Modeling, Correctness & Systems Issues in Supporting Advanced Database Applications Using Workflow Management Systems [†]

Mohan Kamath and *Krithi Ramamritham*
Computer Science Technical Report 95-50
Department of Computer Science
University of Massachusetts
Amherst MA 01003
{kamath,krithi}@cs.umass.edu

Abstract

Advanced database applications like office automation, CAD/CAM and Software Engineering are characterized by the presence of long-duration, cooperative or multidatabase tasks. These applications can be data-centric or process-centric or both. Extended transaction models designed to address the requirements of these applications use a data-centric approach and they do not have adequate implementation support. On the other hand, workflow management as promoted by industry uses a process-centric approach. However, due to lack of concrete guidelines, they contain customized features for modeling and executing applications. While they certainly have many practical features that extended transaction models do not provide, they do not have adequate support to satisfy the modeling & correctness requirements of advanced applications. To date, no systematic studies have been undertaken to design proper support for these applications. This paper attempts to fill the lacuna by unifying the workflow and extended transaction model based approaches in an effort to meet the needs of these applications. By studying the requirements of different applications and the features provided by current workflow systems as well as advanced transaction models we identify the modeling features essential for supporting advanced database applications. We then provide an implementation architecture and discuss mechanisms for adequately supporting the identified modeling features.

Keywords: Advanced Transaction Models, Workflow Management, Cooperation, Office Automation, CAD/CAM, Software Engineering, Correctness, CSCW

† Supported by the National Science Foundation under grant IRI - 9314376.

Contents

1 Introduction **1**

2 Modeling & Correctness Requirements of Applications **2**
2.1 Office Automation 2
2.2 CAD/CAM and CASE 4

3 Workflow Management Systems **5**
3.1 Modeling and Execution Support 5
3.2 Standardization & Interoperability 6

4 Advanced Transaction Management **7**
4.1 Analysis of Advanced Transaction Models 7
4.2 Environments to Implement Advanced Models 9

5 Proposed Model **9**
5.1 Supporting Coordinated Tasks 9
5.2 Supporting Cooperative Tasks 10
5.3 Handling Interoperability & Heterogeneity 11
5.4 Discussion 11

6 Systems Issues **12**
6.1 Proposed Architecture 12
6.2 Supporting Coordinated Tasks 13
6.3 Supporting Cooperative Tasks 16
6.4 Other Features 17
6.5 Discussion 18

7 Conclusions **18**

1 Introduction

In the last decade, there has been growing interest in advanced database applications like office automation, CAD/CAM and software engineering. These applications are characterized by the presence of long-duration, cooperative or multidatabase tasks. Two approaches have emerged to tackle the needs of such applications.

Adopting a *data-centric* approach, several ***advanced transaction models*** have been proposed [Elm92] that relax the ACID (Atomicity, Modeling, Isolation and Durability) properties of *traditional* transactions. By exploiting the semantics of the applications and using relaxed correctness criteria, these advanced models provide special features to handle concurrency control and recovery. But only recently have initiatives been taken to implement these models [GHKM94, BDG$^+$94].

Adopting a *process-centric* approach, industry has been promoting *workflow management* as a technique for modeling, executing and monitoring such applications[1]. Aided by advances in client-server computing and distributed database techniques, early office information systems evolved into workflow management systems. While several prototype and commercial workflow management systems are available, due to lack of proper standards and guidelines, each provides its own set of features for modeling and execution. Though they have many features to address the needs of real working environments that extended transaction models fail to consider, they don't have adequate support to satisfy the modeling & correctness requirements of advanced applications. The deficiencies include lack of support to keep track of data dependencies between different workflows, lack of support to control concurrent accesses to objects managed by non-transactional resource managers[2], lack of support for cooperative activities, and insufficient support for recovery.

Even as researchers are trying to implement extended transaction models, workflow management systems are evolving, primarily driven by industry. Unfortunately, very little has been done to study the modeling/correctness aspects of workflow management systems and combining (process-centric) workflow management with (data-centric) advanced transaction management. This paper attempts to close the gap between the two approaches and provide guidelines essential for the development of flexible and better workflow management systems. Recently, the authors of [Geo95] have provided a broad overview of the state-of-the-art in workflow management. They have also discussed at a high-level how some of the limitations may be overcome by the use of extended transactions. Our contributions are complementary:

- We identify the modeling features essential for supporting advanced database applications by studying requirements of different applications, features provided by current workflow systems and advanced transaction models.
- We describe an implementation architecture and extensively discuss mechanisms to support the identified modeling features (including those for concurrency control and recovery) in the context of this architecture.

Thus our work also tries to address some of the concerns raised in [DGMH$^+$93]. In our discussion we

[1] A procedural description of how and what is to be performed to achieve work in termed as a *workflow* or a *task*. The individual steps that comprise a workflow are termed *activities*. Activities may involve humans as well as programs.

[2] A resource manager like a file system that does not provide concurrency control and recovery facilities.

will not be addressing other workflow management issues like security, monitoring and simulation.

The rest of the paper is organized as follows: Modeling and correctness requirements of different advanced database applications are studied in depth in section 2. Section 3 surveys some of the workflow management systems in an effort to understand their limitations in modeling and executing applications. In section 4 we study some of the advanced transaction models and execution environments. This helps us in identifying interesting concepts that can be used in workflow management systems. Essential modeling features for workflow management systems are identified in Section 5. Section 6 discusses an implementation architecture and mechanisms to support the modeling features. Section 7 concludes the paper.

2 Modeling & Correctness Requirements of Applications

We study a small but representative application suite consisting of Office automation, CAD/CAM, and CASE.

2.1 Office Automation

Figure 1 depicts the flow of control between activities in an office automation task for processing a health insurance application (to reduce congestion, the flow of data is not shown). Activities access resources from multiple sites and hence *heterogeneity* and *interoperability* are important issues. Some of the activities themselves contain a task, resulting in *nested* tasks. As shown in the figure, there are conditional executions based on the outcome of activities (termed *value* dependencies [DE89]). During travel planning [WR92], only certain combinations of plane-car-hotel reservations are permitted. To achieve this, a set of activities are grouped into an *atomic-unit*. The telecommunication applications for service order provisioning [ANRS92] mostly contain activities which are transactions that in turn can contain subtransactions with execution dependencies. The loan approval application considered in [BDS$^+$93] exemplifies inter-task data dependencies. Among other activities, a task contains a *risk-evaluation* and a *risk-update* activity. To maintain correctness of execution, while a pair of tasks are executing concurrently, the risk-evaluation activity of a second task can execute only after the risk-update activity of the first task. Consider another application, reviewing bids from contractors for projects in a consulting company. The activity *review-bid* is usually done in parallel for each bid received. Since the number of bids received cannot be predicted, rather than specifying it statically at task definition time it should be possible to dynamically determine the number of instances of the review-bid activity to be executed in parallel.

Role modeling and *task-definition updates* must also be provided for. The person responsible for executing a human activity is usually specified by a role rather than by name. Hence the organization hierarchy is to be modeled using staff and roles names. Organizational changes require changes to staff definitions. When staff definitions are changed, activities that are already being worked on by a staff whose role has changed are to be handled properly such that inconsistencies are not caused. To accommodate changes in task procedures, task definitions are modified periodically. While some applications prefer that the new definition be used only for fresh instances, others require that even the executing instances of the task follow the new definition in which case care should be taken to ensure consistency and correctness.

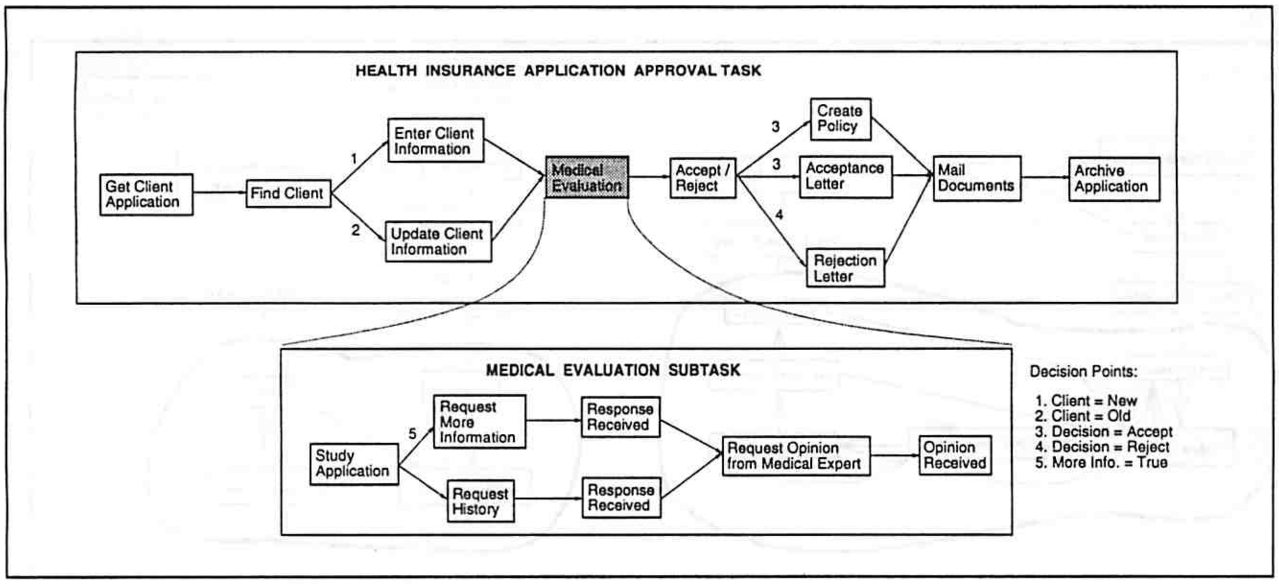

Figure 1: *Office Automation Example*

Next we discuss *failure handling* requirements for applications. Failures are of two types — *system* and *logical* failures. In the case of system failures, most applications require that the task be restored to the most recent consistent state (forward recovery) so that work performed over long-durations is not lost. To handle logical failures, usually an alternative action is taken by annulling the effects of the executed activities using *compensating activities*. Here again if activities are nested, then some applications require that the parent activity be compensated with a single action whereas others require that each of the child activities be compensated individually. If compensating activities cannot be specified then locks are to be held for the required duration of atomicity & isolation. When it is difficult to specify compensating activities or when system failures are too complex to handle (so that a task cannot be restored to a consistent state), tasks are *dynamically modified* (individually) by administrators to restore consistency.

From the above discussion it can be clearly seen that due to the diverse nature of office automation tasks there are a variety of requirements. We can summarize the following as the necessary features to support office automation tasks (to specify task structures, consistency of data and correctness of execution):

- **Task Modeling**
  - ▷ data flow between activities
  - ▷ execution dependencies between activities (control flow)
    - □ value based
      - ◇ conditional execution between activities
      - ◇ multiple invocations of an activity
    - □ state based (for transactional[3] activities)
    - □ time based
  - ▷ grouping one or more activities into an atomic-unit

[3]Transactional activities are those whose commit and abort events are visible outside. The resources are accessed from a transactional resource manager.

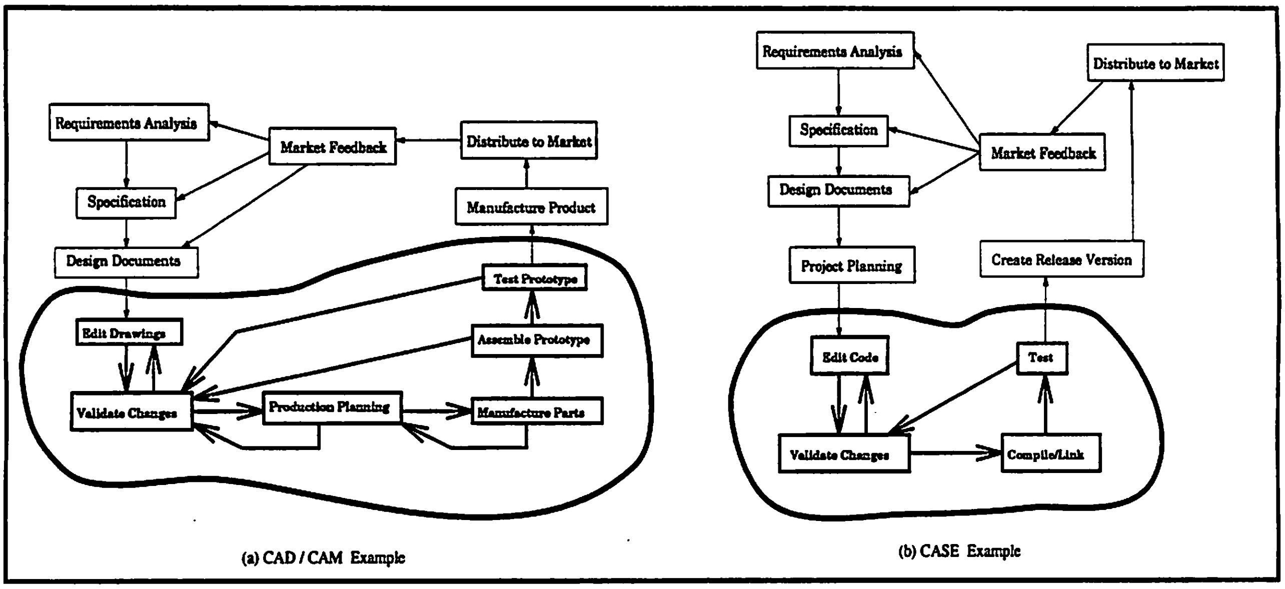

Figure 2: *CAD/CAM and CASE Examples*

- ▷ **nested activities**
- ▷ **task completion (set of final activities)**
- ▷ **entity responsible for performing activity**
  - ☐ **application program**
  - ☐ **human role (staff definition & organizational hierarchy)**
- ▷ **modifying task definition**
  - ☐ **applying new definition to future invocations only**
  - ☐ **applying new definition to currently executing instances also**

- **Task Execution**
  - ▷ **handling inter-task data dependencies**
  - ▷ **multi-system accesses**
    - ☐ **interoperability**
    - ☐ **heterogeneous data formats**
    - ☐ **coordinating access to object at non-transactional resource managers**
  - ▷ **system failures**
    - ☐ **forward recovery**
    - ☐ **dynamic modification of task (exception handling)**
  - ▷ **logical failures**
    - ☐ **applying compensating action and following alternate path**
    - ☐ **dynamic modification of task (exception handling)**

2.2 CAD/CAM and CASE

CAD/CAM applications usually consist of designers cooperating to design and manufacture products of different types while CASE applications consist of software developers cooperating to design, code and test software. Since the requirements of CAD/CAM and CASE are similar, we discuss both the areas in this subsection. Generic examples of CAD/CAM and CASE applications are shown in figure

2. In both the examples, the highlighted areas in the figure show activities which are to be executed in a cooperative manner while the rest requires coordination as in the case of office automation tasks. Design activities usually tend to be in groups and hence tasks are divided into subtasks, to form a *hierarchy*. Note that these hierarchies correspond to the ***composition hierarchy*** of the objects being designed and hence are different from the nested tasks discussed in the context of office automation. The difference is that here an activity at the higher level is complete only if *all* the activities at the next (lower) level are complete. As can be seen from the figure, drawings (objects), ideas, queries and feedbacks are exchanged in a dynamic (unpredictable) manner between the various subtasks and activities. Also some of the designs are done iteratively and hence there must be provision to backtrack to a previous step (not global backtracking as in the case of *checkpointing*) to take a different approach from that step. It is not possible to specify all these interactions and design iterations in a *static* task definition. However there should be means to ensure that such a dynamic interaction does not produce inconsistencies in the data accessed. In most cases it is also required that if an object is changed deep down in the composition hierarchy, then checks are to be performed at successive higher levels to verify if additional changes are required (changes have to percolate to higher levels). Details of these changes are also to be reflected in the product manuals (external use) and design documents (internal use).

Based on the above details, it is easy to identify the following *additional* features (apart from those identified earlier) needed to support CAD/CAM & CASE tasks:

- hierarchy of activities
- controlling the interleaving of object accesses
- facility to perform controlled backtracking
- facility to propagate changes

Additional discussion on correctness issues related to long-lived activities can be found in [MP90, KS90, BK91, KP92].

3 Workflow Management Systems

In this section we will enumerate some of the salient features provided by current workflow management systems for task modeling and managing task execution. This study will help us identify some of the limitation of these systems. We will focus only on systems that handle *production* workflows [McC92] since they provide better modeling and execution support than the rest.

3.1 Modeling and Execution Support

We first review prototype systems. APRICOTS [Sch93] is a prototype implementation of ConTracts where a task/process (ConTract) is defined using sequential, parallel, branched or nested blocks (set of steps) that can be enclosed in a transaction. Suspend, resume, activate, compensate and restart functions are provided. Concurrent access of global data from different tasks is controlled using *invariants* (predicates that check constraints). Forward recovery is provided via check-pointing of blocks. The METEOR project [Kri95] has built a system for specifying and executing multi-system workflows that may consist of both transactional and non-transactional tasks. It uses a two level approach to specify and execute tasks — one at the workflow level (to control inter-task execution dependencies) and one at the task level (to control local programs). Forward recovery and dynamic workflow modification

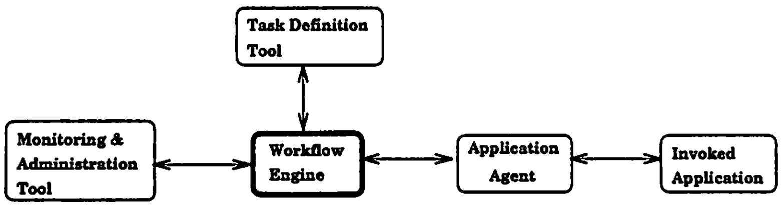

Figure 3: *Architecture of Workflow Management System*

are also supported. Other models and approaches for coordinating activities to handle workflow type applications have been proposed in [DHL90, DHL91, GMGK+90].

Turning our attention to commercial workflow systems like FlowMark[4] [FD94], InConcert[5] [MS93] and others [Hsu93, Hsu95], the following is a list of features provided by them. They have been listed in the same manner as the application requirements (see Section 2) for ease of comparison.

- **Task Modeling**
  - ▷ data flow between activities:
    - ☐ implicitly passing all data to all activities
    - ☐ explicitly mapping data between activities
    - ☐ controlled by check-in/check-out
  - ▷ control flow between activities:
    - ☐ value based *(activity started only after preceding ones complete)*
      - ◇ conditional execution between activities
      - ◇ multiple invocations of an activity
  - ▷ nested activities
  - ▷ correctness conditions for activities to start and finish
  - ▷ entity responsible for performing activity
    - ☐ application program
    - ☐ human role (staff definition & organizational hierarchy)
  - ▷ modifying task definition
    - ☐ applying new definition to future invocations only
- **Task Execution**
  - ▷ email or database for transfer of data/control
  - ▷ application client/agents to interact with entity responsible for performing activity
  - ▷ system & logical failures
    - ☐ forward recovery
    - ☐ dynamic modification of task (exception handling)

3.2 Standardization & Interoperability

In an attempt to standardize the features provided by workflow management systems, the ***workflow management coalition*** (WfMC) has developed a reference model [Mem94]. Its objective is to provide interoperability between different workflow products and hence the main focus is on (i) describing a common task definition model and (ii) describing components and interfaces for interoperability. The task modeling features resembles the list of features we just enumerated. A generic architecture of

[4]From IBM, Germany
[5]From Xerox

a workflow management system based on the reference model is shown in figure 3. The application programs that perform work for the various activities are usually managed by several ***application agents.*** The ***workflow engine*** performs the navigation through the instances of task structures and communicates with the application agent to perform the activity. To allow interoperability between different workflow systems, several scenarios are also defined — process instances or activities to be transferred between different workflow systems (*i.e.* a process instance started in one system continues on another), tasks in one system to be executed on behalf of an activity in another system (nested remote tasks), composing activities from different systems and finally synchronization of activities across different systems. We return to this later when we discuss systems issues.

From our discussions thus far, it should be clear that workflow systems lack support for maintaining data consistency in the case of concurrent coordinated tasks and provide very minimal support for modeling and executing cooperative tasks. Some of these limitations have also been stressed in [Geo95]. A recent panel on workflow automation [DHM+95] concluded that modeling requirements for long-running processes are complex and additional efforts are needed to address them.

4 Advanced Transaction Management

In this section we analyze some of the extended transaction models and execution environments. Apart from analyzing the ability of these models to themselves model advanced applications, this will help us determine ideas and concepts, especially from concurrency control, recovery and commit processing perspective that can be used to improve modeling and execution support provided by workflow management systems.

4.1 Analysis of Advanced Transaction Models

As a first step towards adapting concepts of advanced transaction models for workflow management, we have attempted to classify these models into groups which have some common applicability. Due to the nature of the models, some of them tend to fall in one or more groups and we have identified such cases. To our knowledge there have been no attempts to perform such a classification of advanced transaction models and this is one of the secondary contributions of our work. Before we get into the details of our approach to classify these models, we list some of the advanced transaction models:

- *Extended Models:* Nested Transactions [Mos81], Multi-Level Transactions [BSW88], Open nested transactions [WS92]
- *Relaxed Models:* Sagas [GMS87], Nested Sagas [GMGK+91], Flexible Transactions [ELLR90, MRB+92, ZNBB94], Split-Join Transactions [PKH88] , Poly Transactions [SRK92]
- *Cooperative Models:* Cooperative CAD Transactions [BKK85], Transaction Groups & extensions [FZ88, Ska89, NZ90], Group Oriented CAD Transactions [KSUW85], Participant Transactions [Kai90]

Nested transactions was one of the first advanced models and it proposed hierarchical structuring of transactions. This hierarchical structuring concept has been used later in many models like multi-level transactions and open nested transactions for exploiting semantics of applications and improving concurrency. Hence all these fall into the category of ***transaction models with hierarchical structures***

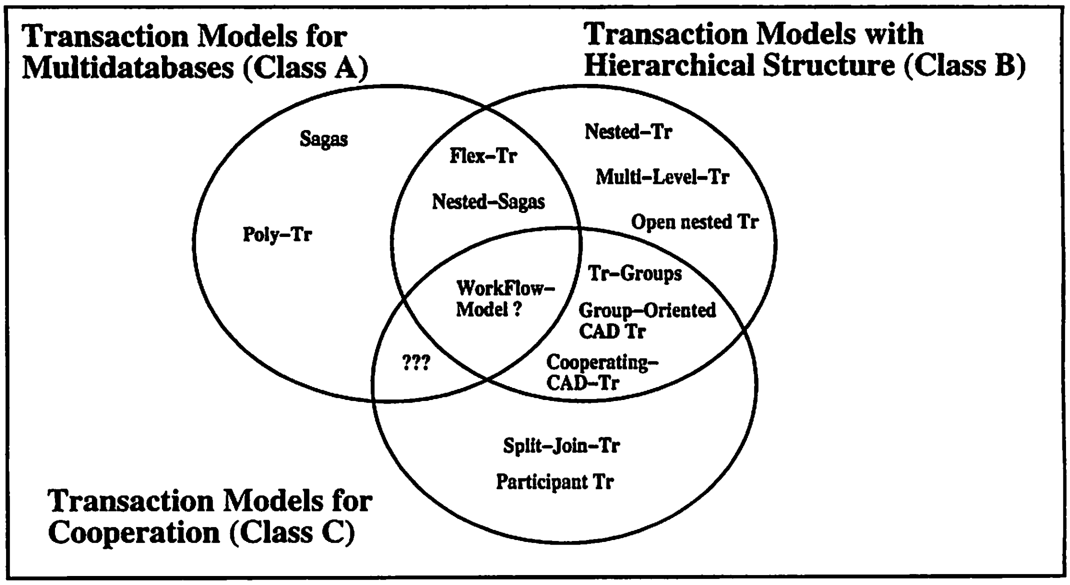

Figure 4: *Taxonomy of Advanced Transaction Models*

(Class B) as shown in in figure 4. Split- and join- transactions were then proposed to allows transactions to pass uncommitted objects to other transactions. Since it is useful for cooperative tasks, this model was later adapted in participant transactions to suit the requirements of software engineering. These models fall into the category of ***transaction models for cooperation*** (Class C). Newer models for cooperation have also been proposed that used hierarchical structures in some form and also allow selective transfer/access of uncommitted objects between transactions. They fall into the intersection of the two classes and they include cooperative CAD transactions, transaction groups, group-oriented CAD transactions. To handle the autonomy requirements of multidatabase systems, newer transaction models have been proposed like the polytransactions model. Though sagas was originally designed for long-duration transactions it can be used for multidatabase applications [BST92]. Hence we have established another class for these models called ***transaction models for multidatabases*** (Class A). Nested Sagas is a hybrid model that was developed by combining sagas and nested transactions. Flex transactions was designed in the context of multidatabases applications to allow hierarchical structuring of tasks (subtransactions) with provisions to compensate and try an alternate (contingency) subtransaction if a particular subtransaction fails. Both these models come under the intersection of classes A & B.

Workflow systems must have provisions to define hierarchical tasks, facilitate coordinated access to multiple systems, and should also allow controlled transfers/accesses to uncommitted objects between users to support cooperative tasks. Hence the workflow model has been placed at the intersection of all the classes. Using basic primitives from models in the different classes, it is possible to build complex applications. A discussion on this is beyond the scope of the paper but details of achieving this using the ACTA framework can be found in [CR94]. Workflows that use features of advanced transaction models (structure and correctness requirements) are categorized as ***transactional workflows*** [SR93]. Advantages and disadvantages of using some of these models to develop advanced applications are presented in [Moh94].

4.2 Environments to Implement Advanced Models

ASSET [BDG+94] provides linguistic primitives to program extended transaction models. Other than the traditional primitives like *begin*, *commit*, *abort* and *wait*, it provides primitives based on the ACTA model such as *delegate* and *permit* that can be used to relax isolation requirements between tasks. In contrast, TSME [GHKM94] uses a toolkit based approach to specify & construct application specific extended transaction models. Transaction model designers provide new models and application programmers can write applications that use the models that are defined and supported. Other approaches to implement advanced transaction models have also been suggested in [US92, ST94]. A preliminary approach to integrate a semantic transaction manager and workflow manager has been suggested in [BDS+93] where the former treats an entire workflow as a single transaction whose semantic properties are to be exploited whereas the latter treats each task as an independent transaction and satisfies the dependencies between the transactions. Advanced transactions models have been used to support workflow applications [ANRS92], where flexible transactions have been developed to integrate multiple independently developed applications.

In summary, extended transactions provide features to handle multi-system, hierarchical and co-operation tasks. Some of these features can be incorporated into workflow management systems to provide better modeling and correctness primitives and we discuss this in the next section.

5 Proposed Model

Based on our study of the application requirements and the features provided by workflow management systems and advanced transaction models, we have analyzed and recommended modeling features necessary for workflow systems. For each feature, we discuss the options (if any) needed to flexibly specify the application requirements and the desired degree of correctness. Our objective is to group together all the essential features, some of which are novel, while others are already available in various systems.

5.1 Supporting Coordinated Tasks

Workflow systems provide good modeling features to handle most requirements in this category. Since they do not address consistency, correctness and recovery issues, we have added some features. Control flow (execution dependencies) in workflow management typically allows a task to start only after the previous one has finished. Advanced transactions provide more options that are useful for certain applications and hence we have included them. Inter-task dependencies and task-definition modification issues are handled better in our model. Modeling primitives should be provided to specify the following:

- **data flow** between preceding-activity and succeeding-activity. Options are to be provided to *implicitly* route task data to all activities (Read/Write mode specified) or *explicitly* specify routing (for security reasons) or allow *checkin/checkout* based user access.
- **control flow** (execution dependencies) between preceding-activity and succeeding-activity. Options are to be provided to specify them based on *value* (for conditional or multiple invocations), *state*

(only for transactional activities — commit-commit, commit-abort, etc.), *outcome* (success or failure of an activity) and *time* (temporal-start, etc.).

- **nested tasks:** Options are to be provided to specify recovery action — if the task is to be compensated by a *single* compensating action or *multiple* compensating actions are to be executed, one for each child.
- **atomic-units of activities:** Guidelines are to be provided on *which* activities can be chosen to form a atomic-unit (*e.g.* activities that form a sequence in the task, or a random selection) and if *overlap* of atomic units is permitted.
- **inter-task dependencies:** Options are needed to specify if the *whole task* or a *specific set of activities* are to be serialized.
- **correct execution of task** — defining the *start* and *finish* activities or *conditions on execution states* in case of transactional activities. Options are to be provided to specify is compensation is required for *all, some* or *none* of the completed activities.
- **correct execution of activity** — defining the *start* and *finish* conditions and *what* action to take if the conditions are not satisfied.
- **entity responsible for executing** an activity — a *role* or a *program* or both.
- **task-definition modification:** Options are to be provided to specify if the definition is to be applied for *future* instances only or for *current* instances as well.
- **staff-definition** and **organizational hierarchy**. This defines the *role* names to be specified for the entity responsible to execute an activity
- **handling system failures:** By default, *forward-recovery* is to be provided so that the task is restored to a consistent state. *Dynamic modification* of task instances are to be allowed to handle exceptions.
- **handling logical failures:** This is to be handled by specifying compensating activities. If it can not be specified, *dynamic modification* of task instances should also be allowed.

5.2 Supporting Cooperative Tasks

Support for cooperation is clearly inadequate in workflow systems. Most of the primitives provided here have their origins in the area of advanced transaction management where considerable work has been done to support cooperative tasks. Since the notion of *hierarchy*, *correct execution* of an activity and *work performed* by an activity is different from that of coordinative tasks, we define features for each of them. Features are also provided to meet the special requirements for managing object accesses, change propagation, selective backtracking and hierarchy modification. *Additional* modeling primitives should be provided to specify the following:

- **hierarchy of activities:** The hierarchy specifies what activities exist at each level. (Activities at a level are usually executed in parallel).
- **work performed** by each activity. a object (*e.g.* a *module* or a *component*) that is to be designed is specified for each activity.

- **correct execution of activity** by defining a *test-specification* that is to be satisfied.
- **managing concurrent object accesses:** Options are provided to use one of:
  - ▷ **protocols:** using information about *state* of object access and *role* responsible for creating that state. (similar to automata that govern object accesses in [NZ90])
  - ▷ **semantic primitives:** like *form_dependency* & *permit*, as described in ASSET.
- **selective backtracking** on errors (handling logical failures). This is essential if protocols are used to control object accesses. (handling logical failures). Based on the history of object accesses all designers who have accessed objects on the backtracking path will be notified.
- **event based action:** based on the occurence of specific events, actions like *notification* or *triggering an activity/task* can be executed. This will facilitate propagation of object changes.
- **hierarchy modification:** (similar to task-definition modification). *e.g.* changing test specifications or adding activities at any level.

5.3 Handling Interoperability & Heterogeneity

Important requirements for applications that access data from multiple systems include *interoperability* between workflow systems and handling *heterogeneous* data formats of applications. In situations where tasks/activities are coordinated across multiple workflow systems, several issues relating to data flow, control flow, system & logical failures can arise. We have provided features and options that address these issues. Another factor often disregarded by workflow systems is access to non-transactional resource managers (*e.g.* file systems that manage spreadsheets, word-processing documents). Data consistency requirements have to be clearly specified if concurrent access to object are allowed. *Additional* modeling primitives should be provided to specify the following:

- **interoperability between workflow systems:** Since control crosses system boundaries, options are to be provided for defining the following *across boundaries* — *all* or *selective* data transfer, *execution dependencies*, *task-definition modification*, *system* and *logical failures.*
- **heterogeneous application data formats:** Options should be provided (in activity definition) to specify the *data exchange standard* (*e.g.* OLE) to which the data must be converted before it is provided to the next activity.
- **non-transactional resource managers:** Options are to be specified if objects accessed from such resource managers should be *locked*, *when* and *how* changes and compensating actions are applied to the objects.

5.4 Discussion

We have identified the essential modeling features to be provided by workflow systems to support different types of advanced applications. We believe we have provided sufficient flexibility to specify task structures and the desired correctness. Primitives mentioned here can be combined to develop large complex applications, but they have to be used based on proper guidelines (care should be taken such that *incompatibilities* do not occur) and the task definition-tool/compiler should enforce this at task definition time. For example, care should be taken when primitives meant for coordinated and cooperative tasks are combined. Due to space limitations a detailed discussion on this is beyond the scope of the paper.

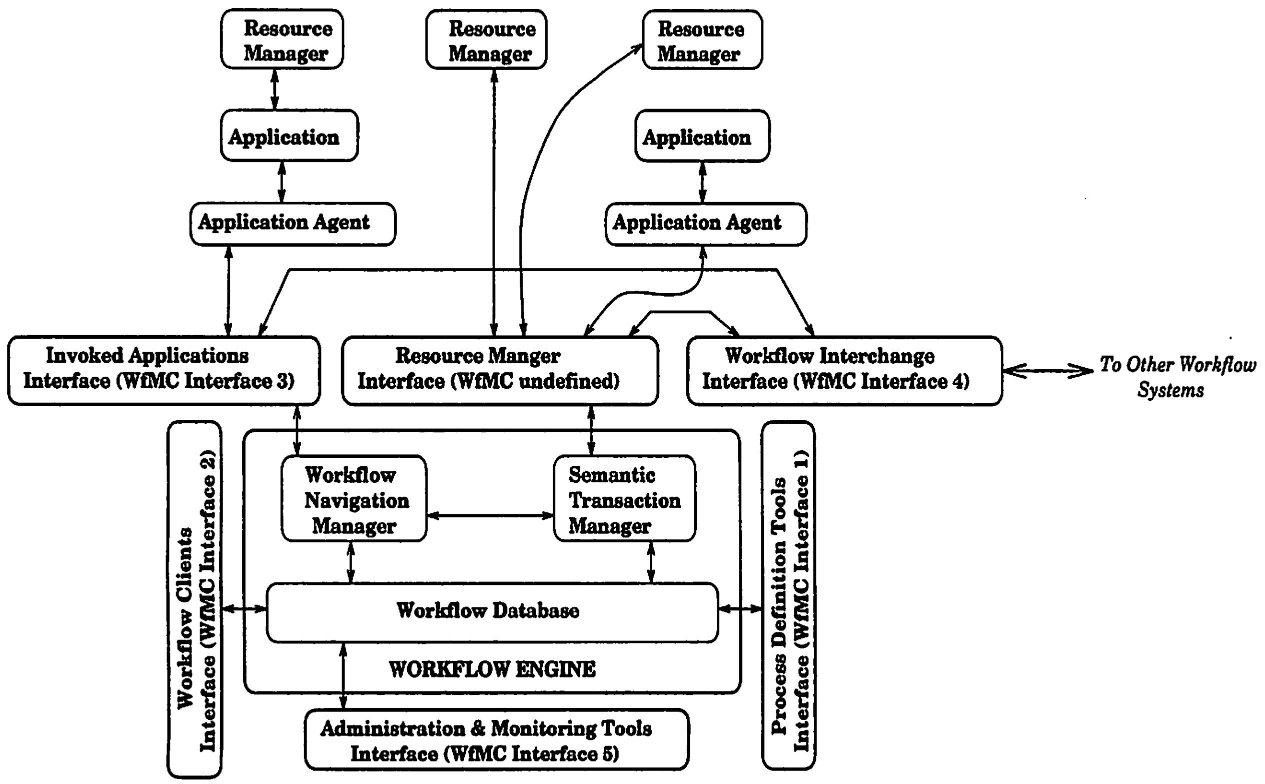
Resource Manager
Resource Manager
Resource Manager
Application
Application
Application Agent
Application Agent
Invoked Applications Interface (WfMC Interface 3)
Resource Manger Interface (WfMC undefined)
Workflow Interchange Interface (WfMC Interface 4)
To Other Workflow Systems
Workflow Clients Interface (WfMC Interface 2)
Workflow Navigation Manager
Semantic Transaction Manager
Workflow Database
WORKFLOW ENGINE
Process Definition Tools Interface (WfMC Interface 1)
Administration & Monitoring Tools Interface (WfMC Interface 5)

Figure 5: *Workflow Manager Architecture*

6 Systems Issues

In this section we first present an implementation architecture that has been designed to support the model we discussed in the last section. Based on this architecture, we then discuss schemes needed to support each modeling primitive. Concurrency control, recovery and commit processing issues are discussed when they arise.

6.1 Proposed Architecture

The *workflow navigation manager* (WNM) plays an important role in a workflow system by handling task execution. While the execution dependencies here are typically based on results produced by an activity, other applications utilize execution dependencies based on state. We have also seen that to control concurrent object accesses in cooperative applications, semantic primitives like form_dependency and permit are useful. These features are better handled by a *semantic transaction manager* (STM). Hence to ensure proper support for all the modeling features, our *workflow engine* architecture contains both a workflow navigation manager and a semantic transaction manager. Applications using semantic primitives do not directly communicate with resource managers, instead the requests pass through the STM. Though preliminary ideas towards this approach have been presented in [BDS+93], there are several other important issues which we discuss in this section. To facilitate forward recovery and ensure consistency of task definition and process instance data (status and results of activities/tasks) at all times, a *workflow database* (WDB) is necessary to store the meta-information. To manage task executions, the workflow navigation manager and the semantic transaction manager consult this database and provide appropriate instructions to the application-agents and local resource managers respectively. Data belonging to completed tasks are deleted from this database and archived periodically. Reliability of this database is critical to the functioning of the system and these issue are discussed in [KAGM95]. To promote interoperability between workflow systems, our proposed architecture based on the WfMC reference model is shown in figure 5. Based on this architecture we discuss implementation support for all the features we discussed in the previous section. For workflows that will benefit from the use of advanced transaction models, we believe that ACTA/ASSET based primitives should be used in conjunction with concepts from advanced transaction models to provide flexibility in building complex workflows rather than providing a library of extended transaction models (less flexible) that can be used by the workflow designers. Since efforts are currently underway [CR95] to provide ACTA/ASSET based primitives in industrial strength systems, we assume that the local resource managers can provide such primitives.

6.2 Supporting Coordinated Tasks

- **data flow:** If *explicit* routing or *implicit-read* is specified, WNM retrieves the objects from the WDB and supplies it to the application agent which in turn hands it over to the invoked application. In the case of *implicit-write*, the WNM gives the application agent a write *token* for each object. Hence if there are concurrent activities which modify the same object (like documents), object accesses from them will be serialized to ensure data consistency. The checkin/checkout option is also handled similarly. Transactional activities (defined by transactions) typically have data dependencies between them and hence there is no need for specifying data flow.
- **control flow:** The WNM handles the value and outcome based dependencies. By referring to the appropriate results and the task definition in the WDB, the WNM schedules the qualifying activities and notes this fact in the WDB. The STM handles the state and temporal-state dependencies. Scheduling strategies developed in [Kle91, ASSR93, G9̈3, GHKM94, MLTA94] can be used for this purpose. Task control can be passed between WNM and STM if for example there are value and state based dependencies in the same task. Since all facts are noted in the WDB, even in case of system failures, activities will not be missed or scheduled twice.
- **nested tasks:** When the parent activity is scheduled, the WNM recognizes from the task definition that a subtask is to be invoked. All data available to the parent activity is made available when control is passed to the subtask. The subtask is now executed just like a regular task. In case of system failures, the subtask is restored to its latest consistent state (activity that was being executed at time of failure). When the terminating activity of the subtask has finished, control is passed along with the result to the parent activity. If there are logical failures within a subtask then compensating actions are to be applied for all activities (including upper levels) until an activity from which a different path can be taken is reached (achieved by using activity outcome dependency). If there are logical failures after a subtask has executed then depending on the specification in the task definition, the subtask is either compensated by a single action or by the invocation of a nested compensation task which compensates each of the individual activities in the reverse order. Nesting of transactional tasks is handled by the STM in the form of dependencies and hence there are no new issues to be discussed.

- **atomic-unit of activities:** In the case of non-transactional activities grouped into an atomic-unit, the implementation is as follows: When each activity completes, the outcome is noted in the WDB and when all the activities complete, a decision is made whether compensating actions are needed for some of the completed activities (since certain activities could not complete successfully and the atomic-unit has failed). If all of them are successful then the atomic-unit is successful. If several transactional-activities are grouped into a atomic-unit, to maintain data consistency, their changes can be made persistent only after determining the outcome of the activities. Hence locks are to be held for all the transactions till a decision can be made. If it is decided that all transactions are successful then a group-commit (as explained in ASSET) is used to make the changes persistent, else all the changes are aborted. To prevent loss of work performed by all activities due to system failure when the remaining activities are executed, the local resource managers should save the changes persistently in a shadow area on the disk and apply it to the database only at group-commit time. When the transactions are accessing resources from multiple resource managers, the semantic transaction manager should ensure that the primitives are handled properly. Below we give a simple algorithm to show how form_dependency (an ASSET primitive used in group-commit) can be implemented across multi-systems:

```
form_dependency(GC, $t_i$, $t_j$)
    $S_i$ = set of local sites of $t_i$;
    $S_j$ = set of local sites of $t_j$;
    At each site $s$ in $S_i \cap S_j$ do:
        form_dependency(GC, $t_i^s$, $t_j^s$)
```

- **inter-task dependencies:** Inter-task dependencies are easier to enforce. Since the set of activities which are dependent is specified, the WNM enforces dependencies by referring to the task histories in the WDB. Activities are scheduled for execution based on the order determined by the first set of dependent activities of tasks involved in the dependency. In the case of transactional activities, ticket-based strategies described in [GRS94] can be used.

- **correct execution of task:** Here we will mainly discuss *proper termination* of tasks. Other correctness issues (like system failures) are discussed wherever appropriate in the context of the rest of the primitives. As soon as the WNM recognizes that the required $final - set$ of activities (or final-set of transactions states in case of transactional activities) is reached, the task is considered complete and this fact is recorded in the WDB. If for any reason the task is to be aborted then activities which are to be compensated (as indicated in the task definition) are automatically compensated to restore consistency and the task is marked as aborted in the WDB. Execution of compensating activities is recorded by the workflow engine in the WDB the same way the forward execution of activities is recorded. Hence even if there are multiple system failures while compensating actions are being executed, after restart, the system will be aware that it is performing backward recovery from logical failures.

- **correct execution of activity:** Here we discuss recovery from application failures which are caused by a system failure at the local systems (failure of application agent is discussed later) or a logical failure within the application-program. Consider the failure of the local system while executing an activity (application-program). Recovering from such a failure without causing inconsistency is difficult because the *internal state* of the application-program at the time of system failure is not known. Numerous issues must to be considered to restore the consistency of the data accessed by the application. Such a discussion is beyond the scope of this paper and techniques described in [FaBK87] can be used. Logical failures in applications should be handled by applying the appropriate compensating actions. In the case of activities which are transactional, such failures (aborts) are usually handled by suitable commit/abort dependencies (handled by STM).

- **entity responsible for executing an activity:** This is specified at task definition time. A role that has been defined in the staff-database or an application-program with all its environmental settings is specified to perform work for an activity. If the specified role does not respond within a specified time, a superior role is to be notified to handle the activity.
- **task-definition modification:** When a task-definition modification is complete, and if it to be immediately applied to the executing instances of the task, then a integrity check is performed by the system to see if inconsistencies (*e.g.* incorrect data/control dependencies) will be formed by doing so. If there are violations, the role modifying the task-definition must be notified.
- **staff-definition** and **organizational hierarchy**. Here again when there is a modification, the system has to perform integrity checks to ensure consistency of data. For example, if activities (work) have been assigned to a role, then the system will not allow elimination of that role until the work is performed or assigned to some other role.
- **handling system failures:**. Here we discuss mechanisms to recover from system failures. Since the WDB contains state information of all tasks and activities, if there is a failure at the workflow engine forward recovery is possible. Hence the information recorded in the WDB must reflect the actual states. After a program terminates, the results are to be passed by the application agent to the workflow engine which records it in the WDB. Consider the failure of the workflow engine followed by the failure of the application agent. Unless proper care is taken, results of the programs that completed execution between the two failures will be lost. Although the program has terminated successfully, the state of the application as recorded in the WDB is still 'executing', which is an inconsistency. Hence significant events that happen at the local systems are logged at the application agent. Inconsistencies are still possible due to the *window* of time that exists between the actual commit of the transaction(s) by the local resource manager and the instant when control returns from the program to the application agent. If there is a failure during this window, the transactions executed on behalf of the program have committed while this fact is not recorded at the application agent which also fails. Hence a better alternative is to have the local database and the WDB perform a two-phase commit to ensure that the result and the status of the activity is properly recorded in the WDB. Though it is a good idea, a couple of problems have to be solved before it can be implemented. The first problem is that not all local resource managers provide the two-phase commit interface and even if they do, most do not yet conform with the XA interface standard proposed by X/Open [X/O92]. The second problem exists because of legacy applications. Since transactions are bundled somewhere in the legacy code, it is not clear how many transactions each of these applications contain and what is the status of each when a failure occurs. One solution to this is to provide suitable wrapper or library routines that can capture commits/aborts.
- **handling logical failures:**. A simple way of handling logical failures (resulting from unsuccessful execution of activities) is to define tasks with suitable provisions to handle such failures. This is

another instance where concepts from advanced transaction models (like flexible transactions) can be used to develop applications by defining contingent activities. Using the 'outcome' primitive for control flow, by compensating executed activities, alternative options can be explored. Another option which current workflow systems provide (also useful in case of system failures) is dynamic modification of task instances (exception handling) to manually resolve the failure.

6.3 Supporting Cooperative Tasks

- **hierarchy of activities:** This provides the task definition for cooperative tasks and is stored in the WDB. The STM ensures that all activities at a particular level are complete (they meet the test specifications) before its parent activity (at the next higher level) is completed. Concurrency control and recovery issues are discussed in the later primitives.
- **work performed by each activity:** The STM ensures that roles are properly notified about the actions (*e.g.* all objects to be designed) they need to perform. This primitive works in conjunction with the event based action, selective backtracking and hierarchy modification features. Hence even after an activity has been completed (*e.g.* object has been designed) if changes are needed due to rollbacks or changes to other objects or changes in test specification, the roles responsible for the activity are notified.
- **correct execution of activity:** An activity is considered complete when it meets the test specification[6]. The test specification check can be performed automatically or manually.
- **managing concurrent object accesses:** Compatibility matrices, checkin/checkout schemes and its extensions [KLMP84, RRD90], version based schemes and *notify* locks [GR91] are insufficient for cooperative tasks since large environments have flexible concurrency and ordering (controlled interleaving of object accesses) requirements to ensure global consistency of objects. We briefly discuss how two schemes, one based on protocols (similar to automata based control as described in [NZ90]) and the other based on ACTA/ASSET primitives work.
  - ▷ **protocols:** At task definition time, protocols specify rules that govern object accesses at run time. They allow task structures to be defined dynamically. Roles constantly read, modify and certify that the objects meet certain specifications. Other roles can access the objects for viewing, if the object meets certain specification (taken care of by the protocol), else there will be inconsistency. Implementing these access protocols is similar to handling temporal-state dependency. Since the STM can handle them, these protocols are enforced in coordination with the local resource managers by accessing rules and history information from the WDB. Since changes are made persistently by individual activities, selective rollback (discussed later) is needed if objects are to be redesigned.
  - ▷ **semantic primitives:** Another option to control concurrent object accesses in cooperative environments is through semantics-based ASSET primitives like *form_dependency* and *permit*. The difference is that the type of interactions is actually governed by the designers dynamically. While it allows more dynamic interactions than protocols, it is difficult to automatically enforce constraints on the interactions. At the activity level, if one activity (role responsible for the activity)

[6]This is slightly different from exit conditions for activities in coordinated tasks. In the case of coordinated tasks checks are used to ensure that the activity has been performed, whereas here the test specification is used to ensure the consistency of the objects designed.

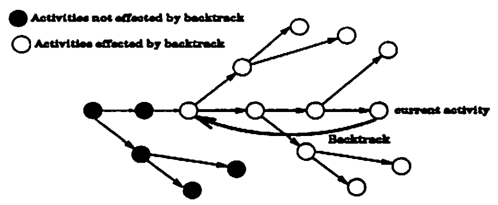
● Activities not effected by backtrack
○ Activities effected by backtrack
current activity
Backtrack

Figure 6: *Backtracking in Cooperative Applications*

permits another to perform certain operations on some of its objects, then the STM will in turn split this operation into permit() actions at the respective local resource managers. The actions are then made persistent using group-commit.

- **selective backtracking:** Support for back-tracking is essential in workflow systems to handle co-operative applications that use protocols for controlling object accesses. In cooperative applications, if a designer backtracks, all other designers who made their decisions based on the outcome of the activities that were backtracked have to be notified about those activities as shown in figure 6. The STM refers to the history in the WDB and selects the entire tree of activities that evolved with the backtracked activity as the root. The designers who were responsible for those activities are notified, who in turn should backtrack or take appropriate action if needed. Other activities that are not part of this tree are not affected. In all these primitives it is assumed that the local resource manager provides facilities to persistently store local changes made by the activity (like a shadow portion on the disk) so that changes are not lost due to local system failures.
- **event based action:** Both the STM and the WNM contain routines that can trap specific events (as defined in the event-action database in the WDB). Thus changes to an object can be trapped and the required designers notified or alternative actions can be performed automatically. Transitive closure of event actions allows a change to be propagated along the hierarchy.
- **hierarchy modification:** Unlike task modification in coordinated tasks, these modifications go into effect immediately. Depending upon the modification, appropriate roles are notified.

6.4 Other Features

- **interoperability between workflow systems:** The WFMC reference model allows transfer of task/activity control between different workflow systems. When tasks are crossing boundaries, system failures should not cause inconsistency. Hence our approach is to use a two-phase commit between the WDB's of the two system to ensure that both the systems have consistent information about the transfer. Since the reference model doesn't specify the need for a WDB in the workflow engine we feel it is essential in several ways to ensure data consistency. Data that is to be transferred between the systems can be bundled along with control. Also if there are task/activity definition modifications on one system, other systems which refer to those have to be notified to ensure the overall consistency of task/activity definitions. System and logical failures are handle by the individual systems as described earlier. If compensating actions are to be executed by different workflow systems on behalf of a task, control is transferred between systems in the same manner as that of regular task transfer.

- **heterogeneous application data formats:** The workflow engine stores data in its own internal format in the workflow database. To handle applications which accept data that conform to specific standards (like OLE), the application agent will be responsible for converting the data into the required standard before presenting it to the application. Hence the application agents should be embedded with information about such information exchange standards.
- **non-transactional resource managers:** To ensure consistency in the case of concurrent accesses to objects managed by non-transactional resource managers, we have explored the following options. From the architecture point of view, the only option possible is to force object (*program* data and not task data) accesses from the programs to pass through the application agent. The application agent will provide the object if it is not locked or will delay/notify the entity responsible for performing the activity. Two other options are possible which require additional information to be specified at task definition time. One is to explicitly specify at definition time, the objects needed by the activity which the application agent will track of at each site during execution. The second option is to disallow concurrent execution of activity instances with the same definition. If instances of different activity definitions try to access the same objects, the problem can be handled by providing a compatibility matrix at task definition time based on the tuple *(TaskID,ActivityID,ObjectID,AccessMode)*. At execution time, as and when activities start and terminate such tuples are added and deleted from the WDB. If a conflict is detected the activity is blocked and the person is notified. Note that tasks are not serialized, but only conflicting accesses are serialized. Executing compensating activities in the case of resource managers which are not transactional is difficult since the local system does not have its own log to restore before images if needed. As described earlier, a simple option is to have object requests from applications to be trapped by the application agent, which can store a before-image and restore it if compensation is required.

6.5 Discussion

A high level discussion on how advanced (customized) transaction models can be used to improve the capabilities of current workflow systems is provided in [Geo95]. Our work complements this by providing a fairly detailed description of an implementation architecture and mechanisms to support the essential modeling primitives we have identified. Specifically we focused on correctness and consistency problems in a variety of situations and provided schemes to handle them. Wherever appropriate, we also discussed problems that arise in implementing these schemes. Other areas which current workflow systems fail to address like performance and availability have not been discussed here. These issues are discussed in [MAGK95].

7 Conclusions

Workflow management is emerging as a powerful tool to improve productivity of organizations. However in its current state, it still lacks certain features that are essential for their functionality; especially support for cooperative activities, ensuring consistency of data and correctness of execution. Since these issues have been extensively investigated in area of advanced transaction management, in this work we have cross fertilized the two areas to develop a model and an architecture that provides flexibility in defining tasks and specifying correctness and consistency requirements of advanced database applications.

Our primary contributions are as follows. We have identified three important defining properties for modeling — supporting coordinated tasks, supporting cooperative tasks and features to handle heterogeneity and interoperability. For each of them we discussed the required modeling primitives and correctness options. We then described an implementation architecture on the lines of the reference model provided by the workflow coalition. Our workflow engine contains a semantic transaction manager along with a workflow navigation manager to handle all the different types of modeling primitives we have identified. Based on this architecture we explained implementation mechanisms for each of the modeling primitives. We discussed concurrency control, recovery and commit processing issues wherever appropriate. We also discussed problems that can arise in implementing our schemes. A secondary contribution of our work is the classification of advanced transaction models based on the functionality they provide. Our future plans include building a prototype system based on this architecture to study modeling, correctness and implementation issues when mobile users are involved, and when there are network/site failures.

Workflow management is still in its infancy and we feel our work is among the first few steps in an attempt to define concrete guidelines for improving the features offered by workflow management systems. By no means is our work exhaustive and lot more efforts are needed both from research and industry before workflow management systems become common place as database systems are today in enterprise management.

References

[ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using flexible transactions to support multi-system telecommunication applications. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 65, Vancouver, BC, Canada, August 1992.

[ASSR93] P. C. Attie, M. P. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proc. Intl' Conf. on Very Large Data Bases*, page 134, Dublin, Ireland, August 1993.

[BDG+94] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, and K. Ramamritham. ASSET: A System for Supporting Extended Transactions. In *Proc. 1994 SIGMOD International Conference on Management of Data*, pages 44–54, May 1994.

[BDS+93] Y. Breitbart, A. Deacon, H.J. Schek, A. Sheth, and Weikum. G. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *ACM SIGMOD Record*, 22(3), 1993.

[BK91] N. S. Barghouti and G. E. Kaiser. Concurrency control in advanced database applications. *ACM Computing Surveys*, 23(3):269, September 1991.

[BKK85] F. Bancilhon, W. Kim, and H. Korth. A model of CAD transactions. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 25, Stockholm, Sweden, August 1985.

[BST92] Y. Breitbart, A. Silberschatz, and G. R. Thompson. An approach to recovery management in a multidatabase system. *The VLDB Journal*, 1(1):1, July 1992.

[BSW88] C Beeri, H Schek, and G. Weikum. Multi-level transaction management - theoretical art or practical need? volume 303 of *Lecture Notes in CS*, pages 135–154. Springer, New York, 1988.

[CR90] P. K. Chrysanthis and K. Ramamritham. ACTA: A framework for specifying and reasoning about transaction structure and behavior. In *Proc. ACM SIGMOD Conf.*, page 194, Atlantic City, NJ, May 1990.

[CR94] Panos Chrysanthis and Krithi Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Trans. on Database Sys.*, 19(3):450–491, 1994.

[CR95] Pedregal-Martin C. and K. Ramamritham. ARIES/RH: Robust Support for Delegation by Rewriting History. Technical Report TR 95-51, University of Massachusetts, Computer Science Dept., 1995.

[DE89] W. Du and A. Elmagarmid. Quasi serializability; a correctness criterion for global concurrency control in interbase. In *Proc. Intl' Conf. on Very Large Data Bases*, August 1989.

[DGMH+93] U. Dayal, H. Garcia-Molina, M. Hsu, B. Kao, and M.C. Shan. Third Generation TP Monitors: A Database Challenge. In *Proc. of 1993 SIGMOD International Conference on Management of Data*, pages 393–398, May 1993.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing Long-running Activities with Triggers and Transactions. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 204–214, June 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-running Activities. In *Proceedings of the Sixteenth International Conference on Very Large Databases*, pages 113–122, August 1991.

[DHM+95] U. Dayal, M. Hsu, C. Mohan, M Rusinkiewicz, and F Schwenkreis. Workflow Automation, 1995. Panel Session at 11th International Conference on Data Engineering, Taiwan.

[ELLR90] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for interbase. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 507, Brisbane, Australia, August 1990.

[Elm92] A. Elmagarmid, editor. *Transaction Models for Advanced Database Applications*. Morgan-Kaufmann, 1992.

[FaBK87] J.C. Freytag and F. Cristian abd B. Kaehler. Masking system crashes in database application programs. In *Proc. of 13th Int'l. Conf. on Very Large Data Bases*, Brighton, England, 1987.

[FD94] Leymann F. and Roller D. Business Process Management With FlowMark. In *Proceedings of IEEE CompCon*, pages 230–234, 1994.

[FZ88] M. F. Fernandez and S. B. Zdonik. Transaction groups: A model for controlling co-operative transactions. In *Proc. Workshop on Persistent Object Sys.: Their Design, Implementation, and Use*, page 128, Newcastle, Australia, January 1988.

[G$\ddot{9}$3] R. Günthör. Extended transaction processing based on dependency rules. In *Proc. of the RIDE-IMS Workshop*, Vienna, Austria, 1993.

[Geo95] Georgakopolous D. and Hornick M. and Sheth A. An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–152, 1995.

[GHKM94] D. Georgakopoulos, M. Hornick, P. Krychniak, and F. Manola. Specification and management of extended transactions in a programmable transaction environment. In *Proc. IEEE Int'l. Conf. on Data Eng.*, page 462, Houston, TX, February 1994.

[GMGK+90] H. Garcia-Molina, G. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating Multi-Transaction Activities. Technical Report CS-TR-247-90, Princeton University, 1990.

[GMGK+91] H. Garcia-Molina, G. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling Long-Running Activities as Nested Sagas. *Bulletin of the Technical Committee on Data Engineering*, 14(1), 1991. IEEE Computer Society.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proc. 1987 SIGMOD International Conference on Management of Data*, pages 249–259, May 1987.

[GR91] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, CA, 1991.

[GRS94] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using tickets to enforce the serializability of multidatabase transactions. *IEEE Trans. on Knowledge and Data Eng.*, 6(1):166, February 1994.

[Hsu93] M. Hsu. Special Issue on Workflow and Extended Transaction Systems. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 16(2), 1993.

[Hsu95] M. Hsu. Special Issue on Workflow Systems. *Bulletin of the Technical Committee on Data Engineering, IEEE*, 18(1), 1995.

[KAGM95] M Kamath, G. Alonso, R. Günthör, and C. Mohan. Providing High Availability in Workflow Management Systems. Technical report, IBM Almaden Research Center, 1995.

[Kai90] G. E. Kaiser. A flexible transaction model for software engineering. In *Proc. IEEE Int'l. Conf. on Data Eng.*, page 560, Los Angeles, CA, February 1990.

[Kle91] J. Klein. Advanced Rule Driven Transaction Management. In *Proceedings IEEE Spring Compcon*, 1991.

[KLMP84] W. Kim, R. Lorie, D. McNabb, and W. Plouffe. A transaction mechanism for engineering design databases. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 355, Singapore, August 1984.

[KP92] Ramamritham K. and Chrysanthis P.K. In Search of Acceptability Criteria: Database Consistency Requirements and Transaction Correctness Properties. In *Distributed Object Management, Ozsu, Dayal, Valduriez, Ed.*, Morgan Kaufmann Publishers, 1992.

[KR95] M. Kamath and K. Ramamritham. Modeling, Correctness & Systems Issues in Supporting Advanced Database Applications using Workflow Management Systems. Technical Report TR 95-50, University of Massachusetts, Computer Science Dept., 1995.

[Kri95] Krishnakumar N. and Sheth A. Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. *Distributed and Parallel Databases*, 3(2):119–152, 1995.

[KS90] H. F. Korth and G. D. Speegle. Long duration transactions in software design projects. In *Proc. IEEE Int'l. Conf. on Data Eng.*, page 568, Los Angeles, CA, February 1990.

[KSUW85] P. Klahold, G. Schlageter, R. Unland, and W. Wilkes. A transaction model supporting complex applications in integrated information systems. In *Proc. ACM SIGMOD Conf.*, page 388, Austin, TX, May 1985.

[MAGK95] C. Mohan, G. Alonso, R. Gunther, and M. Kamath. Exotica: A research perspective on workflow management systems. *Bulletin of the Technical Committee on Data Engineering*, 18(1), March 1995. IEEE Computer Society.

[McC92] S. McCready. There is more than one kind of Work-flow Software. *ComputerWorld*, November 1992.

[Mem94] WfMC Members. Workflow Management Reference Model, 1994. The Workflow Management Coalition, Accessible via: http://www.aiai.ed.ac.uk/WfMC/.

[MLTA94] M.P.Singh, L.G.Meredith, C. Tomlison, and P.C. Attie. An Algebraic Approach for Workflow Scheduling. Technical Report Carnot-049-94, MCC, 1994.

[Moh94] C. Mohan. Advanced Transaction Models - Survey and Critique, 1994. Tutorial presented at ACM SIGMOD International Conference on Management of Data.

[Mos81] J. E. B. Moss. Nested transactions: An approach to reliable distributed computing. Technical report, PhD Thesis, MIT, Cambridge, MA, April 1981.

[MP90] B. Martin and C. Pedersen. Long-Lived Concurrent Activities. Technical Report HPL-90-178, Hewlett-Packard Labs, 1990.

[MRB$^+$92] S. Mehrotra, R. Rastogi, Y. Breitbart, H. Korth, and A. Silberschatz. Ensuring transaction atomicity in multidatabase systems. In *Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, page 164, San Diego, CA, June 1992.

[MS93] D.R. McCarthy and S.K. Sarin. Workflow and Transactions in InConcert. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993. IEEE Computer Society.

[NZ90] M. H. Nodine and S. B. Zdonik. Cooperative transaction hierarchies: A transaction model to support design applications. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 83, Brisbane, Australia, August 1990.

[PKH88] C. Pu, G. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 26, Los Angeles, CA, August 1988.

[RRD90] M.A. Ranft, S. Rehm, and K.R. Dittrich. How to Share Work on Shared Objects in Design Databases. In *Proc. IEEE Int'l. Conf. on Data Eng.*, 1990.

[Sch93] F. Schwenjreis. APRICOTS - A Prototype Implementation of a ConTract System - Management of Control Flow and the Communication System. In *Proc. of the 12th Symposium on Reliable Distributed Systems, IEEE Computer Society Press*, Princeton(NJ), 1993.

[Ska89] A. H. Skarra. Concurrency control for cooperating transactions in an object-oriented database. In *Proc. ACM SIGPLAN Workshop on Object-Based Concurrent Programming, ACM SIGPLAN Notices*, page 145, April 1989. Published as Proc. ACM SIGPLAN Workshop on Object-Based Concurrent Programming, ACM SIGPLAN Notices, volume 24, number 4.

[SR93] A. Sheth and M. Rusinkiewicz. On transactional workflows. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993. IEEE Computer Society.

[SRK92] A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 14, pages 555–582. Morgan Kaufmann Publishers, San Mateo, 1992.

[ST94] B. Salzberg and D. Tombroff. A Programming Tool to Support Long-Running Activities. Technical Report NU-CCS-94-10, College of Computer Science, Northeastern University, Boston, 1994.

[US92] R. Unland and G. Schlageter. A Transaction Manager Development Facility for Non Standard Database Systems. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 11, pages 399–466. Morgan Kaufmann Publishers, San Mateo, 1992.

[WR92] H. Waechter and A. Reuter. The ConTract Model. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7, pages 219–263. Morgan Kaufmann Publishers, San Mateo, 1992.

[WS92] G. Weikum and Hans-J. Schek. Concepts and Applications of Multilevel and Open Nested Transactions. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 13, pages 515–554. Morgan Kaufmann Publishers, San Mateo, 1992.

[X/O92] X/Open, editor. *Distributed Transaction Processing: The XA Specification.* X/Open Company Limited, UK, 1992.

[ZNBB94] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proc. 1994 SIGMOD International Conference on Management of Data*, pages 67–78, 1994.