

# Providing High Availability in Very Large Workflow Management Systems<sup>1</sup>

M. Kamath  
Dept. of Computer Science  
University of Massachusetts  
Amherst, MA 01003, USA  
kamath@cs.umass.edu

G. Alonso  
Institute for Information Systems  
Database Group, ETH Zentrum  
CH-8092 Zürich, Switzerland  
alonso@inf.ethz.ch

R. Günthör  
IBM European Networking Center  
Vangerowstr. 18, 69115 Heidelberg  
Germany  
rgunthor@heidelberg.ibm.com

C. Mohan  
IBM Almaden Research Center  
650 Harry Road, San Jose, CA 95120  
USA  
mohan@almaden.ibm.com

## Abstract

Workflow management systems (WFMS) support the modeling, coordinated execution and monitoring of business processes within an organization. In particular, very large workflow management systems are used in organizations with several thousand users, hundreds of thousands of process instances, and several thousand sites, all distributed over wide geographic areas. In these environments, failure of the WFMS or the underlying workflow database which stores the meta-information about the processes is not tolerable. This paper addresses the problem of providing high availability in workflow management systems by proposing a backup technique which ensures that execution of a process instance can be resumed at any point in time in the event of a failure. An essential characteristic of our backup scheme is that it allows the user to define different availability levels, reducing the cost of maintaining backups. The backup scheme is implemented using the workflow semantics, which we believe will (i) make it independent of the underlying workflow database, thus permitting the use of heterogeneous databases as primary and backup, (ii) reduce overheads, especially when compared to backup schemes provided by database systems.

## 1 Introduction

Workflow management systems (WFMS) enjoy an increasing popularity due to their ability to coordinate and streamline complex *business processes* within large organizations. The goal of WFMSs is to orchestrate the execution of business processes, ultimately being responsible for the control of the organization's activities. Thus, a WFMS becomes not just a support tool but an integral part of the organization. This is substantiated by the fact that the most likely candidates to use a WFMS are large corporations such as banks, insurance companies and telecommunication companies which need to coordinate several instances of a variety of business processes. Current customer requirements indicate that the number of potential users can be in the tens of thousands, the number of process instances in the hundreds of thousands and the number of sites in the thousands all distributed over a wide geographic area and based on heterogeneous systems. With these figures, continuous availability becomes a crucial aspect of any successful commercial WFMS. Though more than 70 workflow products exist in the market [Fry94], this is an issue that has been ignored by workflow developers, and to our knowledge, has not yet been addressed by research in the area of workflow management.

---

<sup>1</sup>This work was performed while M. Kamath, G. Alonso and R. Günthör were visiting scientists at the IBM Almaden Research Center

This paper reports ongoing research in the availability of WFMSs, as part of the research efforts of the *Exotica* project [MAGK95]. The research has been centered around *FlowMark* [IBMa, LR94], a WFMS from IBM. However our results can be easily generalized to any WFMS.

In this paper our focus will be on WFMSs that use a centralized database (workflow database) to store meta-information about the business processes. They conform to the reference model developed by the Workflow Management Coalition [WPMC94] and typically handle *production* workflows [GHS95]. In environments where several workflow databases coexist, each of them typically stores a different subset of processes instances. If one database fails, all process instances running off that database will stop their execution. This behavior is unacceptable for several critical business processes which cannot be delayed. Hence the workflows database remains a single point of failure that may have grave consequences. To address this problem, our approach is to use process instance replication to guarantee that if the database where the particular instance resides fails, execution can be resumed elsewhere based on the replicated information.

The novelty of our approach is to use workflow semantics<sup>2</sup> as the basis for replication and backup, as opposed to low level constructs such as pages or log records used in traditional database backup techniques. The reasons are as follows:

- *Necessity for WFMSs that are database independent:* Relying on the backup facilities supplied by the underlying database ties the system to the platforms where such a database runs. Since we envision a system in which heterogeneous databases, even as different as relational and object-oriented to be used as backup for each other, low level techniques as those provided by database management systems cannot be used. By using application (workflow) level backup, the replication scheme becomes independent of the underlying platform, thus allowing the use of heterogeneous databases.
- *Need for flexibility in replicating process instance data:* In traditional database backup techniques, the units of exchange tend to be pages or log records [GMP90, BGHJ92, MTO93] which makes it difficult to control which data is actually being replicated. If such techniques are used to replicate data between two workflow databases, log records of changes to all objects, including those that correspond to static information about processes will be replicated. This is not really required and there can be excessive overheads when several hundred thousand processes run concurrently. Instead the overheads can be minimized by providing different availability levels (each having different overheads and degree of reliability) and exploiting workflow semantics to minimize the amount of information that is replicated.

The specific contributions of this paper are:

1. Provision for different availability levels by categorizing process instances as *normal*, *important*, and *critical*, each with its own backup approach.
2. A backup architecture and mechanism to efficiently use the resources available in a distributed workflow system to perform backup, *i.e.*, use the workflow servers and workflow databases to function both as a primary (for some process instances) and as a secondary (for some other process instances).

---

<sup>2</sup>From the perspective of the database that stores the meta-information, the WFMS is an application and hence workflow semantics can also be referred to as *application semantics*.

3. Backup algorithms to replicate process instance data for each of the above process instance categories during normal processing and when there are failures at the primary or backup site. These algorithms have been analyzed in a simple manner to show that they can scale for very large WFMSs.

The rest of the paper is organized as follows: in the next section we describe the basic ideas behind a WFMS and the backup requirements in WFMSs. A discussion on backup techniques and how they can be adapted to WFMS is presented in section 3. Section 4 discusses availability levels, the backup architecture and issues related to object state replication. In section 5 we introduce the necessary data structures, describe our backup algorithms. Section 6 concludes the paper.

## 2 Workflow Management

In this section we present some basic concepts of workflow management that are necessary for discussing particular aspects of the implementation of our replication and backup technique. Specific details of workflow management systems are described based on FlowMark. We then briefly discuss the requirements of high availability in WFMSs.

### 2.1 Workflow Model and Architecture

A common term used to refer to the work performed in large organizations is a *business process*, defined by the Workflow Management Coalition 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" [WfMC94]. A *workflow* is a particular representation of a business process. The role of a *workflow management system*, WFMS, is the scheduling and coordination of all the activities encompassing a business process. This involves determining the order of execution, interacting with each activity's tools, establishing the data flow between activities, mapping activities to users, checking the timeliness of the execution, monitoring the progress and determining when a process has terminated. Though these concepts have been around for a number of years, the technology required to implement suitable systems has however been available only recently. Currently, there is a considerable amount of attention devoted to this area [GHS95, Hsu95, She94].

A workflow model is a description of a business process. For this purpose, the Workflow Management Coalition has proposed the following reference model [WfMC94]: A business process is represented by a schema that includes the *process* name, version number, start and termination conditions and additional data for security, audit and control. A process consists of several steps. Each step within a process is an *activity*, which has a name, a type, pre- and post-conditions and scheduling constraints. It also has a *role* and may have an *invoked application* associated with it. The *role* determines who will execute the activity. The *invoked application* is the tool to be used in the execution, which may range from computer programs to human actions with no specific tool, e.g., a meeting. Furthermore, activities use *relevant data* defined as the data passed to and produced by the activities. Relevant data have a name and type associated with them. The users of the system are represented in terms of *roles*. Finally, and for notational purposes, a specific invocation of a process is referred to as a *process instance*. For a given process, there can be many instances of it running concurrently. The instances have the same components and structure as the process.

The architectural components of the reference model are as follows. A *process-definition tool* is used to define the schema of each business process. A *workflow engine* creates a process instance, interprets the process schema and determines which activities are ready for execution. If an activity is to be performed by a role, then the

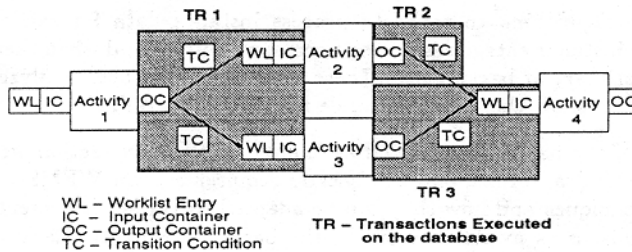


Figure 1: *Transactions Executed during Process Navigation*

eligible role is notified by placing the activity on the role's *worklist*. The role has to then select the activity for execution from the worklist using a *worklist-handler*. On the other hand, if the activity is to be performed automatically, then the workflow engine selects it for execution. If an application program is to be invoked to execute the selected activity, the workflow engine notifies an *application-agent* that exists at the location (node) where the program is to be invoked. The application agent then executes the program and returns the results of the program to the workflow engine.

For our study, we have selected FlowMark as the specific WFMS since its architecture closely resembles the architecture described by the reference model. FlowMark is based on a client-server architecture and uses an object-oriented DBMS as a central database to store meta-information about workflows. The meta-information consists of process scheme definitions and all runtime information related to active process instances. Besides the database, FlowMark is organized into four other components. The *FlowMark Server* acts as the workflow engine, the *Runtime Client* acts as the worklist-handler, the *Program Execution Client* acts as the application-agent, and the *Buildtime Client* acts as the process-definition tool. A single database usually acts as a repository to several workflow (FlowMark) servers.

*Navigation* is the procedure by which the server determines the next activities to execute. It takes place through transactional updates to the centralized database. Since the reference model does not provide implementation details, we describe navigation in the context of FlowMark since its model closely resembles the reference model. In FlowMark, the *flow of control*, which is specified by *control connectors* between activities, is the order in which these activities have to be executed. Each control connector has a state. Before the activity from where it originates finishes, the control connector is *unevaluated*. Once the activity terminates, the *transition condition* associated with the outgoing control connectors are evaluated and their state is set to TRUE or FALSE. Such transition conditions are boolean expressions involving data returned by activities in the output containers. For each activity there is an *input data container* and an *output data container* where the invoked application takes its input and places its output. The *flow of data*, specified through *data connectors*, corresponds to mappings between output data containers and input data containers and allows passing data from activity to activity.

The object representation of the modeling concepts discussed above is shown in Figure 1. The shaded areas represent sets of objects in the centralized database updated by different transactions. The interactions with the database in terms of transactions are as follows. Transaction  $TR_1$  changes the state of activity 1, updates its output container, evaluates the control connectors leading to activities 2 and 3. If the start condition of the succeeding activities evaluates to true, then the input containers are populated and the activities are added to the appropriate worklists.

Transaction  $TR_2$  is executed upon completion of activity 2 and performs much the same tasks as  $TR_1$ , except that activity 4 is not yet scheduled for execution. When activity 3 completes, it is transaction  $TR_3$  that will schedule activity 4 if its starting condition evaluates to true.

## 2.2 Requirements of High Availability

In large enterprises, different departments or divisions should be able to use existing database platforms as repositories (workflow database) for their WFMS. As a result, an important requirement is that the WFMS and the backup mechanism should be database independent. Also from the perspective of the enterprise, certain processes are mission critical and more important than others. Replicating all data immediately for providing high availability can result in huge overheads especially when several thousand process instances are running concurrently. On the other hand, without replication the system becomes very vulnerable to database failures. Both of these solutions are unacceptable and hence another important requirement is flexibility in defining the availability requirements of different processes.

## 3 Adapting Database Backup Techniques to Workflows

In this section we present salient aspects of backup techniques developed for databases and discuss how some of the basic concepts can be adapted to workflow environments. This will help us focus on the specifics of our workflow semantics-based backup technique in the rest of the sections.

The basic ideas behind backup mechanisms are well known. Consider, for instance, Tandem's Remote Data Facility [Lyo90]. In this environment a server or group of servers, known as *primary*, use a single backup computer, known as *secondary*. Under normal operation, a client sends requests to the primary, and the log records generated at the primary are sent to the backup and applied to its state. Therefore, the backup is an exact replica of the primary and it is kept up to date, which allows the backup to take over almost immediately upon failure of the primary. This mode of operation in which the backup can take over immediately is known as *hot standby*. Note that the overhead incurred is high: a transaction takes place in the primary, log records are sent to the secondary, a transaction applies those log records in the secondary, and a 2 Phase Commit protocol (2PC) is used to guarantee the combined atomicity of both transactions. Note that since there are only two participants involved, the primary and the backup, some optimizations of 2PC can be implemented [SBCM95, GR93]. When using 2PC, this approach is known as *2-safe* and it is similar in some aspects to the very-safe case of [GR93]. Contrary to the 2-safe policy, the *1-safe* policy does not require the primary to wait for the secondary. It commits its transaction first and then propagates the log records to the secondary. There is, of course, the risk of data loss when the backup takes over but in practice the 1-safe policy is preferred over the 2-safe due to its lower overhead. Algorithms for the maintenance of remote copies under the 1-safe and 2-safe policies are discussed in [GMP90] and [BGHJ92] respectively.

In the case of workflow systems, we do not believe that the 1-safe option is viable always. In the case of traditional databases, since transactions are short and perform few changes, losing these changes may not be critical. In a WFMS, an update may correspond to the execution of very complex activities, making the loss of data unacceptable in many cases. Hence in this paper we will discuss only *variations of the 2-safe policy* to ensure that the take over by the backup does not require redo of work already completed.

A hot standby policy places a heavy burden on both the primary and the backup. When a dedicated machine is used as a backup, the load in the backup is not a major

concern. However in our case, it is a concern since the backup is not a dedicated machine but a workflow server that is used both as a primary for some process instances and as a backup for some other process instances. This is done to effectively use the available resources. In some cases, quick take over by the backup may not be a key issue, allowing the backup to perform the updates asynchronously and therefore reducing the overhead due to replication. In the extreme case, the backup does not perform any updates unless it has to take over the operation of the primary. This is known as a *cold standby* policy. In this policy, the information sent from the primary is stored at the backup but not applied immediately. Obviously, the trade off is lower overheads at the primary during normal operation at the cost of a slower recovery at the backup in case of failures. To add flexibility to our design, we provide the user with the possibility of using both approaches: cold standby and hot standby.

Inconsistencies that can arise due to transaction dependencies during crash recovery are discussed in [FCK87, MHL<sup>+</sup>92]. However they are irrelevant in the workflow context since the execution states of different business processes are independent. Techniques to replicate data at a high level in the context of relational databases are discussed in [Gol94] where SQL calls are trapped at the primary and executed at the backup site. However, this is not a suitable approach since there are enormous overheads due to duplication of processing and different order of executions at the primary and backup can cause inconsistencies within a business process.

Coordinating the nodes of a distributed system that contain replicated data is discussed in [CS92] in the context of the  $D^3$  model. The model provides schemes for choosing a coordinator and a new primary/backup in case of node failures. The model also has provision for load-balancing and for adding/removing nodes from the system without interruption of normal operation. Since these techniques can be directly adapted to WFMS, we do not discuss node coordination issues in the rest of the paper. Techniques that exploit parallelism in processing the log records at the backup, and schemes that allow new transactions to begin in parallel along with the takeover process are discussed in [MTO93]. These can be adapted to WFMS to speed up recovery. A disaster recovery scheme with options to choose the desired degree of data durability and service durability through different *insurance levels* and *readiness levels* on a per-transaction basis is discussed in [BT90]. However, the scheme is based on remote logging of data pages and cannot be used for process-instance replication since this can lead to higher overheads.

Apart from less overheads, another important requirement for our backup technique is the fact that it should be database independent. A workflow system is designed to run over multiple platforms and each server may be connected to an entirely different database, not only in terms of schema but also in terms of conceptual approach. Hence the assumption used in traditional backup techniques that the backup system is identical to the primary is no longer true in a WFMS. Therefore, we cannot rely on physical descriptions such as pages, segments or log records because they do not have any meaning across heterogeneous systems. The only entities that maintain their meaning from server to server are workflow objects, which are actually databases independent. Hence, to design a backup mechanism that can be implemented across heterogeneous systems and that minimizes the replication overheads, data replication must be performed at the application (workflow) level.

#### 4 The Exotica Approach to High Availability in WFMSs

In this section we describe the details of our backup approach. Specifically we focus on availability levels, the backup architecture and issues related to object-state replication. Through the rest of the paper we use the following notation for the primary

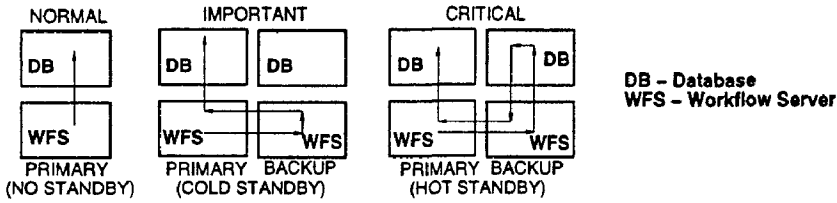


Figure 2: Process Categories and Replication Policies

and the backup. The primary and secondary copies of the workflow data are located at the primary and the secondary *databases* respectively. The workflow (FlowMark) servers that are connected to the primary and secondary database are termed as primary and secondary *servers* respectively.

#### 4.1 Availability Levels — Cost-effective Backup of Process Instances

To provide an adequate level of flexibility in the design of workflows and to allow the user to specify when it is cost-effective to provide a backup for a given process, we define three process categories: *normal*, *important* and *critical*. We associate with each of them a different backup mechanism. From the many variations that are possible, we selected them since they are the most meaningful. These three *backup categories* are described below and the message exchanges they require are shown in Figure 2.

- **Critical:** similar to 2-safe with hot standby. When navigation takes place at the primary and workflow objects change their state, the new state is sent to the backup. The backup will update the corresponding local objects immediately.
- **Important:** similar to 2-safe with cold standby. For this type of processes, the procedure is identical to that followed for critical processes, except that the backup does not apply the new state immediately: the coordination is between the primary committing the navigation transaction and the backup committing the new state to stable storage. These changes can be applied to the database in an asynchronous and lazy fashion, thereby reducing the overheads both at the primary and the backup.
- **Normal:** processes of this type are not replicated at all. If forward recovery is provided, there is no data loss when there are server failures. However, execution cannot be resumed until the same server is back on line.

The particular category to which a process belongs is specified by the workflow designer. There are two possibilities. One is to associate a backup category with a process class. The other is to associate the backup category with the process instance. They are not mutually exclusive options, and they are orthogonal to the discussion here so we will not analyze them further. Finally, and since most workflow systems allow the nesting of processes, it is necessary to specify the semantics of the backup category for nested processes. For simplicity, we will assume that the subprocesses will *inherit* the category of the parent process, even if it has a different category on its own. In the rest of the paper we focus only on the critical and important process categories.

An alternative to availability levels is to have different WFMSs whose underlying workflow databases have different degrees of reliability and assign process instances dynamically to the WFMSs. We believe such alternatives do not provide the required flexibility and also increase resource requirements and system complexity.

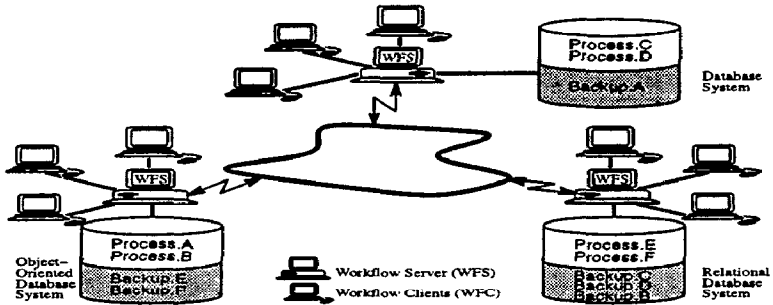


Figure 3: A Backup Architecture for Workflow Environments

### 4.2 Backup Architecture

Most WFMSs are based on a client/server architecture with the workflow server connected to a single workflow database. In a large enterprise, there are typically several servers and several databases, many of which could be heterogeneous in nature. A dedicated backup approach in such a highly distributed and heterogeneous environment would require a backup for every individual system. Hence it is not feasible to use a dedicated backup since the cost is too high. Therefore, in our implementation we do not architecturally distinguish a primary server from a backup. All workflow servers are used both as primary for some processes and as backup for others. This being the case, there is no guarantee that the backup server selected by the user will use the same type of database system as the primary. In fact, it is possible for the primary to be a relational database while the backup is an object-oriented database. Moreover, in our scheme, replication will take place on a per process instance basis, i.e., each active process instance has its own primary and backup, and thus, the notion of backup is not tied to the primary site but to each process instance. The actual decision of choosing which server will be the primary and which will be the backup is left to the user. This can also be done automatically using the techniques described in [CS92]. Another alternative to achieve this is to use load balancing algorithms, but this issue is beyond the scope of the paper. The architecture is summarized in Figure 3.

We will assume there is an underlying communication layer that provides the appropriate interface and mechanisms for the clients and servers to communicate. We will further assume that messages do not get lost and arrive uncorrupted. This same communication layer will perform failure detection.

### 4.3 Object State Replication

As discussed earlier, variations of 2-safe policies are to be used to avoid data loss. To reduce the overheads for processes run under this policy, we need to minimize the information exchanged between primary and backup servers. We also need to cope with the fact that the databases may be heterogeneous. Hence we need to replicate information that is significant to any database, regardless of the schema it implements. To accomplish all this, our approach is to send information only about state changes of workflow objects such as activities or connectors, instead of about the data structures over which they are implemented, i.e, we will only replicate the state of workflow entities. Another scheme to avoid data loss is to use the messages exchanged between the primary server and the clients. By replaying (not executing) the messages in the same temporal order at the backup server, the progress of a process can be easily recreated at the backup. However, this would require that the backup perform the navigation itself which may result in greater delays. Also, if the WFMS allows the modelling of time events to trigger activities, then it is not guaranteed that

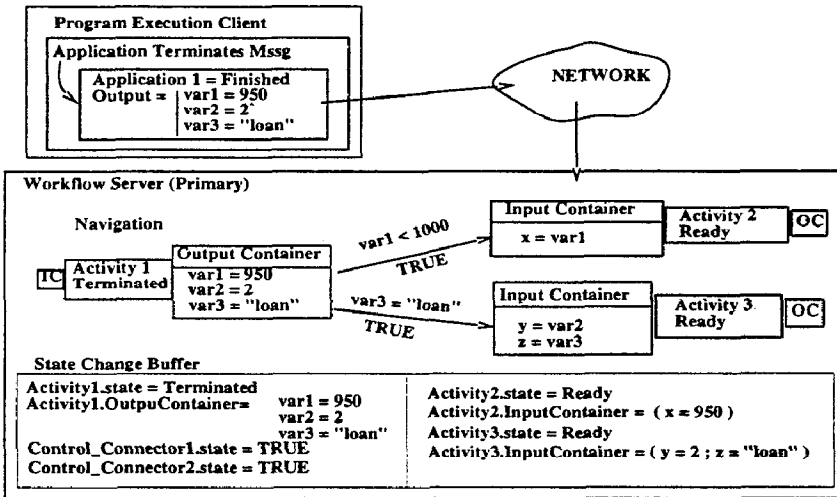


Figure 4: Navigation at the Primary and State Changes

replaying the messages on the backup results in the same workflow execution. Instead of using this, our approach takes advantage of the very clear set of components of a workflow specification and records the state changes in these components. There are only a few of these components that are relevant during the navigation procedures (see Figure 1), and they usually correspond to tangible database entities such as objects or records in a table. Therefore, it is possible to replicate the state of a process by recording the state changes of these entities at the primary database, sending those changes to the backup server and directly updating the state of the entities at the backup database. At the backup server, the state of an activity can be modified directly either through an SQL statement if it is a relational database, or through the corresponding method if it is an object-oriented database. This approach has the advantage of being faster (having reduced overheads) both at the primary and the secondary servers, especially when the number of entities involved is large. It has the disadvantage that a mapping is needed between the database schemas to guarantee that both sides refer to the same activity. These ideas are summarized in Figure 4.

When a client reports that a program has terminated, the server performs navigation as explained in Section 2. For each item - activities, containers, connectors - that changes its state, the new state is recorded in a buffer. When navigation is completed, this buffer contains all the changes that have occurred. This buffer is then sent to the backup. The information in this buffer is stored using the canonical representation explained in the next section. This representation is understood by all servers and can be mapped to the particular database in use.

## 5 Backup Algorithms for Providing High Availability

This section first introduces the data mappings/structures and then describes the backup algorithms.

### 5.1 Data Mappings between Primary and Backup — Canonical Representation

In many cases, the primary and the backup databases will have different schemas. Thus, a mechanism is needed to map entities between both databases. For instance, in object-oriented databases, objects reference each other by their OIDs. However, these OIDs are usually not visible to the user, hence a surrogate identifier must be used to refer to the object [WN94]. Moreover, even if the OIDs of objects were to be available, two databases will assign different OIDs to what is essentially the same

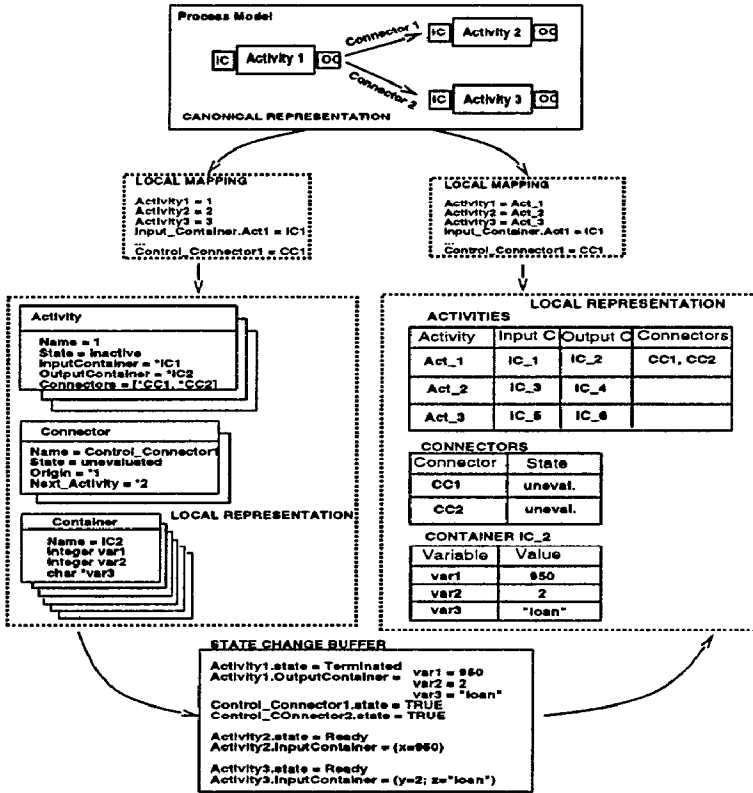


Figure 5: Mappings between the different representations of a process

object, one at the primary and one at the backup database. To solve this problem we rely, once more, on semantic information about the particular process instance.

Usually, in a workflow model only the process schemas/instances and activities have user assigned names. Connectors and data containers exist in relation to a particular activity or process instance, but usually have no names on their own. To be able to uniquely identify each component of the workflow, the system must supply a unique name such as "control connector j". This information is extracted from the definition of the workflow when it is compiled into a database schema. With this information, primary and backup databases can ensure consistency. Such a representation of a process is called a *canonical representation*. Note that it is different from the process model since it contains much less information, it is only a list of agreed upon names for the workflow entities. This canonical representation is chosen to optimize storage and communication overhead. Hence at both the primary and the backup servers a *local mapping* must exist between the canonical representation and the particular format of the database or *local representation*. These ideas are shown in Figure 5, where the *canonical representation* is mapped to hypothetical object-oriented and relational databases by supplying a correspondence to object identifiers or primary keys. Note that the state changes are transmitted using the canonical representation, and therefore will also have to be mapped to the local representation.

## 5.2 Hash Table

To organize the data about messages sent between the primary and secondary, so that it can be easily accessed whenever needed, we use a persistent open hash table

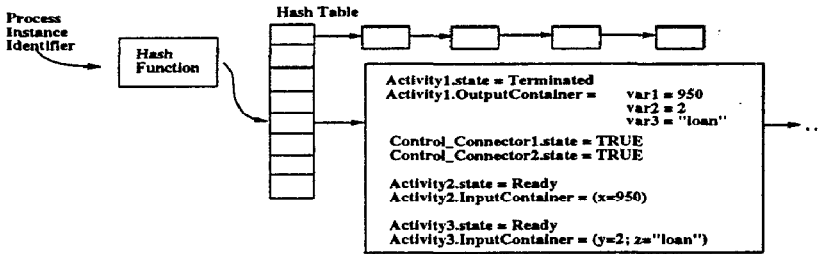


Figure 6: Storage of State Changes in a Hash Table

like the one shown in Figure 6. Each entry in the table corresponds to an executing process instance, and the messages are stored as a linked list. Using a hash table also obviates the need for reorganization as in the case of sequential logs. Such a hash table is maintained both at the primary and the backup workflow servers. When activities complete, these hash tables are updated and when a process completes, the corresponding entries are deleted.

When a failure occurs at the primary database, the backup database becomes the primary database and hence a new backup database must be selected. For important processes (instances), the information to restore processes to the same state they had at the primary server is found in the hash table. For critical processes we also keep the messages received with the state changes in the hash table. Therefore, for both critical and important processes the new backup can be established by sending the information stored in the hash table. Creating a new backup database, on the failure of the existing backup database is handled similarly. To further speed up the setting of a new primary/backup database, it may be possible to use database *loaders* [WN94]. However, their efficient use requires, for instance, to turn logging off, which would conflict with normal operations and many object-oriented databases do not provide such a feature. Hence, it is not yet clear if this is a feasible option. Exact details about how the hash table is maintained and used are provided in the next section while discussing the backup algorithm.

### 5.3 Backup Algorithm

Since we exploit workflow semantics to reduce replication overheads, instead of replicating all information, only the *canonical representation* of a process is replicated. Changes are packed into a buffer which is then sent to the backup server. The algorithms are associated with the following *system events*: *process initialization, normal execution, failure of the primary database, failure of the backup database, backup database creation, and site initialization*.

Process initialization is necessary to ensure that both the primary and the backup servers have the information they need about a process schema before starting to execute it. In existing WFMSs like FlowMark, which have no backup, the definition of a process model is compiled into a database schema. However, if a backup mechanism is in place *this compilation cannot take place directly since at the point a process model is defined we don't know what the local representations (schemas) will be for each instance*. Hence, when a process model is defined, we can only extract the canonical representation for it. If the number of process schemas and the number of platforms are numerous, then extracting the local mappings at compile time may not be efficient. Hence the actual local mappings and representations will be established when an instance is to be executed. The algorithm for this is shown in Figure 7.

During normal execution of a process instance, the algorithm used to replicate data from the primary to the backup is shown in Figure 8. The buffers are filled and

```

Process is defined; (* using buildtime client *)
Canonical representation is created and stored; (* at some server *)
User selects process for execution; (* from runtime client *)
begin PROCESS INSTANCE CREATION;
  Canonical representation and process definition sent to primary and backup;
  Both at the primary and the backup servers:
    Canonical representation mapped to local representation;
    Local mapping generated;
    Entry in hash table created;
end PROCESS INSTANCE CREATION;
Process instance is ready for execution;

```

Figure 7: *Algorithm for Process Initialization*

At the PRIMARY SERVER	At the BACKUP SERVER
At a client, a program terminates; Client sends message to primary server; Primary server performs navigation; Changes are recorded in a buffer; Buffer is stored in hash table; Buffer is sent to backup server;	Backup server receives buffer; Buffer is stored in hash table; If process is <i>critical</i> then update backup database; Ack. sent to primary server
Primary server concludes execution step;	

Figure 8: *Algorithm for Replicating Data during Normal Processing*

sent to the backup when the transactions explained in Figure 1 are executed on the completion of activities. At the primary, the buffers are stored in the hash table so that in the event of a failure of the backup, a new backup can be quickly established by just resending the buffers stored in the hash table for this process instance.

Upon failure of the primary database the backup database is to be used. Since a single database acts as a repository to several workflow servers, a failure of this database does not imply that all workflow servers or clients have also failed. Also note that if the failure occurs only at the workflow server, the problem can be easily solved by starting another workflow server somewhere else and connecting it to the original database. Failure of a particular database is notified by the communication layer to all servers in the system. Upon receiving such a message, a workflow server must determine which of the active process instances in its own hash table were running at the failed database. When the database fails, the workflow servers connected to it cannot perform navigation. However, they are used as relay nodes so the backup server can contact the clients who were connected to the primary server. In this way, the failure of the primary database is transparent to the user. All the data is currently available at a different database at the backup, and messages are rerouted to reach the backup server instead of the primary server. Since the primary and the backup servers know about each other, this rerouting is easy to establish. The algorithm to handle the failure of primary database is shown in Figure 9.

If during normal operation a backup database fails, the primary server must halt the execution of all critical and important processes (instances) until a new backup database is established. The notification mechanism is the same as the one described

Backup server receives message informing of failure;  
 Backup server examines hash table to determine processes to take over;  
 Backup server *creates a new backup database*;  
 Original backup server becomes new primary server (so do the databases);  
 The following happens at the new primary (original backup):  
   For *critical* processes:  
     resume execution;  
   For *important* processes:  
     Retrieve changes from hash table;  
     Apply changes to its own database;  
     resume execution;

Figure 9: *Algorithm for Handling Failure of Primary Database*

Primary server receives message informing of failure;  
 Primary server examines hash table to determine the processes that need  
   a new backup;  
 Primary server *creates a new backup database*;  
 Primary resumes execution;

Figure 10: *Algorithm for Handling Failure of Backup Database*

above. A server must identify which of its process instances were using the failed database as a backup. Each server maintains a table specifying which servers are connected to the backup database for the active process instances running at that server. This table is used for routing the state change buffers when navigation takes place. In case of failures of the backup database, the table is used to determine the affected process instances. The algorithm to handle the failure of backup database is shown in Figure 10.

In case of failures, either of the primary or the backup database, it is necessary to *create a new backup database*. The algorithm to locate a new backup database, initialize it and bringing it up to date is shown in Figure 11.

After a failure, the primary or the backup database will later recover. This may happen while a new primary/backup database is being established. It would be an obvious error to let the recovered database to resume execution because its actions will conflict with whoever has taken over. Hence, when a database recovers from a failure, the corresponding workflow server must check, for each process, whether someone has already taken over. The algorithm for site initialization is shown in Figure 12.

#### 5.4 Garbage Collection

Of all the process categories, the one that will probably be used most is *important*. It provides data backup in case of failures without excessive overheads. The only problem with these processes is the space they require. If, to reduce the overhead at the backup server, changes are only applied when the backup server actually takes over the navigation from the primary server, the amount of state changes accumulated can be considerable. Hence, some garbage collection algorithm is necessary to keep the space manageable. All required information is available in the hash table we mentioned earlier. When a process instance terminates, the entry is discarded from the hash table, thus deallocating the space. The termination of a process instance can be easily detected by examining the state changes sent by the primary server or by having the primary server communicating this fact to the backup server. In either

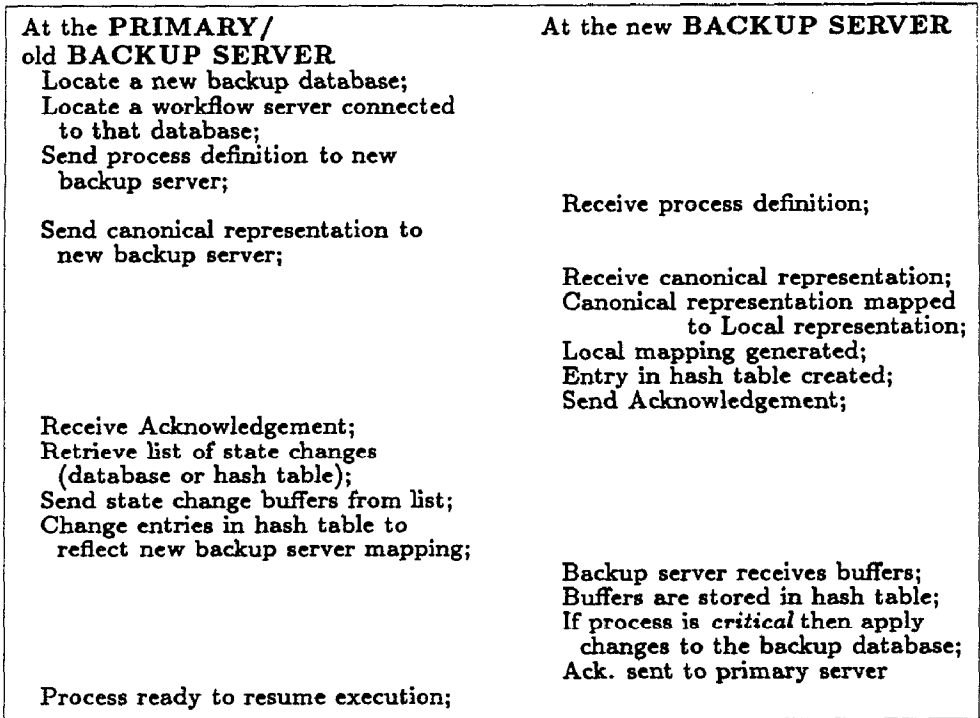


Figure 11: *Algorithm for Creating a New Backup Database*

case, when the process instance has completed its execution, its copy at the backup server is no longer needed. Similarly entries that correspond to terminated processes are discarded from the hash table at the primary server as well. Also, as explained earlier in the algorithm, a recovering server has to discard all the processes whose execution has been resumed elsewhere using a similar procedure.

### 5.5 Analysis and Discussion

While performing data replication, two types of overheads are to be considered — overheads at primary/backup during normal processing and network overheads during normal processing. The most important among these as far as scalability is concerned are network overheads. We have reduced this as follows: By just replicating workflow specific information at the end of each activity instead of replicating log records of changes to each object, we considerably reduce the number of messages transmitted. Since we just ship output results of activities as name-value pairs, the size of the buffer is also reduced and hence the number of messages exchanged per buffer is reduced. We introduced the notion of availability levels and the associated process categories to reduce the overheads during normal processing both at the primary and the backup. Hence we expect workflow administrators/users to take full advantage of this option (especially the “important” category) to get scalability.

We described algorithms for message exchange between the primary and the backup workflow servers to handle failures of the workflow database. Wherever relevant, we addressed the special requirements of the different process categories. The algorithms do not rely on replication features offered by the database and hence allow the use of heterogeneous databases. Since we exploit workflow semantics and just

```

For primary processes active at the moment of failure:
  Locate the backup server of each process instance;
  Query the backup server for status;
  If backup server has taken over
    then Discard the process instance;
  else Notify backup server;
    Resume execution;
For backup processes active at the moment of failure:
  Locate the primary server of each process instance;
  Query primary server for status;
  If primary server has new backup
    then Discard the process instance;
  else Notify primary server;
    Resume execution;

```

Figure 12: *Algorithm for Site Initialization*

replicate the canonical representation, the replication overheads are minimized. Our backup technique is different from checkpointing schemes of fault-tolerant systems since we also have provision to handle failures at backup (which is usually ignored in fault-tolerant systems). Our algorithms are quite general and do not contain references to FlowMark-specific features and hence can be used to backup process data in any WFMS.

## 6 Conclusions

In the future, organizations will increasingly rely on WFMSs for their daily business activities. To make workflow technology successful, WFMSs should be able to work with heterogeneous systems and at the same time provide high availability. WFMSs are extreme example of applications where due to the environment in which they are embedded, traditional database approaches are not suitable to provide backup across heterogeneous systems and for replicating data to provide high availability. Hence we have proposed a set of new techniques that exploit workflow semantics to meet these objectives.

In this paper we introduced the notion of availability levels in the context of WFMSs and described a backup architecture and the necessary algorithms for efficiently maintaining backups of process data. The backup mechanism is based on the idea of replicating workflow information as opposed to replicating database structures. This allows us to implement backup over heterogeneous systems, i.e., over databases with different schemas and of different nature. The cost of replicating process instances is minimized by exchanging information only about changes that are relevant. Hence our algorithms can scale to very large WFMSs. By carefully analyzing the workflow semantics/requirements and comparing tradeoffs we have provided a practical solution for providing high availability in WFMSs. Future work includes a complete evaluation of some of the tradeoffs as well as a study of their impact on the overall performance and a prototype based on FlowMark.

## Acknowledgements

This work was partially supported by funds from IBM Networking Software Division and IBM Software Solutions Division. We would also like to thank Prof. Divyakant Agrawal and Prof. Amr El Abbadi for their help in formulating some of the ideas described in this paper. Additional information about the Exotica project can be found at the URL <http://www.almaden.ibm.com/cs/exotica/>

## References

- [BGHJ92] A. Bhide, A. Goyal, H Hsiao, and A. Jhingran. An Efficient Scheme for Providing High Availability. In *Proc. of 1992 SIGMOD International Conference on Management of Data*, pages 236–245, May 1992.
- [BT90] D. L. Burkes and R. K. Treiber. Design Approach for Real-Time Transaction Processing Remote Site Recovery. In *Proceedings of IEEE Compcon*, pages 568–572, 1990.
- [CS92] D. D. Chamberlain and F. B. Schmuck. Dynamic Data Distribution ( $D^3$ ) in a Shared-Nothing Multiprocessor Data Store. In *Proceedings of 18th VLDB Conference*, pages 163–174, Vancouver, British Columbia, 1992.
- [FCK87] J. C. Freytag, F. Cristian, and B. Kaehler. Masking System Crashes in Database Application Programs. In *Proceedings of 13th VLDB Conference*, pages 407–416, Brighton, England, 1987.
- [Fry94] C. Frye. Move to Workflow Provokes Business Process Scrutiny. *Software Magazine*, pages 77–89, April 1994.
- [GMP90] H. Garcia-Molina and C. A. Polyzois. Two Epoch Algorithms for Disaster Recovery. In *Proc. of 16th VLDB Conference, Brisbane, Australia*, pages 222–230, 1990.
- [GHS95] Georgakopolous D. and Hornick M. and Sheth A. An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. *Distributed and Parallel Databases Journal*, 3(2):119–152, 1995.
- [Gol94] R. Goldring. A Discussion of Relational Database Replication Technology. *InfoDB*, 8(1), 1994.
- [GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, CA, 1993.
- [Hsu95] M. Hsu. Special Issue on Workflow Systems. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 18(1), March 1995.
- [IBMa] IBM. *FlowMark for OS/2: Managing Your Workflow*. Document No. SH19-8176-00, May 1994.
- [LR94] F. Leymann and D. Roller. Business Processes Management with FlowMark. In *Proc. 39th IEEE Computer Society Int'l Conference (CompCon), Digest of Papers*, pages 230–233, San Francisco, California, February 28 – March 4 1994. IEEE.
- [Lyo90] J. Lyon. Tandem's Remote Data Facility. In *Proc. of IEEE Compcon*, 1990.
- [MAGK95] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica: A Research Perspective on Workflow Management Systems. In [Hsu95].
- [MHL<sup>+</sup>92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1), 1992.
- [MTO93] C. Mohan, K. Treiber, and R. Obermarck. Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery. In *Proc. of 9th International Conference on Data Engineering*, pages 511–518, 1993.
- [SBCM95] Samaras, G., Britton, K., Citron, A., Mohan, C. Two-Phase Commit Optimizations in a Commercial Distributed Environment. In *Distributed and Parallel Databases Journal*, Vol. 3, No. 4, October 1995.
- [She94] A.P. Sheth. On Multi-system Applications and Transactional Workflows, 1994. Collection of papers from Bellcore.
- [WfMC94] The Workflow Reference Model. Workflow Management Coalition, December 1994. Accessible via: <http://www.aiai.ed.ac.uk/WfMC/>.
- [WN94] J. Wiener and J. Naughton. Bulk Loading into an OODB: A Performance Study. In *Proceedings of the 20th VLDB Conference*, pages 120–131, Santiago, Chile, 1994.