable communications between the different locations. As with many other distributed applications, workflow systems usually rely on persistent queuing to provide some basic guarantees in the exchange of information [6]. This and other basic features are not present in current systems, which makes it difficult to implement solutions to the existing problems. A first step in the evolution of any workflow system is, therefore, to provide the industrial strength of database and TP-monitors.

3.3 Industrial Strength

Any new system needs some time to evolve and resolve the design inconsistencies, limitations and lack of flexibility of the initial versions. After this evolution period, products become more stable, their functionality well defined, reaching a degree of maturity that makes them reliable, understood and accepted by users. Workflow systems have not yet reached such a state. The demands placed on existing workflow systems go well beyond their capabilities and, in many cases, the *customer profile* designers had in mind was quite different from that of the actual users [48]. The limitations on scalability and availability discussed above are obvious examples, but there are many other glaring limitations. Some of them are product specific and related to the history behind the product (whether it evolved from a document management system, the tools available at the time it was designed, the position of the company in the market, etc.). Examples abound: inability to use subprocesses due to the way data is handled, scalability problems due to the underlying database, architectural limitations due to the communication system used, excessive emphasis on modeling philosophy, and so forth. These limitations are being quickly corrected as the products start to gain a wider customer base and experience with users provides the necessary feedback. There are, however, another set of limitations common to most systems that have no easy solution but need to be addressed before workflow system can claim to have reached any reasonable degree of maturity.

Among these open questions, the one most often mentioned is exception handling. In environments where the number of concurrent instances may be in the hundreds of thousands with every instance taking several weeks to complete, exceptions affecting single processes are likely to occur. Moreover, it is also likely that, occasionally, the behavior of all active instances needs to be modified to accommodate changes in the organization. These two types of changes are currently not satisfactorily supported. The difficulty they pose derives from

the way process instances are stored. There are two ways of doing it: as a compiled program or, more often, as a collection of database entries. Once created, modifying this implicit or explicit "script" is not an easy matter. Any possible exception that may appear during the execution must be coded in as part of the behavior of the process. Otherwise, exceptions to the expected behavior can only be solved by aborting the entire process (or by invoking a subprocesses that hopefully can solve the situation, but this creates a considerable overhead for the end user). Ideally, exceptions should be handled in a more uniform way, allowing the user to access the process definition, do the necessary changes and resume the execution of the process. This requires a very flexible handling of the process definition: rescheduling activities that have been modified, reusing results that have not been affected by the modification, and mapping the state of the old process to the state of the new process. Existing systems are still too rigid to provide such capabilities [28].

Another important issue is the interaction with external applications. In current systems, it is usually not possible to suspend the execution of the external application when the corresponding activity is suspended or the entire process aborted. It is also not possible to control any side effects that the application may cause. As a result, failures and rollback of processes become a fairly complex issue for the user. Currently, these problems are solved via manual intervention (even detecting that there is a problem is left to the user in some systems). In the future, a tighter integration will be desirable. This may be achieved by using standard interfaces or by using persistent queues as a way of ensuring reliable asynchronous communication between autonomous systems [6].

A third issue related to industrial strength is the ability to express logical units within the workflow language. For this, transactional concepts could be used. There is an extensive literature on advanced transaction models [18] which has touched upon many areas related to workflow management [5, 11, 21, 44, 53]. Transactions are an excellent abstraction to encapsulate behavior (atomicity and isolation, for the most part) and have proven extremely useful in developing a widely accepted theory of transaction management. Current commercial workflow systems, however, do not incorporate transactional notions but there are many indications that this will change in the future [3, 5, 14, 15, 33, 37, 41]. In a workflow environment, transactions can play a significant role as a system component. Persistence in workflow systems is achieved by using a database, a feature that it is unlikely to change. Interactions with databases require to use transactions (as shown in Figure 7). The very nature of the environment requires to use transactions if execution guarantees have to be provided. The same problems of distributed commitment and atomicity that arise in any dis-

tributed environment also arise in a workflow system. These problems could be addressed using the concepts successfully implemented in TP-monitors [26]. In addition, transactions may also play a significant role in the workflow language. As has already been pointed out [5], many of the ideas proposed in advanced transaction models can be used in workflow environments: compensation [21], alternative execution [44], spheres of control and atomicity [37], to mention a few. Thus, workflow system can be seen as a ubiquitous programming environment for implementing the applications targeted by advanced transaction models [5, 23, 24]. An example of how transaction may influence the workflow language is the use made of transactions in Encina, a TP-monitor that provides *transactional C* [51]. Transactional C is an extension of C in which it is possible to bracket sets of instructions (usually service invocations) within a transaction and specify what to do in case the transaction commits or aborts. The same idea, as well as more sophisticated concepts, can be applied to the workflow language in a similar fashion to allow the programmer of workflow processes to specify, for example, units of atomicity or compensation expanding several activities [37] or alternative execution paths in case of exceptions [5].

4 EVOLUTION OF WORKFLOW MANAGEMENT SYSTEMS

This section discusses and motivates how workflow management technology can be applied and how it might evolve, the main idea being that workflow technology could be used as the basis for programming in distributed, heterogeneous environments.

4.1 Distributed Environments

As mentioned throughout the chapter, the future of workflow management is strongly tied to the evolution of distributed computing. As such, distributed environments require the four categories of functionality discussed in the introduction: interface definition, communication, execution guarantees, and development environments. While existing products are far from providing the four categories, they are slowly converging towards systems that do provide such functionality in an integrated an efficient manner. In such systems, workflow concepts could be one of the basic tools for programming distributed systems.

The characteristics of such distributed environment can be easily derived from the target architecture of existing systems. A quick look to the manuals of products such as implementations of the CORBA standard, TP-monitors, queuing systems, and workflow tools reveals striking similarities in their architecture. In all cases, the system can be succinctly described as shown in Figure 9.

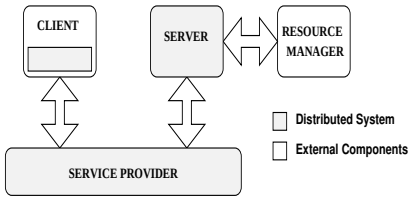

Figure 9 Generic architecture of a distributed system

In general, the *client* represents the user program invoking the services provided by the distributed system. The client usually resides outside the distributed system but interacts with it through a well defined set of APIs. The *server* in Figure 9 is in most cases a simple proxy for the actual service provider which is the *resource manager*. It acts as the common interface for all server applications and it is, in most cases, a glorified form of wrapping. In CORBA, for instance, the server is simply an *object adaptor* that takes care of enforcing the common interface. What CORBA calls a server is actually the resource manager, or the actual implementation of the objects. Finally, the *resource manager* is the application that performs the operations requested by the client. In TP-monitors, for instance, the resource managers are usually databases.

The way this generic architecture is used in practice is common to all the products. In general, the system reacts to client invocations (for instance, a workflow process is started, a method of an object is invoked, a service call takes place). As a result of the invocation, the service provider locates the resources necessary to satisfy the request (for example, locating a server, rerouting the request, recording the invocation, triggering side actions) and forwards it to the appropriate server which, in turn, will translate the request to call to the corresponding resource manager. Once the resource manager terminates, the results are returned to the client by retracing the previous steps. Obviously this is a simplification of what happens in real systems and there are many optimizations and variations of this basic mechanism. The main concepts are, however, similar in all systems, which points towards the emergence of a common platform for distributed computing. It is only natural to expect that, at least for a large class of applications, workflow concepts will be used in this new context as a way to build applications over distributed systems.

4.2 Programming in Heterogeneous, Distributed Environments

In an environment like the one described, there is a wealth of computer tools to perform individual tasks. Workflow management simply provides a way to integrate these tools into a more meaningful system by combining them as necessary on a per process basis: use individual applications as *service providers* and the workflow system as the language to specify the interactions between these service providers. Scientific data management offers a good example of how this idea can be applied in an area different from business processes. Scientific applications are known for the size and volume of the data involved. Moreover, scientific data has the added problem of the multiple formats in which the information is represented and the multiple transformation to which it is subjected. Most existing research in scientific data management often overlooks the fact that scientific data is seldom used raw. In most cases, the data undergoes complex and successive transformations as part of sophisticated models of physical phenomena. Such transformations are a source of *derived data* which cannot be interpreted correctly without knowledge about how it was created. To make matters worse, the transformations and models themselves may evolve as more precise knowledge is available. Support for tracking these data dependencies and evolution is all but lacking in current systems. A number of solutions have been proposed [27, 35], some of them pointing towards using workflow concepts as a way to cope with these problems [2, 40, 12].

Consider, for instance, the model shown in Figure 10 as a typical example of how scientific data is handled. The purpose of the model is to study the changes in the erosion patterns, vegetation and hydrographic characteristics of a given area. The model can be divided in three parts. The erosion model takes information about the slopes of the area, its soil characteristics and vegetation cover to produce an estimate of the erosion of the terrain. Note that the soil information is obtained directly from available data. However, the slope information is not readily available and requires taking elevation samples and processing them to get the desired information. This is done by using two more models, the Digital Elevation Models and Slope Models, also shown in the figure. The data about vegetation changes is the result of a vegetation evolution model. This model takes several inputs, some of them primitive, i.e. raw data such as the soil map, and some of them derived (by applying other models). Finally the discharge model involves interpolating rainfall records, calculating the storm coverage and applying a flow analysis algorithm to define an hydrograph (showing the flow of water at a given point).

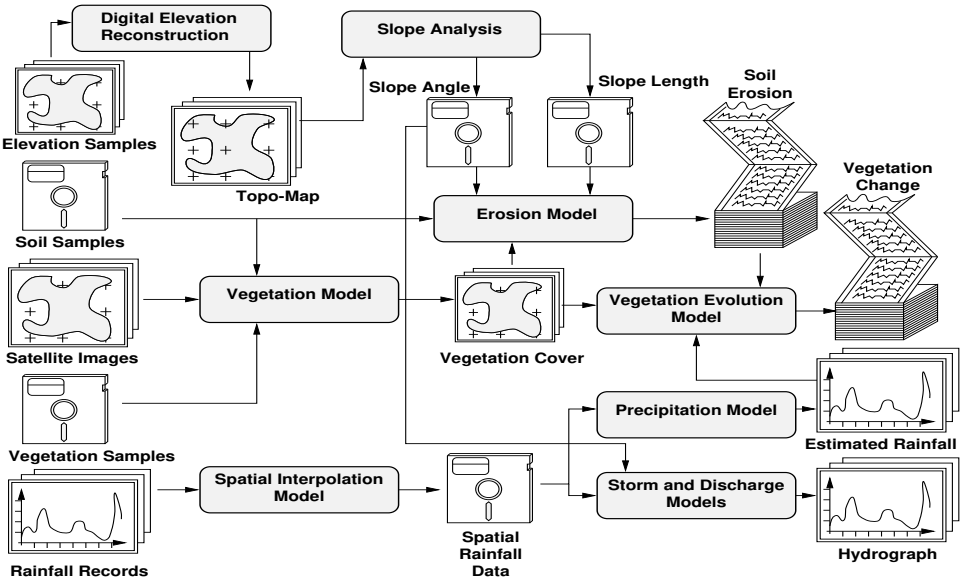

Figure 10 Scientific modeling as a workflow process

Workflow systems provide the tools necessary to capture such *modeling* activities. Figure 10 can be viewed as a workflow process in which the control flow follows the modeling logic and the data flow corresponds to the outputs of particular algorithms that are used as inputs to the next set of algorithms. Using a workflow system for such purpose helps to solve many of the problems posed by scientific data. To start with, the execution is persistent and can be distributed across many different nodes which, first, provides a considerable degree of reliability and, second, opens up the opportunity to parallelize and distribute expensive computations across a network of computers. Moreover, the auditing and monitoring tools of the workflow system keep track of every step of the execution and the data produced. Questions such as the lineage of a data set (how it was produced), data dependencies between data sets, and the algorithms involved in a given model can be easily answered by consulting the audit data of the workflow system. Moreover, complex tasks such as automatic change propagation (triggering the execution of a process when one of its inputs changes) and maintaining data consistency can be performed automatically by the system by using the information recorded about every process.

These ideas can be applied in a variety of scientific environments, and require very small modifications to the workflow engine. It should even be possible to

build a basic system using current commercial workflow products [12, 40]. An important consequence of this example is that workflow concepts can also play a role as control tools of a particular type of high performance computations when these are performed over a network of workstations. Provided that the algorithms for the individual steps are in place, the workflow system can be used as both the language to build (meta-) programs out of those existing programs and the distributed environment in which to execute them.

5 CONCLUSIONS

The notion of workflow management can be traced back to prototypes and research carried out many years ago. Some [49] propose as the earliest ancestors the SCOOP project [56] and Office Talk [16]. Others see the roots of workflow management in the work of imaging companies [20]. In the database community workflow ideas have been proposed under many disguises, mostly in the form of *advanced transaction models* [18, 21, 44, 53, 36]. The Workflow Management Coalition [30] suggests no less than six areas that have had a direct influence on the development of workflow management as it is today: image processing, document management, electronic mail and directories, groupware, transactional systems, project support applications, and business process re-engineering and structured system design tools. Even one of the most popular workflow modeling paradigms [1, 39] can be traced back to early work on artificial intelligence and speech theory. In general, the need for workflow functionality was identified long ago by different communities as they realized the potential offered by computers and communications. For instance, just in the last decade, similar ideas were discussed in areas such as *paperless office* [52], *office automation* [10], *groupware* [16], or *computer supported cooperative work* [36].

Although these are not new ideas, the advances in technology allow to take workflow concepts well beyond the original goals. A workflow management system can be viewed as a meta-programming tool using autonomous, heterogeneous applications as basic instructions. The possibilities of such an approach can be clearly seen in the area of business process reengineering, where workflow management systems have provided an efficient way of designing very complex distributed applications reusing existing components. The example discussed above regarding scientific computing shows that these same ideas can be successfully applied in many other areas, turning workflow management into a key component of future systems. The functionality described in this chapter is not tied to business processes. What today is commonly presented as a workflow

model, has been discussed here as a programming language with the workflow system playing the roles of developing and execution environment. This execution environment is common, as shown above, to many applications and the attention it has received recently makes it a likely candidate to be one of the basic computer infrastructures in the near future. Other efforts like TP-monitors, CORBA, or queuing systems are addressing crucial aspects of such execution environments. Workflow management should be viewed as one more effort in this direction. The focus on business process has helped to create an initial market and allowed to gain important experience in the usage of workflow systems. The next step is to extrapolate these ideas to other areas and combine workflow technology with other on-going efforts in distributed computing to arrive at the next generation of distributed processing tools.

Acknowledgements

Part of this work has been done in the context of the Exotica project ongoing at the IBM Almaden Research Center since 1994. We are grateful to all past and current members of the Exotica project, A. El Abbadi, D. Agrawal, R. Günthör, M. Kamath and B. Reinwald for their contributions to the project. Some of this work is also part of current research projects of the Database Research Group of ETH Zürich. We are grateful to Claus Hagen and Hans-Jörg Schek for many helpful discussions about the future of workflow management systems. Even though we refer to specific IBM products in this chapter, no conclusions should be drawn about future IBM product plans based on the contents of this chapter. The opinions expressed here are our own.

REFERENCES

[1] ActionTechnologies, "What is ActionWorkflow - A Primer", ActionTechnologies, Inc., Alameda, California, 1993.

[2] G. Alonso and A. El Abbadi. "Cooperative Modeling in Applied Geographic Research", International Journal of Intelligent and Cooperative Information Systems, 3(1), 1994, pages 83–102.

[3] G. Alonso, D. Agrawal, A. El Abbadi, "Process Synchronization in Workflow Management Systems", 8th IEEE Symposium on Parallel and

Distributed Processing (SPDS'97), October 23-26, 1996, New Orleans, Louisiana, USA.

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

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

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

[7] 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. Vol. 4, No. 3, pp. 229-247, July 1996.

[8] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan, "Failure Handling in Large Scale Workflow Management Systems", IBM Technical Report RJ 9913, Almaden Research Center.

[9] D. Barbara, S. Mehrota, and M. Rusinkiewicz. "INCAS: Managing Dynamic Workflows in Distributed Environment", Journal of Database Management, 7(1):5–15, IDEA Group Publishing, 1996.

[10] G. Bracchi and B. Pernici. "The Design Requirements of Office Systems". ACM Transactions on Office Information Systems, 2(2):151–170, April 1985.

[11] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, A. and G. Weikum, "Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows", ACM Sigmod Record, September, 1993.

[12] A. Bonner, A. Shrufi and S. Rozen, "LabFlow-1: A Database Benchmark for High Throughput Workflow Management", Proceedings of the Fifth International Conference on Extending Database Technology (EDBT96), Avignon, France, March 25-29, 1996.

[13] C. Bussler. User Mobility in Workflow-Management-Systems. Proceedings of the Telecommunications Information Networking Conference (TINA '95), Melbourne, Australia, February 1995.

[14] Q. Chen and U. Dayal, "A Transactional Nested Process management System", Proceedings of the 12th International Conference on Data Engineering, February/March, 1996, New Orleans, Louisiana, USA.

[15] J. Eder and W. Liebhart, "Workflow Recovery", Proceedings of the 1st International Conference on Cooperative Information Systems (CoopIS96), Brussles, Belgium, June, 1996.

[16] C. A. Ellis, S. J. Gibbs and G. L. Rein, "Groupware, Some Issues and Experiences", Communications of the ACM (CACM), 34(1):39–58, January, 1991.

[17] M. Emmrich. "Object Framework for Business Applications", Proceedings of the 5th EDBT, Avignon, France, March, 1996.

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

[19] W. Fisher and J. Gilbert. "FileNet: A Distributed System Supporting WorkFlo; a Flexible Office Procedures Control Language", IEEE Computer Society Office Automation Symposium, pages 247–249, Gaithersburg, MD, April 1987.

[20] C. Frye. "Move to Workflow Provokes Business Process Scrutiny", Software Magazine, pages 77–89, April 1994.

[21] H. García-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem, "Modeling Long-Running Activities as Nested Sagas", Bulletin of the IEEE Technical Committee on Data Engineering, 14(1):18–22, March 1991.

[22] D. Gawlick. "High Performance TP-Monitors – Do We Still Need to Develop Them?", in [45].

[23] D. Georgakopoulos and M. Hornick, "A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows", International Journal of Intelligent and Cooperative Information Systems, 3(3), September, 1994.

[24] D. Georgakopoulos, M. Hornick, and F. Manola, "Customizing Transaction Models and Mechanisms in a Programmable Environment Supporting Reliable Workflow Automation", IEEE Transactions on Knowledge and Data Engineering, April, 1996.

[25] D. Georgakopoulos, M. Hornick, A. Sheth. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, vol. 3, no. 2, April, 1995, pp. 119-153 .

[26] J. Gray and A. Reuter. "Transaction Processing: Concepts and Techniques", Morgan Kaufman, 1993.

[27] N.I. Hachem, K. Qiu, M. Gennert and M. Ward. "Managing Derived Data in the Gaea Scientific DBMS", Proceedings of the 19th International Conference on Very Large Databases, Dublin, Ireland, 1993.

[28] C. Hagen, "Kombination von aktiven Mechanismen und Transaktionen im TRAMs-Projekt", 8th Workshop "Grundlagen von Datenbanken", Friedrichsbrunn, Deutschland, May, 1996.

[29] M. Hammer and J. Champy, "Reengineering the Corporation: A Manifesto for Business Revolution", HarperBusiness, New York, 1993.

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

[31] M. Hsu. Special Issues on Workflow and Extended Transaction Systems, Bulletin of the IEEE Technical Committee on Data Engineering, 16(2), June 1993 and 18(1), March 1995.

[32] IBM, "FlowMark - Managing Your Workflow, Version 2.1", IBM Document No. SH19-8243-00, March, 1995.

[33] I. Ben-Shaul and G.E. Kaiser, "A Paradigm for Decentralized Process Modeling", Kluwer Academic Publishers, Boston, 1995.

[34] M. Kamath, G. Alonso, R. Günthör, C. Mohan. "Providing High Availability in Very Large Workflow Management Systems", Proceedings of the Fifth International Conference on Extending Database Technology (EDBT96), Avignon, France, March 25-29, 1996.

[35] R.H. Katz *et al.*, "Design of a Large Object Server Supporting Earth System Science Researchers", AAAS Workshop on Advances in data Management for the Scientist and Engineer, Boston, Massachussetts, USA, February, 1993, pages: 77–83.

[36] T. Kreifelts, E. Hinrichs, K.H. Klein, P. Seuffert, G. Woetzel. "Experiences with the DOMINO Office Procedure System", Proceedings ECSCW '91, Amsterdam, September, 1991.

[37] F. Leymann. "Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems", GI-Fachtagung Datenbanken in Büro Technik und Wissenschaft - BTW'95, Springer-Verlag, Dresden, Germany, 1995.

[38] J. Lyon, "Tandem's Remote Data Facility", Proceedings of IEEE Compcon, 1990.

[39] R. Medina-Mora, H.K. Wong and P. Flores, "ActionWorkflow as the Enterprise Integration Technology", Bulletin of the IEEE Technical Committee on Data Engineering, 16(2), June, 1993.

[40] J. Meidanis, G. Vossen, M. Weske, "Using Workflow Management in DNA Sequencing", Proceedings of the 1st International Conference on Cooperative Information Systems (CoopIS96), Brussles, Belgium, June, 1996.

[41] C. Mohan, "Tutorial: State of the Art in Workflow Management System Research and Products", 5th International Conference on Extending Database Technology, Avignon, March 1996 and at ACM SIGMOD International Conference on Management of Data, Montreal, June 1996.

[42] C. Mohan. "A Survey of DBMS Research Issues in Supporting Very Large Tables", Proceedings of Data Organization and Algorithms, pages 279–300, Chicago, Il., 1993.

[43] C. Mohan, R. Dievendorff, "Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing", Bulletin of the IEEE Technical Committee on Data Engineering, 17(1), March, 1994.

[44] M.H. Nodine and S.B. Zdonik, "Cooperative Transaction Hierarchies: A Transaction Model to Support Design Applications", Proceedings of the 16th International Conference on Very Large Databases, Brisbane, Australia, August, 1990.

[45] R. Obermack.,"Special Issue on TP Monitors and Distributed Transaction Management", Bulletin of the IEEE Technical Committee on Data Engineering, 17(1), March, 1994.

[46] Object Management Group, "The Common Object Request Broker: Architecture and Specification", John Wiley, 1992.

[47] W. Schaad, H.-J. Schek and G. Weikum, "Implementation and performance of multi-level transaction management in a multidatabase environment", Proc. of the 5. Int. Workshop on Research Issues on Data Engineering, Distributed Object Management, Taipei, Taiwan, 1995.

[48] B.R. Silver, "The Guide to Workflow Software: A Visual Comparison of Today's Leading Products", BIS Strategic Decisions, 1995.

[49] K.D. Swenson, R.J. Maxwell, T. Matsumoto, B. Saghari, and K. Irwin, "A Business Process Environment Supporting Collaborative Planning", Journal of Collaborative Computing, 1(1), Spring, 1994.

[50] L. Thé, "Getting into the Workflow", Datamation, October, 1994.

[51] Transarc Corporation, "Writing Encina Applications", Transarc Corporation, 1995. ENC-D5012-00.

[52] D. Tsichritzis. "Form Management", Communications of the ACM, 25(7):453–478, July, 1982.

[53] H. Wächter and A. Reuter, "The ConTract Model", in [18].

[54] D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz-Dittrich, "The Mentor Project: Steps Towards Enterprise-Wide Workflow Management", Proceedings of the 12th International Conference on Data Engineering, February/March, 1996, New Orleans, Luoisiana, USA.

[55] Workflow Management Coalition Members, Glossary, A Workflow Management Coalition Specification. Technical report, The Workflow Management Coalition, November 1994. Accessible via: http://www.aiai.ed.ac.uk/WfMC/.

[56] M. Zisman. "Office Automation: Evolution or Revolution", Sloan Management Review, 19(3):1–16, Spring, 1978.