1 TRANSACTIONS IN TRANSACTIONAL WORKFLOWS

Devashish Worah
and Amit Sheth

Abstract: Workflow management systems (WFMSs) are finding wide applicability in small and large organizational settings. Advanced transaction models (ATMs) focus on maintaining data consistency and have provided solutions to many problems such as correctness, consistency, and reliability in transaction processing and database management environments. While such concepts have yet to be solved in the domain of workflow systems, database researchers have proposed to use, or attempted to use ATMs to model workflows. In this paper we survey the work done in the area of transactional workflow systems. We then argue that *workflow requirements in large-scale enterprise-wide applications involving heterogeneous and distributed environments either differ or exceed the modeling and functionality support provided by ATMs. We propose that an ATM is unlikely to provide the primary basis for modeling of workflow applications, and subsequently workflow management.* We discuss a framework for error handling and recovery in the METEOR$_2$ WFMS that borrows from relevant work in ATMs, distributed systems, software engineering, and organizational sciences. We have also presented various connotations of *transactions* in real-world organizational processes today. Finally, we point out the need for looking beyond ATMs and using a multi-disciplinary approach for modeling large-scale workflow applications of the future.

1.1 INTRODUCTION

A *workflow* is an activity involving the coordinated execution of multiple *tasks* performed by different processing entities [Krishnakumar and Sheth, 1995]. A *workflow process* is an automated organizational process involving both human and automated tasks. *Workflow management* is the automated coordination, control and communication of work as is required to satisfy workflow processes [Sheth et al., 1996a]. There has been a growing acceptance of workflow technology in numerous application domains such as telecommunications, software engineering, manufacturing, production, finance and banking, health care, shipping and office automation [Smith, 1993, Joosten et al., 1994, Georgakopoulos et al., 1995, Fischer, 1995, Tang and Veijalainen, 1995, Sheth et al., 1996b, Palaniswami et al., 1996, Bonner et al., 1996, Perry et al., 1996]. Workflow Management Systems (WFMSs) are being used in inter- and intra-enterprise environments to re-engineer, streamline, automate, and track organizational processes involving humans and automated information systems.

In spite of the proliferation of commercial products for workflow management (including modeling and system supported enactment), workflow technology is relatively immature to be able to address the myriad complexities associated with real-world applications. The current state-of-the-art is dictated by the commercial market which is focused toward providing automation within the office environment with emphasis on coordinating human activities, and facilitating document routing, imaging, and reporting. However, the requirements for workflows in large-scale multi-system applications executing in heterogeneous, autonomous, distributed (HAD) environments involving multiple communication paradigms, humans and legacy application systems far exceeds the capabilities provided by products today [Sheth, 1995].

Some of the apparent weaknesses of workflow models that need to be addressed by the workflow community include the lack of a clear theoretical basis, undefined correctness criteria, limited support for synchronization of concurrent workflows, lack of interoperability, scalability and availability, and lack of support for reliability in the presence of failures and exceptions [Breitbart et al., 1993, Jin et al., 1993, Georgakopoulos et al., 1995, Mohan et al., 1995, Alonso and Schek, 1996, Kamath and Ramamritham, 1996, Leymann et al., 1996, Alonso et al., 1996a]. In addition, a successful workflow-enabled solution should address many of the growing user needs that have resulted from:

- emerging and maturing infrastructure technologies and standards for distributed computing such as the World Wide Web, Common Object Request Broker Architecture [OMG, 1995b], Distributed Common Object Model (DCOM), ActiveX, Lotus Notes, and Java.

- increasing need for electronic commerce using standard protocols such as Electronic Data Interchange (EDI) (e.g., ANSI X.12 and HL7),
- additional organizational requirements in terms of security and authentication,
- demands for integrated collaboration (not just coordination) support,
- increasing use of heterogeneous multimedia data, and
- requirements to support dynamic workflows to respond to the fast changing environment (e.g., defense planning), or for supporting today's dynamic and virtual enterprises.

Workflow technology has emerged as a multi-disciplinary field with significant contributions from the areas of software engineering, software process management, database management, and distributed systems [Sheth et al., 1996a]. In spite of the standardization efforts of the Workflow Management Coalition [Coalition, 1994], a consensus on many broader aspects have not yet been achieved.

Work in the areas of transaction processing [Gray and Reuter, 1993] and database systems, and many (but not all) efforts related to ATMs [Elmagarmid, 1992, Chrysanthis and Ramamritham, 1991, Georgakopoulos et al., 1994], are based on a strong theoretical basis. They have proposed or documented solutions (although many of which have yet to be implemented) to problems such as correctness, consistency, and recovery when the constituent tasks are transactional, or the processing entities provide a transactional interface. There exists a strong school of thought, primarily comprised of researchers from the database community, which views a workflow model as an extension of ATMs [Georgakopoulos and Hornick, 1994, Georgakopoulos et al., 1994, Chen and Dayal, 1996, Biliris et al., 1994, Weikum, 1993, Wachter and Reuter, 1992]. However, it has also been observed [Breitbart et al., 1993, Alonso et al., 1996b, Worah and Sheth, 1996] that ATMs have limited applicability in the context of workflows due to their inability to model the rich requirements of today's organizational processes adequately.

Traditional database transactions provide properties such as failure atomicity and concurrency control. These are very useful concepts that could be applicable in workflows. For example, failure atomicity can be supported for a task that interacts with a DBMS, or a group of tasks using the two-phase commit protocol. There is a potential need for concurrency control and synchronization of workflow processes for addressing correctness concerns during workflow execution [Jin et al., 1993, Alonso et al., 1996a]. Based on our review of requirements of existing applications using workflows [Worah and Sheth,

1996], we feel that transactional features form only a small part of a large-scale workflow application. Workflow requirements either exceed, or significantly differ from those of ATMs in terms of modeling, coordination and run-time requirements. It would definitely be useful to incorporate transactional semantics such as recovery, relaxed atomicity and isolation to ensure reliable workflow executions. Nevertheless, to view a workflow as an ATM, or to use existing ATMs to completely model workflows would be inappropriate. We do not think that existing ATMs provide a comprehensive or sufficient framework for modeling large-scale enterprise-wide workflows.

Our observations in this chapter reflect our experience in modeling and development efforts for a real-world workflow application for immunization tracking [Sheth et al., 1996b, Palaniswami et al., 1996], experience in trying to use flexible transactions in multi-system telecommunication applications [Ansari et al., 1992], and our understanding of the current state of the workflow technology and its real-world or realistic applications [Sheth et al., 1996b, Medina-Mora and Cartron, 1996, Bonner et al., 1996, Ansari et al., 1992, Vivier et al., 1996, Sheth and Joosten, 1996].

We emphasize the need for looking beyond the framework of ATMs for modeling and executing workflow applications. The term *transaction* as it is used in business processes today has multiple connotations, database transactions being only one of them. For example, EDI transactions are used for defining interfaces and data formats for exchange of data between organizations and Health Level 7 (HL7) transactions are used to transfer patient data between health care organizations. We discuss other uses of this term in section 1.7. Workflow systems should evolve with the needs of the business and scientific user communities, both in terms of modeling and run-time support. Of course, it is possible that in some specific domains, ATM based workflow models may be sufficient, however, we believe, such cases would be very few.

The organization of this chapter is as follows. Sections 2 through 5 are tutorial in nature. In section 2 we review the research in the domain of ATMs. The next section discusses the characteristics of transactional workflows and significant research in this area. One of the primary focus of transactional workflows is recovery. In section 4 we highlight the issues involved in recovery for workflow systems. Section 5 discusses the types of errors that could occur during workflow execution. In section 6 we discuss a practical implementation of error handling and recovery in a large-scale WFMS. Section 6 provides a perspective into the characteristics and interpretation of *transactions* as they exist in workflow applications today. Finally, we conclude the paper with our observations regarding the role of transactions in transactional workflows.

1.2 ADVANCED TRANSACTION MODELS

In this section we will briefly describe some of the ATMs discussed in the literature [Gray and Reuter, 1993, Elmagarmid, 1992]. These models can be classified according to various characteristics that include transaction structure, intra-transaction concurrency, execution dependencies, visibility, durability, isolation requirements, and failure atomicity. ATMs permit grouping of their operations into hierarchical structures, and in most cases relax (some of) the ACID requirements of classical transactions. In this section, we discuss some of the ATMs that we feel are relevant in the context of workflow systems.

1.2.1 Nested Transactions

An important step in the evolution of a basic transaction model was the extension of the flat (single level) transaction structure to multi-level structures. A *Nested Transaction* [Moss, 1982] is a set of subtransactions that may recursively contain other subtransactions, thus forming a *transaction tree*. A child transaction may start after its parent has started and a parent transaction may terminate only after all its children terminate. If a parent transaction is aborted, all its children are aborted. However, when a child fails, the parent may choose its own way of recovery, for example the parent may execute another subtransaction that performs an alternative action (a *contingency subtransaction*). Nested transactions provide full isolation at the global level, but they permit increased modularity, finer granularity of failure handling, and a higher degree of intra-transaction concurrency than the traditional transactions.

1.2.2 Open Nested Transactions

Open Nested Transactions [Weikum and Schek, 1992] relax the isolation requirements by making the results of committed subtransactions visible to other concurrently executing nested transactions. This way, a higher degree of concurrency is achieved. To avoid inconsistent use of the results of committed subtransactions, only those subtransactions that *commute* with the committed ones are allowed to use their results. Two transactions (or, in general, two operations) are said to commute if their effects, i.e., their output and the final state of the database, are the same regardless of the order in which they were executed. In conventional systems, only *read* operations commute. Based on their semantics, however, one can also define update operations as commutative (for example increment operations of a counter).

1.2.3 Sagas

A *Saga* [Garcia-Molina and Salem, 1987] can deal with long-lived transactions. A Saga consists of a set of ACID subtransactions $T_1, \ldots, T_n$ with a predefined order of execution, and a set of *compensating subtransactions* $CT_1, \ldots, CT_{n-1}$, corresponding to $T_1, \ldots, T_{n-1}$. A saga completes successfully, if the subtransactions $T_1, \ldots, T_n$ have committed. If one of the subtransactions, say T_k, fails, then committed subtransactions $T_1, \ldots, T_{k-1}$ are undone by executing compensating subtransactions $CT_{k-1}, \ldots, CT_1$. Sagas relax the full isolation requirements and increase inter-transaction concurrency. An extension allows the nesting of Sagas [Garcia-Molina et al., 1991]. *Nested Sagas* provide useful mechanisms to structure steps involved within a long running transaction into hierarchical transaction structures. This model promotes a relaxed notion of atomicity whereby forward recovery is used in the form of *compensating transactions* to undo the effects of a failed transaction.

1.2.4 Multi-Level Transactions

Multi-Level Transactions are more generalized versions of nested transactions [Weikum and Schek, 1992, Gray and Reuter, 1993]. Subtransactions of a multi-level transactions can commit and release their resources before the (global) transaction successfully completes and commits. If a global transaction aborts, its failure atomicity may require that the effects of already committed subtransactions be undone by executing *compensating subtransactions*. A compensating subtransaction t^- semantically *undoes* effects of a committed subtransaction t, so that the state of the database before and after executing a sequence $t\ t^-$ is the same. However, an inconsistency may occur if other transaction s observe the effects of subtransactions that will be compensated later [Gray and Reuter, 1993, Garcia-Molina and Salem, 1987, Korth et al., 1990]. Open nested transactions use the commutativity to solve this problem. Since only subtransactions that commute with the committed ones are allowed to access the results, the execution sequence $t\ s\ t^-$ is equivalent to $s\ t\ t^-$ and, according to definition of compensation, to s, and therefore is consistent. A somewhat more general solution in the form of a *horizon of compensation*, has been proposed in [Krychniak et al., 1996] in the context of multi-level activities.

1.2.5 Flexible Transactions

Flexible Transactions [Elmagarmid et al., 1990, Zhang et al., 1994] have been proposed as a transaction model suitable for a multidatabase environment. A flexible transaction is a set of tasks, with a set of functionally equivalent subtransactions for each and a set of execution dependencies on the subtransactions, including failure dependencies, success dependencies, or external dependencies. To relax the isolation requirements, flexible transactions use com-

pensation and relax global atomicity requirements by allowing the transaction designer to specify acceptable states for termination of the flexible transaction, in which some subtransactions may be aborted. IPL [Chen et al., 1993] is a language proposed for the specification of flexible transactions with user-defined atomicity and isolation. It includes features of traditional programming languages, such as type specification to support specific data formats that are accepted or produced by subtransactions executing on different software systems, and preference descriptors with logical and algebraic formulae used for controlling commitments of transactions. Because flexible transactions share some more of the features of a workflow model, it was perhaps the first ATM to have been tried to prototype workflow applications [Ansari et al., 1992].

1.2.6 ACTA and its derivatives

Reasoning about various transaction models can be simplified using the *ACTA metamodel* [Chrysanthis and Ramamritham, 1992]. ACTA captures the important characteristics of transaction models and can be used to decide whether a particular transaction execution history obeys a given set of dependencies. However, defining a transaction with a particular set of properties and assuring that an execution history will preserve these properties remains a difficult problem.

In [Biliris et al., 1994], the authors propose a relaxed transaction facility called *ASSET*. It is based on transaction primitives derived from the ACTA framework that can be used at a programming level to specify customized, application specific transaction models that allow cooperation and interaction. The transaction primitives include a basic and an extended set of constructs that can be used in an application that needs to support custom transactional semantics at the application level. These can be used to support very limited forms of workflows that involve transaction-like components. In some sense, this demonstrates the limitations one may face when trying to use an ATM as a primary basis for workflow modeling.

1.3 TRANSACTIONAL WORKFLOWS

The term *transactional workflows* [Sheth and Rusinkiewicz, 1993] was introduced to clearly recognize the relevance of transactions to workflows. It has been subsequently used by a number of researchers [Breitbart et al., 1993, Rusinkiewicz and Sheth, 1995, Krishnakumar and Sheth, 1995, Georgakopoulos et al., 1995, Tang and Veijalainen, 1995, Leymann et al., 1996]. Transactional workflows involve the coordinated execution of multiple related tasks that require access to HAD systems and support selective use of transactional properties for individual tasks or entire workflows. They use ATMs to specify

workflow correctness, data-consistency and reliability. Transactional workflows provide functionality required by each workflow process (e.g., allow task collaboration and support the workflow structure) which is usually not available in typical DBMS and TP-monitor transactions. Furthermore, they address issues related to reliable execution of workflows (both single and multiple) in the presence of concurrency and failures.

Transactional workflows do not imply that workflows are similar or equivalent to database transactions, or support all the ACID transaction properties. They might not strictly support some of the important transaction features supported by TP monitors (e.g., concurrency control, backward recovery, and consistency of data). Nevertheless, such workflows share the objectives of some of the ATMs in terms of being able to enforce relaxed transaction semantics to a set of activities.

In a somewhat conservative view, transactional workflows are workflows supported by an ATM that defines workflow correctness and reliability criteria [Georgakopoulos et al., 1995]. In such a workflow, the tasks are mapped to constituent transactions of an advanced transaction supported by an ATM [Georgakopoulos et al., 1994], and control flow is defined as dependencies between transactional steps. Similarly, in [Weikum, 1993] an extra *control* layer in terms of dependencies is added to ATM to provide functionality to the transactions running in a large-scale distributed information systems environment.

A WFMS may provide transactional properties to support forward recovery, and use system and application semantics to support semantic based *correct* multi-system application execution [Sheth, 1995, Krishnakumar and Sheth, 1995]. These could include transaction management techniques such as logging, compensation, etc. to enable forward recovery and failure atomicity. In addition, the workflow could exhibit transactional properties for parts of its execution. It might use transaction management technology such as transactional-RPC between two components of a WFMS (e.g, scheduler and task manager), an extended commit coordinator [Miller et al., 1996], or a transactional protocol (XA) between a task manager and a processing entity.

In our view, the scope of transactional workflows extends beyond the purview of database transactions and ATMs. Workflow executions include tasks that might involve database transactions; however, large-scale workflow applications typically extend beyond the data-centric domains of databases and infrastructures that inherently support transaction semantics (e.g., TP-monitors), to more heterogeneous, distributed and non-transactional execution environments.

1.3.1 Previous Research on using Transactions for Workflows

Two major approaches have been used to study and define transactional workflows. The first one utilize a workflow model that is based on supporting organizational processes (also called business process modeling) as its basis, and complements it with transactional features to add reliability, consistency, and other transaction semantics. In the second approach, ATMs are enhanced to incorporate workflow related concepts to increase functionality and applicability in real-world settings. The degree to which each of the models incorporates transactional features varies, and depends largely on the requirements (such as flexibility, atomicity and isolation of individual task executions and multiple workflow instances, etc.) of the organizational processes it tries to model. In the remainder of this section, we discuss some of the research that has been done using ATMs and workflows.

ConTracts [Wachter and Reuter, 1992] were proposed as a mechanism for grouping transactions into a multitransaction activity. A ConTract consists of a set of predefined actions (with ACID properties) called *steps*, and an explicitly specified execution plan called a *script*. An execution of a ConTract must be *forward-recoverable*, that is, in the case of a failure the state of the ConTract must be restored and its execution may continue. In addition to the relaxed isolation, ConTracts provide relaxed atomicity so that a ConTract may be interrupted and re-instantiated.

Workflow applications are typically long-lived compared to database transactions. A workflow is seen as a *Long-Running Activity* in [Dayal et al., 1990, Dayal et al., 1991]. A Long-Running Activity is modeled as a set of execution units that may consist recursively of other activities, or top-level transactions (i.e., transactions that may spawn nested transactions). Control flow and data flow of an activity may be specified statically in the activity's *script*, or dynamically by Event-Condition-Action (ECA) rules. This model includes compensation, communication between execution units, querying the status of an activity, and exception handling.

Motivated by advanced application requirements, several ATMs have been proposed (refer to [Chrysanthis and Ramamritham, 1991, Georgakopoulos and Hornick, 1994] for frameworks for defining and comparing ATMs, [Elmagarmid, 1992] for several representative ATMs, for a representative model and specification that support application specific transaction properties, and [Breitbart et al., 1993, Hsu, 1993, Rusinkiewicz and Sheth, 1995] for earlier views on relationships between workflows and ATMs). ATMs extend the traditional (ACID) transaction model typically supported by DBMSs to allow advanced application functionality (e.g., permit task collaboration and coordination as it is required by ad hoc workflows) and improve throughput (e.g., reduce transac-

tion blocking and abortion caused by transaction synchronization). However, many of these extensions have resulted in application-specific ATMs that offer adequate correctness guarantees in a particular application, but not in others. Furthermore, an ATM may impose restrictions that are unacceptable in one application, yet required by another. If no existing ATM satisfies the requirements of an application, a new ATM is defined to do so.

In [Georgakopoulos et al., 1994], the authors define an extended (advanced) transaction framework for execution of workflows called the *Transaction Specification and Management Environment* (TSME). A workflow in this framework consists of constituent transactions corresponding to workflow tasks. In addition, workflows have an execution structure that is defined by an ATM; the ATM defines the correctness criteria for the workflow. The TSME claims to support various ATMs (extended transaction models) to ensure correctness and reliability of various types of workflow processes. Extended transactions consist of a set of constituent transactions and a set of dependencies between them. These transaction dependencies specify the transaction execution structures or correctness criteria. A programmable transaction management mechanism based on the ECA rules [Dayal et al., 1990] is used to enforce transaction state dependencies.

Semantic transaction models aim to improve performance and data consistency by executing a group of interacting steps within a single transaction and relaxing the ACID properties of this transaction in a controlled manner. In [Weikum, 1993], the author suggests that semantic transaction concepts be merged with workflow concepts to promote workflow systems that are consistent and reliable. The author defines a transactional workflow to be a control sphere that binds these transactions by using dependencies to enforce as much behavioral consistency as possible thereby enforcing reasonable amount of data consistency.

The *METEOR*† [Krishnakumar and Sheth, 1995] workflow model is an integration of many of the approaches described above. A workflow in METEOR is a collection of multiple tasks. Each of the tasks could be heterogeneous in nature. The execution behavior of the tasks are captured using well-defined task structures. This model supports tasks that have both transactional and non-transactional semantics. Groups of tasks along with their inter-task dependencies can be modeled as compound tasks. The compound tasks have their task structures too. Transactional workflows can be modeled using transactional tasks and transactional compound tasks as the basis of the workflow model. The METEOR$_2$ WFMS [Miller et al., 1996, Sheth et al., 1996b] is based on the METEOR model. It extends the model in terms of providing better sup-

†METEOR refers to the project carried out at Bellcore. METEOR$_2$ is its follow on at the LSDIS Lab of the University of Georgia.

port for failure recovery and error handling in heterogeneous and distributed workflow environments (see section 1.6.1 for additional details).

The *Exotica* project [Alonso et al., 1995, Alonso et al., 1996b] explores the role of advanced transaction management concepts in the context of workflows. A stated objective of this research is to develop workflow systems that are capable enough (in terms of reliability, scalability, and availability) to deal with very large, heterogeneous, distributed and legacy applications. One of the directions of this project is to research the synergy between workflow systems and advanced transaction models; the results that follow point in the direction that workflow systems are a superset of advanced transaction models [Alonso et al., 1996b] since workflow systems incorporate process and user oriented concepts that are beyond the purview of most ATMs. Partial backward recovery has been addressed in the context of the FlowMark WFMS [Leymann, 1995] by generalizing the transactional notions of compensation.

One of the projects in which transactional semantics have been applied to a group of steps define a logical construct called a *Consistency unit* (C-unit) [Tang and Veijalainen, 1995]. A C-unit is a collection of workflow steps and enforced dependencies between them. C-units relax the isolation and atomicity properties of transactional models. The authors also discuss how C-units can be used to develop transactional workflows that guarantee correctness of data in the view of integrity constraints that might exist across workflow processing entities.

The *INformation CArrier* (INCA) [Barbara et al., 1996] workflow model was proposed as a basis for developing dynamic workflows in distributed environments where the processing entities are relatively autonomous in nature. In this model, the INCA is an object that is associated with each workflow and encapsulates workflow data, history and processing rules. The transactional semantics of INCA procedures (or steps) are limited by the transaction support guaranteed by the underlying processing entity. The INCA itself is neither atomic nor isolated in the traditional sense of the terms. However, transactional and extended transactional concepts such as *redo of steps*, *compensating steps*and *contingency steps* have been included in the INCA rules to account for failures and forward recovery.

In the *Nested Process Management* environment [Chen and Dayal, 1996] a workflow process is defined using a hierarchical collection of transactions. Failure handling is supported using a two-phase approach. During the first phase of recovery, a bottom-up lookup along the task tree is performed to determine the oldest parent transaction that does not need to be compensated. The next phase involves compensation of all the children of this parent. In this model, failure atomicity of the workflow is relaxed in terms of compensating only parts of the workflow hierarchy.

The *Workflow Activity Model*(WAMO) [Eder and Liebhart, 1995] defines a workflow model that enables the workflow designer in modeling reliable workflows [Eder and Liebhart, 1996]. It uses an underlying relaxed transaction model that is characterized by relaxing i) failure atomicity of tasks, ii) serializability of concurrent and interleaved workflow instance executions, and iii) relaxing isolation in terms of externalization of task results.

Thus we see that transaction concepts have been applied to various degrees in the context of workflows. They have been used to define application specific and user-defined correctness, reliability and functional requirements within workflow executions. In the next section, we discuss features specific to transactions and ATMs that would be useful for implementing recovery in a WFMS.

1.4 WORKFLOW RECOVERY

Reliability is of critical importance to workflow systems [Georgakopoulos et al., 1995, Georgakopoulos, 1994, Jin et al., 1993]. WFMS should not only be functionally correct, but should also be robust in the view of failures. Workflow systems (both commercial and research prototypes) in their current state, lack adequate support for handling errors and failures in large-scale, heterogeneous, distributed computing environments [Georgakopoulos et al., 1995, Alonso and Schek, 1996, Kamath and Ramamritham, 1996, Sheth et al., 1996a, Leymann et al., 1996]. Failures could occur at various points and stages within the lifetime of the workflow enactment process. They could involve failures associated with the workflow tasks (such as unavailability of resources, incorrect input formats, internal application failures, etc.), failures within the workflow system components (such as schedulers, databases, etc.), and failures in the underlying infrastructure (such as hardware and network failures). Reliability in the context of workflows requires that tasks, their associated data, and the WMFS itself be recoverable in the event of failure, and that a well defined method exists for recovery.

A workflow process is heavily dependent on the organizational structure, and business policies within an organization. Workflows are activities that are *horizontal* in nature and are spread across the organizational spectrum as compared to transaction processing activities (e.g., database transactions) that are more *vertical* or *hierarchical* in nature and might form only part of the workflow process. In other words, hierarchical decomposition used for complex advanced transaction models is not sufficient for modeling workflows. A WFMS needs to support recovery of its tasks, associated data and the workflow process as a whole. The heterogeneous nature of workflow tasks and processing entities might preclude any transactional semantics that are required for assuring transactional behavior of the workflow or the constituent tasks themselves. A viable

recovery mechanism should be consistent with and should support the overall goal of the business process concerned.

Valuable research addressing recovery has been done in transaction management and ATMs [Bernstein et al., 1987, Gray and Reuter, 1993, Korth et al., 1990, Moss, 1987, Wachter and Reuter, 1992, Chen and Dayal, 1996] (see sections 1.2 and 1.3.1). A strictly data-centric approach has been used to address recovery issues in transaction processing. The problem domain of recovery in a WFMS is broader than that of transaction systems and ATMs due to its process-oriented focus, and diverse multi-system execution requirements. Although the ideas proposed in ATMs are limited in terms of the domains and environments they apply to, they are valuable in terms of their semantics and overall objectives. In the next section, we discuss the value and applicability of transaction concepts in the context of workflow recovery.

1.4.1 Transaction Concepts in Modeling Workflow Recovery

Earlier, we have discussed some of the ATMs that have been proposed in the literature. Recovery involves restoration of state - a concept which is voiced by transactional systems also. Later, we also reviewed some of the work in transactional workflows, and different approaches for incorporating transactional semantics into workflow models. We feel that transaction concepts are necessary for a recovery mechanism to be in place; however, basing a workflow recovery framework on a transactional (or advanced) transactional model would be naive.

As discussed in section 1.2.1, the hierarchical model in nested transactions [Moss, 1982] allows finer grained recovery, and provides more flexibility in terms of transaction execution. In addition to database systems, nested transactions can been used to model reliable distributed systems [Moss, 1987]. There is a lot to learn from work done in nested transactions. It provides a model for partitioning an application system into recoverable units; transaction failure is often localized within such models using retries and alternative actions. Workflow systems can borrow these ideas to a great extent, and tasks can be retried in the case of certain failures (e.g., failures related to unavailability of input data, or inadequacy of resources for executing a task at a processing entity), or alternate tasks can be scheduled to handle other more serious errors (e.g., when a certain number of retries fail, or when a task cannot be activated due to unavailability of a processing entity) that might cause a task to fail.

In the work on nested process management systems [Chen and Dayal, 1996] (discussed in section 1.3.1), the authors present a formal model of recovery that utilizes relaxed notions of isolation and atomicity within a nested transaction structure. Although, this model is more relaxed in terms of recovery

requirements as compared to nested transactions, it is strict for heterogeneous workflow environments that involve tasks that are non-transactional in nature. Moreover, the recovery model uses backward recovery of some of the child transactions for undoing the effects of a failed global transaction. The backward recovery approach has limited applicability in workflow environments in which it is either not possible to strictly reverse some actions, or is not feasible (from the business perspective) to undo them since this might involve an additional overhead or conflict with a business policy (e.g., in a banking application).

The notion of compensation is important in workflow systems. Undoing of incomplete transactions (or backward recovery) is an accepted repair mechanism for aborted transactions. However, this concept is not directly applicable to most real-world workflow tasks which are governed by actions that are in general permanent (e.g, human actions and legacy system processing). One can define a semantically inverse task (commonly referred to as *compensating tasks*), or a chain of tasks that could effectively undo or repair the damage incurred by a failed task within a workflow. In addition to Sagas, semantic transaction models have been proposed to address many such issues in which failure atomicity requirements have been relaxed. Compensation has been applied to tasks and groups of tasks (*spheres*) to support partial backward recovery in the context of the FlowMark WFMS [Leymann, 1995].

Work on flexible transactions[Elmagarmid et al., 1990, Zhang et al., 1994] discusses the role of *alternate transactions* that can be executed without sacrificing the atomicity of the overall global transaction. This provides a very flexible and natural model for dealing with failures. These concepts are applicable in workflow environments also. A prototype workflow system that implments a flexible transaction model has been discussed in [Alonso et al., 1996b].

In transactional models, the unit of recovery is a transaction. Each transaction has a predefined set of semantics that are compliant with the transaction processing system. The model for recovery in a workflow system is more involved since the recovery process should not only restore the state of the workflow system, but should proceed forward in a manner that is compliant with the overall organizational process.

Recovery of Workflow Tasks A task (activity or step) forms a basic unit of execution within a workflow model. A task is a logical unit of work that is used to satisfy the requirements of the business process that defines the workflow concerned. In database systems, it is sufficient to maintain *before* and *after* images of the data affected by a transaction to guarantee enough information needed to recover that transaction in case of its failure. Recovery of tasks, therefore, should be addressed from a broader perspective; in addition

to focusing on data-centric issues, one must focus on the overall business model associated with the actions within a task.

The tasks within a workflow could be arbitrarily complex and heterogeneous (i.e., transactional and non-transactional) in nature. A workflow model proposed in [Georgakopoulos et al., 1994] compares database transactions to tasks within a workflow, thereby regarding a workflow task to be the unit of recovery. This parallelism is valid when the tasks are relatively simple, obey transactional semantics and are executing within an environment that can enforce the transactional behavior of a group of tasks. Most real-world workflow applications and run-time environments are far more complex in nature and may be spread across arbitrary autonomous systems. Hence, a uniform recovery model based solely on transactional assumptions is inapplicable to commercial workflow systems.

Many task models have been defined for workflow systems [Attie et al., 1993, Krishnakumar and Sheth, 1995, Rusinkiewicz and Sheth, 1995]. In spite of this fact, it is difficult to determine the exact execution state of a task since these task models do not model detailed task execution. One could implement a workflow system involving special tasks that reveal their internal state to the WFMS layer; however, this workflow solution is not general enough to handle tasks that are diverse and arbitrarily complex in nature. Guaranteeing strict failure atomicity akin to that in database transactions is therefore difficult for workflow tasks. Hence, recovery of tasks should be addressed from a broader perspective. One should focus on the overall business process model when trying to decide the next action to be performed when resolving task failures.

In the case of non-transactional tasks, it is difficult to monitor the exact state of the task once it has been submitted for execution. This lack of control could leave the system in an undeterministic state in view of failures. In such a scenario, automatic recovery of a failed task becomes impossible due to lack of run-time feedback or transactional guarantees from the processing entities. The role of the human (e.g., workflow administrator) is important for recovery in such situations for determining the state of the failed task based on information that is external to the workflow system. In the METEOR$_2$ system [Worah, 1997], a special task is used to *cleanup* the remnants of such failures and to restore the workflow system to a consistent state. It could involve the role of a human or an application that is programmed to be able to reconfigure the data and applications associated with a task to restore it to a consistent state.

Recovery of Workflow Data Data plays an important role in workflow systems, as is in the case of a DBMS and a TP-system. Data recovery issues have been studied extensively in the context of database systems. *Logging* and *shadow paging* are common mechanisms used in transaction processing to record

state of critical data persistently. Several *checkpointing* mechanisms have been discussed in literature [Bernstein et al., 1987] to enhance the performance of the recovery process. These principles can be applied to workflow systems in situations related to making the state of the workflow components persistent and the recovery process more efficient. In the case of distributed WFMSs, it is also important to replicate data across machines to enhance data availability in the view of hardware and network failures. This problem, once again, has been studied extensively in the area of distributed databases; its applicability has also been studied in workflow systems [Alonso et al., 1994] to enhance their availability.

1.5 WORKFLOW ERROR HANDLING

Error handling is another critical area of workflow research that has not received adequate attention [Georgakopoulos et al., 1995, Alonso and Schek, 1996]. The cause of errors in workflow systems could be multifarious. Errors are logical in nature; they could be caused due to failures within the workflow system, or failures occurring at the task level.

Error handling in database systems has typically been achieved by aborting transactions that result in an error [Gray and Reuter, 1993]. Aborting or canceling a workflow task, would not always be appropriate or necessary in a workflow environment. Tasks could encapsulate more operations than a database transaction, or the nature of the business process could be forgiving to the error thereby not requiring an undo operation. Therefore, the error handling semantics of traditional transactional processing systems are too rigid for workflow systems.

A mechanism for dealing with errors in an ATM for long running activities was proposed in [Dayal et al., 1990, Dayal et al., 1991]. It supported forward error recovery, so that errors occurring in non-fatal transactions could be overcome by executing alternative transactions. Although, this model provides well defined constructs for defining alternative flow of execution in the event of errors, it is restrictive in terms of the types of activities (relaxed transactions) and the operating environment (a database) that form the long running process and therefore, it does not provide the error modeling capabilities of capturing workflow errors.

We can characterize the types of errors arising in a WFMS into three *broad* categories:

- *Infrastructure errors*: these errors result from the malfunctioning of the underlying infrastructure that supports the WFMS. These include communication errors such as loss of information, and hardware errors such as computer system crashes and network partitioning.

- *System errors*: these errors result from faults within the WFMS software. This could be caused due to faults in the hardware, or operating system. An example is the crash of a workflow scheduler.

- *Application and user errors*: these errors are closely tied to each of the tasks, or groups of tasks within the workflow. Due to its dependency on application level semantics, these errors are also termed as *logical* errors [Krishnakumar and Sheth, 1995]. For example, one such error could involve database login errors that might be returned to a workflow task that tries to execute a transaction without having permission to do so at a particular DBMS. A failure in enforcing inter-task dependencies between tasks is another example of an application error.

The above categorization is a descriptive model for categorizing errors within WFMSs. Large-scale WFMSs typically span across heterogeneous operating environments; each task could be arbitrarily complex in nature. To be able to detect and handle errors in such a diverse environment, we need a well-defined error model that would help us specify, detect and handle the errors in a systematic fashion. In 1.6.1.3 we define a hierarchical error model that forms the basis for handling errors in the METEOR$_2$ WFMS.

In the previous sections, we have discussed research done in the area of ATMs, transactional workflows, and the problem of error handling and recovery in WFMSs. In the next section we outline issues that are important for implementing a reliable WFMS. In doing so, we discuss a specific example of a WFMS that exploits many of the concepts from transactional systems and ATMs to include support for error handling and recovery.

1.6 TRANSACTIONS, ATMS AND RECOVERY IN LARGE-SCALE WFMSS

Pervasive network connectivity, coupled with the explosive growth of the Internet has changed our computational landscape. Centralized, homogeneous, and desktop-oriented technologies have given way to distributed, heterogeneous and network-centric ones. Workflow systems are no exceptions. They would typically be required to operate in such diverse environments in a *reliable* manner. Implementation of error handling and recovery in a WFMS is affected by numerous factors ranging from the underlying infrastructure (e.g., DBMS, TP-monitor, Lotus Notes, CORBA, Web), architecture of the supporting framework (e.g., centralized vs. distributed), nature of the processing entities (e.g., open vs. closed, transactional vs. non-transactional, human vs. computer system), type of tasks (user vs. system, transactional vs. non-transactional), and the nature of the workflow application (e.g., ad-hoc vs. administrative vs.

production). Most of these issues are beyond the purview of transaction-based systems, and therefore have not been adequately tackled by them.

A single recovery mechanism cannot be applied to all workflow applications due to the diversity of their business logic. Also, the variations in WFMS run-time architectures and execution environments would dictate the choice of suitable recovery mechanisms. A workflow is a collection of tasks; the tasks could be arbitrary in nature. It is impossible to include task specific semantics within a generalized recovery framework since task behavior is orthogonal to that of the workflow process. Nevertheless, a WFMS should provide the necessary infrastructure to support error handling and recovery as needed by the task. It should also provide tools to allow users to specify failure handling semantics that are conformant with the governing business process model. This is an important characteristic that differentiates failure handling in workflow systems from that in transaction processing where it suffices to satisfy the ACID properties for transactions.

ATMs provide techniques for handling failures (see Section 1.2). However, most of these ATMs do not discuss any aspects of implementation. Implementation of processes in workflow systems require support for business level *details* such as groups, roles, policies, etc. ATMs are weak in this aspect, since they define models that are focused towards the tasks themselves (in this case advanced transactions). Therefore, workflow systems are implemented at a higher level of granularity than ATMs. In fact, in [Alonso et al., 1996b] *sagas* and *flexible transactions* have been implemented using a WFMS.

WFMSs in distributed environments are dependent on inter-process communication across possibly heterogeneous computing infrastructures. In such systems, it is important that communication between processes is reliable. Transactional RPC mechanisms have been used in distributed transaction processing to guarantee reliable messaging between distributed processes. They can also be incorporated into workflow systems [Wodtke et al., 1996] to guarantee transactional messaging between the workflow components thereby increasing the level of fault-tolerance of the WFMS infrastructure.

TP-monitors have been used extensively to guarantee transactional semantics across distributed process spaces. They are, therefore, a viable middleware technology for implementing workflow systems. However, their use within a workflow environment comes with a lot of cost: 1) it is not feasible to impose infrastructural homogeneity (e.g., use of TP-monitors) across autonomous organizations, and 2) it is very expensive to maintain and administer especially when workflow process span multiple organizations. Emerging infrastructure technologies such as Web, CORBA, and DCOM, on the other hand, provide more open and cost effective solutions for implementing large-scale distributed workflow applications [Sheth et al., 1996b, Palaniswami et al., 1996]. In partic-

ular, the CORBA standard [OMG, 1995b] includes specifications for services [OMG, 1995a] such as the Object Transaction Service (OTS), the Concurrency Control Service, and the Persistence Service that can be combined to form a framework for achieving TP-monitor-like functionality in a HAD environments.

1.6.1 Error Handling and Recovery in the METEOR$_2$ WFMS

The study of workflow systems is inter-disciplinary, and stems from areas such as distributed systems, database management, software process management, software engineering, and organizational sciences [Sheth et al., 1996a]. Error handling and recovery are equally critical in these domains, and numerous solutions have been suggested to address these problems [Bhargava, 1987, Bernstein et al., 1987, Cristian, 1991, Saastamoinen, 1995].

In this section, we present an error handling and recovery framework that we have implemented for the distributed run-time of the METEOR$_2$ WFMS. This solution has been based on principles and implementation ideas that we have borrowed from related research in databases, advanced transaction models, software engineering and distributed systems. Due to lack of space, brevity is key in our discussions (for additional details, see [Worah, 1997]).

1.6.1.1 Overview of the METEOR$_2$ Workflow Model. The METEOR$_2$ workflow model is an extension of the METEOR [Krishnakumar and Sheth, 1995] model, and is focused towards supporting large-scale multi-system workflow applications in heterogeneous and distributed operating environments. The primary components of the workflow model include 1) processing entities and their interfaces, 2) tasks, 3) task managers, and 4) the workflow scheduler.

- *Processing Entity*: A processing entity is any user, application system, computing device, or a combination thereof that is responsible for completion of a task during workflow execution. Examples of processing entities include word processors, DBMSs, script interpreters, image processing systems, auto-dialers, or humans that could in turn be using application software for performing their tasks.

- *Interface*: The interface denotes the access mechanism that is used by the WFMS to interact with the processing entity. For example, a task that involves a database transaction could be submitted for execution using a command line interface to the DBMS server, or by using an application programming interface from within another application. In the case of a user task that requires user-input for data processing, the interface could be a Web browser containing an HTML form.

- *Task*: A task represents the basic unit of computation within an instance of the workflow enactment process. It could be either transactional or non-transactional in nature. Each of these categories can be further divided based on whether the task is an application, or a user-oriented task. *Application tasks* are typically computer programs or scripts that could be arbitrarily complex in nature. A *user task* involves a human performing certain actions that might entail interaction with a GUI-capable terminal. The human interacts with the workflow process by providing the necessary input for activating a user task. Tasks are modeled in the workflow system using well-defined task structures [Attie et al., 1993, Rusinkiewicz and Sheth, 1995, Krishnakumar and Sheth, 1995] that export the execution semantics of the task to the workflow level. A task structure is modeled as a set of states (e.g., initial, executing, fail, done), and the permissible transitions between those states. Several task structures have been defined - transactional, non-transactional, simple, compound, and two-phase commit [Krishnakumar and Sheth, 1995, Wang, 1995].

- *Task Manager*: A task manager is associated with every task within the workflow execution environment. The task manager acts as an intermediary between the task and the workflow scheduler. It is responsible for making the inputs to the task available in the desired format, for submitting the task for execution at the processing entity, and for collecting the outputs (if any) from the task. In addition, the task manager communicates the status of the task to the workflow scheduler.

- *Workflow Scheduler*: The workflow scheduler is responsible for coordinating the execution of various tasks within a workflow instance by enforcing inter-task dependencies defined by the underlying business process. Various scheduling mechanisms have been designed and implemented [Wang, 1995, Miller et al., 1996, Das, 1997, Palaniswami, 1997], ranging from highly centralized ones in which the scheduler and task managers reside within a single process, to a fully distributed one in which scheduling components are distributed within each of the distributed task manager processes.

We will focus our discussions on a run-time implementation of a distributed architecture for the METEOR$_2$ WFMS. A recovery framework has been defined for this architecture. The basic distributed model has been enhanced with additional functionality to 1) handle various forms of errors, 2) use transaction semantics at run-time, 3) monitor active workflow components, 4) recover failed components, and 5) log critical data that is necessary to restore the state of a failed workflow.

1.6.1.2 ORBWork: A Distributed Implementation of the METEOR$_2$ WFMS. ORBWork is a distributed run-time engine for the METEOR$_2$ WFMS. It has been implemented using CORBA [OMG, 1995b] and Web infrastructure technologies [Sheth et al., 1996b, Das, 1997, Worah, 1997]. The former provides the necessary distribution and communication capabilities for the workflow components, and the latter makes it possible for humans to interact with the Object Request Broker (ORB)† based workflow layer. The main components of ORBWork are shown in Figure 1.1. In this implementation, task managers, recovery units, data objects, monitors, and clean-up tasks are implemented as CORBA objects.

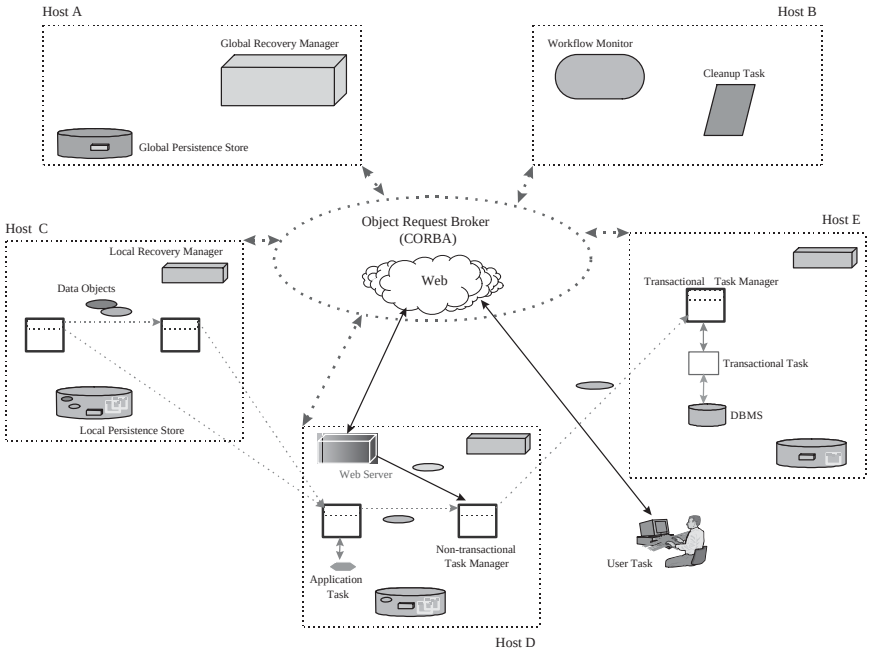

Figure 1.1 System Schematic for the Recovery Framework in ORBWork

In METEOR$_2$, the workflow process that defines the overall organizational process is captured in the form of a *workflow map* that is specified by a workflow designer. This determines the data and control dependencies that need to be enforced as part of the workflow scheduling process. Due to the distributed

†The Object Request Broker forms the core of the CORBA model; it is the middleware layer that makes it possible for distributed objects to communicate with each other. For details see [OMG, 1995b].

nature of the workflow engine, ORBWork does not have a *centralized* scheduling entity. The scheduling mechanism is embedded in each of the task managers.

Each task managers performs four primary functions: 1) task activation, 2) error handling and recovery of task and its own errors, 3) logging of task inputs, outputs, and its internal state, and 4) scheduling of dependent task managers as defined by the workflow process. Task managers communicate with each via the ORB using object method invocations. Due to the location transparency offered by CORBA, they are able to communicate seamlessly irrespective of the host they execute on.

Input and output data elements to the tasks are represented as CORBA objects internal to ORBWork. These CORBA objects are *wrappers* around the actual data elements. This allows workflow data objects to be distributed within the ORB environment. Task managers logically enforce workflow data dependencies and pass data by exchanging references to these data objects.

User tasks have associated "to-do" *worklists* (not shown in the figure) that provide a list of pending tasks for the user. User inputs form one of the implicit dependencies for a *user task manager*. User (human) tasks communicate with the task managers using HTML forms and Common Gateway Interface (CGI) functionality provided by Web servers. In our current implementation, CGI scripts are implemented as CORBA clients to user task manager objects. References to CORBA objects that encapsulate the user provided data are passed as inputs to the task manager.

ORBWork is subject to numerous errors and failures. The architecture of ORBWork, as described above, does not provide support for error handling, other than what is already inherent to the components themselves. The distributed nature of our workflow architecture alleviates problems associated with a single point of failure. This allows scope for incorporating fault-tolerant features into the framework. However, distribution adds to the complexity of the system in terms of management of the various components and detection of failures. This problem is compounded due to the asynchronous communication paradigm used in workflow communication models. Moreover, the communication infrastructure is subject to failures, and could adversely affect workflow enactment. In the following two sections, we describe the error model that we have used to capture such errors, and the failure handling components that form our recovery framework. For a detailed discussion on ORBWork, see [Das, 1997].

1.6.1.3 Modeling Errors in METEOR$_2$. The METEOR$_2$ error model has been defined in a hierarchical manner. We have based it on the layered nature of the METEOR$_2$ workflow model. It enables us to describe and classify the various errors that occur during workflow execution. This, in effect,

makes it possible to modularize our error handling algorithms during workflow execution. Errors are detected and masked as close to the point of occurrence as possible to prevent them from propagating to other, unrelated components of the WFMS. We use a three-tiered approach to classify errors within the METEOR$_2$ workflow model:

1. *Task and Workflow Errors*: this class forms the lowest level within our hierarchy and includes all errors that are specific to tasks, and their inter-task dependencies. Application and user errors (as discussed in Section 1.5) are defined and modeled at this level. The workflow designer is responsible for defining these errors during the workflow definition process. The workflow system does not preclude a task from handling its errors on its own; in such cases, only unhandled errors would be categorized as task errors within the WFMS. Some of these errors may have implications on the whole workflow process. A task error that cannot be resolved is eventually reported to its task manager; such an error falls into the category of task manager errors.

2. *Task Manager Errors*: this class of errors involves all task errors that could not be resolved at the task level (as described earlier), and errors that are specific to the task manager itself. For example, the latter includes errors such as

   - not being able to prepare the inputs for the task,
   - not being able to submit a task for execution,
   - not being able to recover the state of task during failure recovery, and
   - not being able to handle a task error that might have occurred.

   A task manager error that remains unhandled is reported as a workflow error to the scheduler.

3. *WFMS Errors*: These are the highest level of errors within our model and include

   - system errors that affect the task scheduling mechanism,
   - communication errors between the scheduler and the task managers,
   - other failures in workflow components that are common to all instances of a workflow type (e.g., failure recovery units, log managers, etc.), and
   - errors that could not be handled at the level of the task manager.

In our model, task and workflow errors are logical in nature. Error handling at this level is achieved by retries, aborts, cancellations, and by trying alternate tasks. Task manager and WFMS errors are system errors caused due to failures within the WFMS software. Task manager errors are either handled at the level of the task manager itself (e.g., retrying task submission for a task that cannot be submitted). WFMS errors are handled by the recovery components within the WFMS, or by a human that would be provided with information necessary to handle the error.

Principles relating to classification of errors, and handling them in a modular fashion have been commonplace in computer architecture, programming languages, and software engineering. We have mapped the ideas to our workflow model, and have defined the error-handling semantics so that they are in synchrony with the overall business process that defines the workflow. Although, this model has been applied within the METEOR$_2$ WFMS, in principle, it is applicable to any workflow model that has a well-defined modular architecture. The error handling capabilities in ORBWork, are developed on the basis of this error model.

1.6.1.4 Recovery Framework in ORBWork. In this section, we describe the recovery framework for ORBWork (see Figure 1.1). In defining the recovery framework, we have extended the ORBWork workflow engine in terms of being able to handle failures ranging from the task level to the level of the workflow system components. The recovery model assumes a distributed, component-based architecture for the WFMS, and a communication mechanism (in this case CORBA) that makes it possible to interact with components across host boundaries.

Persistence is an essential part of our recovery framework. We have used an object-oriented approach wherein the various workflow components are responsible for logging their respective states to stable storage. This approach is very similar to the notion of recoverable objects in the distributed object-oriented framework of *Arjuna*[Shrivastava et al., 1991]. In our model, data objects inherit from a base interface that attributes it with capabilities to *save* and *restore* its state at runtime from stable storage. A *Local Persistence Store* (LPS) is used as the stable storage mechanism for logging local data critical for recovery purposes. We have used a DBMS as the basis for our LPS. A DBMS provides transactional capabilities to log data. A *Global Persistence Store* is used for logging at the level of the GRM. Logging is done at various stages within the workflow enactment process. For example, 1) task Managers log the state of their tasks (including error codes returned by the task, for future debugging and error recovery), inputs that they receive from other task managers, and

outputs that they send out to dependant task managers; 2) data objects log the state of their data they encapsulate.

Failures in distributed systems are hard to detect, unless there is a fault-tolerant detection mechanism in place. This problem is compounded especially when most of the components communicate in an asynchronous mode. Distributed workflow systems fall into this category due to their asynchronous coordination model. In ORBWork, we have provided additional services for *monitoring* distributed components, to address the issue of failure detection. In this regard, we have borrowed ideas from other work done in reliable distributed systems [Birman and Renesse, 1994, Maffeis, 1996].

Task Managers and data objects on each host are monitored by a *Local Recovery Manager* (LRM) process executing on the same machine. On startup, the task managers and data objects, *register* with the LRM on their host. Once, these components are no longer required within the workflow process, they *deregister* from the LRM. The LRM maintains a *watch-list* of currently registered components that are supposed to be executing as part of the workflow process instance on its host. When an object registers with the LRM, the LRM logs this message and appends it to the list. On deregistration, these objects are removed from the list. The LRM contains a *watchdog* that periodically, polls each of the components on the watch-list to ensure their liveliness. When a failed component is detected, the LRM reactivates the component, which in turn, restores its own state from local logs. The LRM checkpoints it logical view of the local system to the local log to enable its own recovery. In addition to the LRM, each host contains a daemon process called the *Local Activation Daemon* (LAD) (not shown in the figure) that is endowed with the ability to create processes (for the various CORBA objects) on the various hosts.

A *Global Recovery Manager* (GRM) executing on a reliable host in the workflow execution environment monitors the liveliness of all the LRMs and is responsible for reactivating any failed LRMs. On recovery, the failed LRMs synchronize the state of their respective local systems based on their local logs and create any task managers that might have failed in the interim. Due to the infancy of the CORBA standard, and unavailability of many of its object services, we had to rely on programmatic efforts to implement many of the features that we would have otherwise liked to have been supplied by the ORB vendor. The implementation of error handling is achieved via the use of exceptions and *try-catch* blocks that help to isolate the normal flow of execution from the abnormal case during run-time.

Local *configuration files* (not shown in the figure) are used on each host by the LAD. These files are used for directory lookup for the various components (i.e., task manager, data object, LRM, GRM) during activation or recovery of the processes.

During the definition of the workflow design, it might not be feasible to capture all errors and causes of failures that might occur during the enactment process. Also, especially in the case of non-transactional tasks, it is not always possible to *undo* the effects of a task that might have completed partially. We therefore feel that the role of a human is indispensable within the workflow recovery framework. In our model, we have allocated a special human-performed task, called the *cleanup task* to serve the functionality of bringing the system to a consistent state after such irrecoverable failure. This mode of restoration is used only when the WFMS is unable to handle the recovery process automatically.

Let us summarize the main characteristics of our recovery framework.

- Workflow recovery is implemented in a distributed CORBA and Web based execution environment.
- A notion of hierarchical monitoring of workflow components has been used to detect failures, and to initiate the recovery process (i.e., GRM monitors the LRMs; LRMs monitor task managers and data objects; task managers monitor tasks). This allows failures to be localized, and their effects to be masked as close to the point of occurrence as possible.
- The recovery model ensures that there is no single point of failure. Therefore, the failure of a host does not significantly affect the performance of tasks within another (unless they are directly dependant on each other).
- The performance of the workflow system would degrade *progressively* in the case of failures; however, once the failure has been restored, the WFMS would execute normally.
- Each workflow component is responsible for logging its own state. The persistence mechanism used is also local to the component itself.
- The workflow components are responsible for managing their own recovery actions once they have been *recreated*.
- The recovery mechanism is semi-automated. The role of the human is crucial both during the workflow design process and the enactment. The workflow designer specifies the run-time behavior of the error handling and forward recovery mechanism. The workflow administrator is responsible for fixing drastic system failures (e.g., machine crash, network partitioning), and for *cleanup* of failed tasks that cannot be handled by the WFMS.
- The distribution and hierarchical nature of the recovery mechanism makes the system scaleable and manageable.

In this section we have briefly described the design and implementation of error handling and recovery in the distributed run-time of the METEOR$_2$ WFMS (see [Worah, 1997] for more details). We have used this discussion to illustrate the applicability of concepts and basic mechanisms from traditional and ATMs within a practical workflow execution environment. Also, our discussion is suggestive of the need to look for solutions beyond ATMs for addressing reliability issues in WFMSs.

1.7 TYPES OF TRANSACTIONS IN THE REAL-WORLD: BEYOND DATABASE TRANSACTIONS

As practicing researchers, the idea of using related transaction models for modeling workflows was appealing to us. We felt that such a model could provide a rigor or structure that was lacking in the work on workflow management [Ansari et al., 1992, Breitbart et al., 1993]. There are few, if any, examples of successes in developing systems that implement ATMs for significant commercial, large-scale multi-system applications.

Requirements of such applications include:

1. capability to explicitly define the functionality and organizational structure of organizational process involved,
2. support of coordination and execution of tasks in heterogeneous intra- and inter-enterprise environments,
3. modeling and support for human involvement with the run-time system, and
4. error handling and failure recovery.

Workflow management is specifically defined to address these real-world challenges. It provides the tools to integrate humans, computer systems, information resources and organizational processes into a unified solution. Hence, the requirements of WFMSs are far more challenging than those faced by current database systems [Alonso and Schek, 1996]. In workflow applications, database resources might comprise only a part of the entire solution. For a task that entirely interacts with a DBMS, executing it as a transaction is often a desirable choice. At the same time, workflows involve other user and application tasks (e.g., tasks that interact with legacy systems) that are non-transactional in nature.

Due to the wide acceptance and applicability of workflows to application domains that extend beyond transaction based (primarily database related) environments, the term *transaction* is being used in a more loose manner with

various connotations. These interpretations are based on: 1) the type of tasks and processing entities that are part of the workflow process, 2) the application domain or semantics of the organizational process that is being modeled, 3) the communication infrastructure that is used to develop the WFMS, and 4) transactional or advanced transactional semantics (such as relaxed isolation and atomicity) that can be attributed to the tasks, sub-workflow, or the workflow as a whole. It is important to understand each of these interpretations to be able to appreciate the similarities and differences between transactions from the world of database systems and those involved in the realm of multi-system workflow management systems. Let us consider some of the frequently encountered interpretations for the term *transactions* in the context of real-world workflow applications and WFMS that support workflow applications:

1. **Task specific interpretation in databases and distributed transaction processing.** In general, a workflow task is considered to be a *black box* that is functional in nature, i.e., the functionality of the task is orthogonal to that of the workflow process [Alonso et al., 1994]. The tasks themselves could be transactional or non-transactional in nature [Rusinkiewicz and Sheth, 1995, Krishnakumar and Sheth, 1995]. Transactional tasks are those that minimally support the atomicity property and maximally support all ACID properties of traditional transaction models [Miller et al., 1996, Krishnakumar and Sheth, 1995]. These tasks typically include those that interact with a DBMS by using *BEGIN_ TRANSACTION - END_TRANSACTION* semantics, contracts (stored procedures), and two-phase commit (2PC) tasks [Wang, 1995, Miller et al., 1996] for synchronizing transactions across multi-DBMSs. In addition, tasks that use the *XA-Protocol* [Gray and Reuter, 1993] based RPC to communicate with transactional processing entities such as a TP-monitor in a distributed environment [Wodtke et al., 1996] can also be included in this category. Non-transactional tasks are used to include applications that cannot ensure isolation or atomicity as a part of the workflow process. Such task types are commonplace in the real-world and involve activities requiring interaction with humans, legacy systems, and others that interface with other processing entities that do not provide transactional support (e.g., HTTP servers, Lotus Notes, file systems, word processors, spreadsheets and decision support systems).

2. **Domain specific interpretation.** The move from a paper-based society to a paper-less one, and the increasing popularity of electronic commerce have led to evolution of standards for electronic data exchange across organizations. Some of these include (EDI) standards such as ANSI Accredited Standards Committee (ASC) X12 that are used in numerous

commercial settings (e.g., ANSI 270 and 271 transactions for healthcare eligibility inquiry and response used in [Sheth et al., 1996b]), and the ANSI HL7 standard that is used specifically in the medical domain. The term *transaction* in this setting refers to the exchange of sufficient data in a standard electronic format necessary to complete a particular business action often using domain specific information. This view of a transaction tends to focus more on business requirements and contracts rather than on the need for maintaining data consistency within a database or to support atomicity or other transactional property between communicating processes or for a RPC call. Workflow technology is being applied in various forms to application domains such as manufacturing, banking, healthcare and finance that use domain specific *transaction* formats extensively. One of the tasks within a workflow process could involve sending data from one information system to another using an EDI *transaction*. At the receiving end, another workflow task could write the data that it receives to a DBMS in a *transactional* (having ACID properties) manner. The semantics associated with each of these transactions are different. Hence, the WFMS would have to be designed so that it can deal with different *transaction* forms in an appropriate manner.

3. **Business-process specific interpretation.** Database transactions and transaction processing aim at preserving data consistency and ensuring reliability in case of faults and failures. These semantics cannot be applied directly to workflow systems since tasks within a workflow process are both transactional and non-transactional in nature. However, at the same time, workflow systems should be correct and reliable. Correctness and reliability in the case of workflow systems is more applicable from a broader perspective - that of the *organizational process* involved in addition to the data that forms a part of the process. According to [Eder and Liebhart, 1995], a workflow *transaction* should ensure consistency from the business process point of view. The notion of a workflow transaction according to this view, is broader as compared to that of traditional transactions. Implementation support for such a concept would require an additional layer of control than that provided in transaction processing since workflows include features (e.g., roles, worklists, error handling) that are not available in (advanced) transaction models and transaction processing systems.

4. **Infrastructure specific interpretation.** Workflow management systems are large-scale applications that can be implemented using various infrastructure technologies such as Customized Transaction Management (CTM) [Georgakopoulos et al., 1995], Distributed Object Management

specifically using CORBA [Georgakopoulos et al., 1994, Miller et al., 1996, Sheth et al., 1996b, Wodtke et al., 1996, Schuster et al., 1994], World Wide Web [Palaniswami et al., 1996, Sheth et al., 1996b, Technologies, 1995], TP-monitors [Wodtke et al., 1996], Lotus Notes [Reinwald and Mohan, 1996] and security services (as in *secure transactions* supported in the electronic commerce and Web-based services). The concept of *transactions* has been addressed in many of these technologies to some extent. For example CORBA provides an Object Transaction Service as a part of the Common Object Services Specification [OMG, 1995a] that enables objects in distributed environments to take part in a *transactional context*; TP-monitors also provide transactional semantics in a distributed environment. The HTTP protocol used in the Web paradigm, on the other hand, does not provide any transactional semantics. Hence, we see that different interpretations of transactions are supported by each of these infrastructures.

From the above discussion, it is important to observe that the notion of transactions in workflow management is more general compared to that in transaction processing and DBMSs.. Its interpretation could involve various variables associated with the factors mentioned above. *Unlike advanced transaction systems, WFMS interact with database systems if required as part of the organizational process, however, this is not their primary focus.*

1.8 CONCLUSION

We view workflow management as an attractive approach to *programming in the large* for enterprise applications. Tasks within a workflow are modeled at a higher degree of granularity than traditional database transactions (i.e., component transactions in a ATM or subtransactions in a distributed transaction). The tasks themselves could be either transactional (e.g., database transactions, and processes interacting with a TP-monitor) or non-transactional (e.g., human-oriented activities, and processes that do not observe one or more of the transaction properties). Also, most real-world workflow processes involve activities that are long running in nature and execute in distributed and heterogeneous environments. The processing entities that execute or carry out a task might not support the protocol for guaranteeing transaction behavior. At the same time, it is desirable that workflow systems be reliable and ensure correct execution of processes just as transactions guarantee such characteristics for ensuring data consistency. It has been accepted that strict ACID transactions do not have direct applicability in the workflow domain as workflow systems differ to a large degree from traditional database systems.

In our perspective, the role of ATMs in workflow systems is of a supportive nature. Advanced transaction modeling concepts are quite restricted in terms of being directly applicable in process-oriented, large-scale workflow applications that run in HAD computing environments. Workflow systems today are still weak in terms of characteristics such as fault-tolerance, consistency, and in their support for recovery in case of exceptions and failures. ATMs have addressed most of these problems in the domain of database systems. Research in the areas of workflow systems can benefit from these approaches from a conceptual point of view.

Transactional semantics such as atomicity and isolation in their strict sense are not practical in workflow systems since tasks in a workflow domain are generally long-lived and could themselves be non-transactional in nature. Many of the solutions for recovery in transaction processing systems can be used to address recovery issues in workflow systems, for example, advanced transaction concepts such as compensation can be mapped to the workflow domain in terms of a compensating task that could be used to *undo* (often partially) what was done by an incomplete task; logs similar to those in transaction processing could be maintained for recording the history of the workflow process, thereby aiding in the recovery process [Krishnakumar and Sheth, 1995, Alonso et al., 1994, Eder and Liebhart, 1996].

To address many of these advanced issues, workflow systems should borrow ideas that have been used effectively in concurrent, large-scale distributed and database systems, but should not rely entirely on them as many of these systems have developed models for environments that are limited in scope as compared to that in workflow systems.

In conclusion, we summarize the observations we have made in this chapter:

- There are several interpretations for *transactions* in organizational processes today and all or most of them may need to be accommodated in a workflow technology that supports organizational processes.

- Features offered by ATMs meet a very restricted subset of requirements of large-scale enterprise-wide workflow systems (see the appendix for a normative comparison of ATMs and workflow systems).

- We do not see ATMs as being a primary basis for modeling and executing workflow systems that have real-world commercial applicability. However these models provide useful features (e.g., relaxed atomicity, relaxed isolation, concurrency control and recovery) which can be used in components (e.g., tasks) that form a part of a WFMS. Traditional transaction processing and ATMs provide valuable concepts that can be applied towards partly solving the problem of error handling and recovery in WFMSs.

- Implementing reliable large-scale WFMSs involve requirements that are beyond the capabilities of transaction systems and ATMs (e.g., distribution of the workflow architecture, heterogeneity of the operating environment, business process governing the workflow, organizational structure of the enterprise, nature of the tasks, etc.). A lot of valuable research has been done on error handling and recovery in the areas of distributed systems, software engineering, and organizational sciences. Research and development in the domain of reliable WFMS should leverage these efforts to supplement the limitations of traditional transaction and ATM based systems.

There is a need for multi-disciplinary research to address the challenging issues raised by emerging workflow technology. Humans are an essential part of any organizational process, and human work involves many diverse issues. Therefore, research involving expertise from multiple disciplines is most likely to bring the highest return. Information is another critical asset of any organization, as discussed in [Sheth et al., 1996a]; we believe that more human-centric approaches with integral support for information management are needed for a successful workflow technology. We need to look beyond the capabilities provided by transaction processing systems and ATMs in modeling the complexities of large-scale, mission-critical workflow applications of the future.

Acknowledgments

METEOR$_2$ is a group project, and our discussion related to it reflects contributions of numerous past and current members of the project (*http://lsdis.cs.uga.edu/workflow*). Current active members include, S. Das, Prof. K. Kochut, Prof. J. Miller, D. Palaniswami, Prof. A. Sheth, D. Worah and K. Zheng.

This research was partially done under a cooperative agreement between the National Institute of Standards and Technology Advanced Technology Program (under the HIIT contract, number 70NANB5H1011) and the Healthcare Open Systems and Trials, Inc. consortium. See URL:http://www.scra.org/hiit.html. Additional partial support and donations are provided by Post Modern Computing, Illustra Information Technology, and Hewlett-Packard Labs.

References

[Alonso et al., 1996a] Alonso, G., Agrawal, D., and Abbadi, A. E. (1996a). Process Synchronization in Workflow Management Systems. In *Proc. of 8th. IEEE Symposium on Parallel and Distributed Processing*, New Orleans, LA.

[Alonso et al., 1996b] Alonso, G., Agrawal, D., Abbadi, A. E., Kamath, M., and Gunthor, R. (1996b). Advanced Transaction Models in Workflow Contexts.

In *Proc. of 12th. IEEE Intl. Conference on Data Engineering*, pages 574–581, New Orleans, LA.

[Alonso et al., 1995] Alonso, G., Agrawal, D., Abbadi, A. E., Mohan, C., Kamath, M., and Guenthoer, R. (1995). Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Proc. of the IFIP Working Conference on Information Systems Development for Decentralized Organizations*, pages 1–18, Trondheim, Norway.

[Alonso et al., 1994] Alonso, G., Kamath, M., Agrawal, D., Abbadi, A. E., Gunthor, R., and Mohan, C. (1994). Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center.

[Alonso and Schek, 1996] Alonso, G. and Schek, H. (1996). Research Issues in Large Workflow Management Systems. In *[Sheth, 1996]*, Athens, GA.

[Ansari et al., 1992] Ansari, M., Ness, L., Rusinkiewicz, M., and Sheth, A. (1992). Using Flexible Transactions to Support Multi-system Telecommunication Applications. In *Proc. of the 18th Intl. Conference on Very Large Data Bases*, pages 65–76, Vancouver, Canada.

[Attie et al., 1993] Attie, P., Singh, M., Sheth, A., and Rusinkiewicz, M. (1993). Specifying and Enforcing Intertask Dependencies. In *Proc. of the 19th Intl. Conference on Very Large Data Bases*, pages 134–145, Dublin, Ireland.

[Barbara et al., 1996] Barbara, D., Mehrotra, S., and Rusinkiewicz, M. (1996). INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management, Special Issue on Multidatabases*, 7(1):5–15.

[Bernstein et al., 1987] Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

[Bhargava, 1987] Bhargava, B. K., editor (1987). *Concurrency Control and Reliability in Distributed Systems*. Van Nostrand Reinhold Company, New York.

[Biliris et al., 1994] Biliris, A., Dar, S., Gehani, N., Jagadish, H., and Ramamritham, K. (1994). ASSET: A System for Supporting Extended Transactions. In *Proc. of ACM SIGMOD Conference on Management of Data*, pages 44–54, Minneapolis, MN.

[Birman and Renesse, 1994] Birman, K. P. and Renesse, R. V. (1994). *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press.

[Bonner et al., 1996] Bonner, A., Shruf, A., and Rozen, S. (1996). LabFlow-1: A Database Benchmark for High Throughput Workflow Management. In *Proc. of the 5th. Intnl. Conference on Extending Database Technology*, pages 25–29, Avignon, France.

[Breitbart et al., 1993] Breitbart, Y., Deacon, A., Schek, H., and Sheth, A. (1993). Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. *SIGMOD Record*, 22(3):23–30.

[Chen et al., 1993] Chen, J., Bukhres, O., and Elmagarmid, A. K. (1993). IPL: A Multidatabase Transaction Specification Language. In *Proc. of the 13th Intl. Conference on Distributed Computing Systems.*

[Chen and Dayal, 1996] Chen, Q. and Dayal, U. (1996). A Transactional Nested Process Management System. In *Proc. of 12th. IEEE Intl. Conference on Data Engineering*, pages 566–573, New Orleans, LA.

[Chrysanthis and Ramamritham, 1991] Chrysanthis, P. and Ramamritham, K. (1991). A formalism for extended transaction models. In *Proc. of the 17th. Very Large Data Bases Conference.*

[Chrysanthis and Ramamritham, 1992] Chrysanthis, P. and Ramamritham, K. (1992). ACTA: The SAGA Continues. In *[Elmagarmid, 1992]*, chapter 10. Morgan Kaufmann.

[Coalition, 1994] Coalition, T. W. M. (1994). Glossary – A Workflow Management Coalition Specification. Technical report, The Workflow Management Coalition, Brussels, Belgium. URL: http://www.aiai.ed.ac.uk/WfMC/.

[Cristian, 1991] Cristian, F. (1991). Understanding fault tolerant distributed systems. *Communications of the ACM*, 34(2):57–78.

[Das, 1997] Das, S. (1997). ORBWORK: The CORBA-based Distributed Engine for the METEOR$_2$ Workflow Management System. Master's thesis, University of Georgia, Athens, GA. In preparation. URL: http://LSDIS.cs.uga.edu/.

[Dayal et al., 1990] Dayal, U., Hsu, M., and Ladin, R. (1990). Organizing Long-Running Activities with Triggers and Transactions. In *Proc. of the ACM SIGMOD Conference on Management of Data.*

[Dayal et al., 1991] Dayal, U., Hsu, M., and Ladin, R. (1991). A Transactional Model for Long-running Activities. In *Proc. of the 17th. Intl. Conference on Very Large Data Bases*, pages 113–122, Barcelona, Spain.

[Eder and Liebhart, 1995] Eder, J. and Liebhart, W. (1995). The Workflow Activity Model WAMO. In *Proc. of the 3rd. Int. Conference on Cooperative Information Systems*, Vienna, Austria.

[Eder and Liebhart, 1996] Eder, J. and Liebhart, W. (1996). Workflow Recovery. In *Proc. of the 1st. IFCIS Conference on Cooperative Information Systems*, Brussels, Belgium.

[Elmagarmid, 1992] Elmagarmid, A., editor (1992). *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, San Mateo, CA.

[Elmagarmid et al., 1990] Elmagarmid, A. K., Leu, Y., Litwin, W., and Rusinkiewicz, M. (1990). A Multidatabase Transaction Model for InterBase. In *Proc. of the 16th. Intl. Conference on Very Large Data Bases*, pages 507–518, Brisbane, Australia.

[Fischer, 1995] Fischer, L. (1995). *The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering.* Future Strategies, Inc., Alameda, CA, 2nd. edition.

[Garcia-Molina and Salem, 1987] Garcia-Molina, H. and Salem, K. (1987). Sagas. In *Proc. of ACM SIGMOD Conference on Management of Data*, pages 249–259, San Francisco, CA.

[Garcia-Molina et al., 1991] Garcia-Molina, H., Salem, K., Gawlick, D., Klein, J., and Kleissner, K. (1991). Modeling Long-Running Activities as Nested Sagas. *IEEE Data Engineering Bulletin*, 14(1).

[Georgakopoulos, 1994] Georgakopoulos, D. (1994). Workflow Management Concepts, Commercial Products, and Infrastructure for Supporting Reliable Workflow Application Processing. Technical Report TR-0284-12-94-165, GTE Laboratories Inc., Waltham, MA.

[Georgakopoulos and Hornick, 1994] Georgakopoulos, D. and Hornick, M. (1994). A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows. *Intnl. Journal of Intelligent and Cooperative Information Systems*, 3(3):599–617.

[Georgakopoulos et al., 1994] Georgakopoulos, D., Hornick, M., Krychniak, P., and Manola, F. (1994). Specification and Management of Extended Transactions in a Programmable Transaction Environment. In *Proc. of the 10th. Intnl. Conference on Data Engineering*, pages 462–473, Houston, TX.

[Georgakopoulos et al., 1995] Georgakopoulos, D., Hornick, M., and Sheth, A. (1995). An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–154.

[Gray and Reuter, 1993] Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, San Mateo, CA.

[Hsu, 1993] Hsu, M., editor (1993). *Special Issue on Workflow and Extended Transaction Systems.* IEEE Computer Society, Washington, DC.

[Hsu, 1995] Hsu, M., editor (1995). *Special Issue on Workflow Systems.* IEEE Computer Society, Washington, DC.

[Jin et al., 1993] Jin, W., Ness, L., Rusinkiewicz, M., and Sheth, A. (1993). Concurrency Control and Recovery of Multidatabase Work Flows in Telecommunication Applications. In *Proc. of ACM SIGMOD Conference.*

[Joosten et al., 1994] Joosten, S., Aussems, G., Duitshof, M., Huffmeijer, R., and Mulder, E. (1994). *WA-12: An Empirical Study about the Practice of Workflow Management.* University of Twente, Enschede, The Netherlands. Research Monograph.

[Kamath and Ramamritham, 1996] Kamath, M. and Ramamritham, K. (1996). Bridging the gap between Transaction Management and Workflow Management. In *[Sheth, 1996]*, Athens, GA.

[Korth et al., 1990] Korth, H. F., Levy, E., and Silberschatz, A. (1990). A Formal Approach to Recovery by Compensating Transactions. In *Proc. of the 16th. Intl. Conference on Very Large Data Bases*, Brisbane, Australia.

[Krishnakumar and Sheth, 1995] Krishnakumar, N. and Sheth, A. (1995). Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. *Distributed and Parallel Databases*, 3(2):155–186.

[Krychniak et al., 1996] Krychniak, P., Rusinkiewicz, M., Chichocki, A., Sheth, A., and Thomas, G. (1996). Bounding the Effects of Compensation under Relaxed Multi-Level Serializability. *Distributed and Parallel Database Systems*, 4(4):355–374.

[Leymann, 1995] Leymann, F. (1995). Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems. In *GI-Fachtagung Datenbanken in Buro Technik und Wissenchaft*, Dresden, Germany. Springer-Verlag.

[Leymann et al., 1996] Leymann, F., Schek, H. J., and Vossen, G. (1996). Transactional workflows. Dagstuhl Seminar 9629.

[Maffeis, 1996] Maffeis, S. (1996). PIRANHA: A Hunter of Crashed CORBA Objects. Technical report, Olsend & Associates, Zurich.

[Medina-Mora and Cartron, 1996] Medina-Mora, R. and Cartron, K. W. (1996). ActionWorkflow in Use: Clark County Department of Business License. In *Proc. of the 12th. Intnl. Conference on Data Engineering*, New Orleans, LA.

[Miller et al., 1996] Miller, J. A., Sheth, A. P., Kochut, K. J., and Wang, X. (1996). CORBA-based Run-Time Architectures for Workflow Management Systems. *Journal of Database Management, Special Issue on Multidatases*, 7(1):16–27.

[Mohan et al., 1995] Mohan, C., Alonso, G., Guenthoer, R., and Kamath, M. (1995). Exotica: A Research Perspective on Workflow Management Systems. *In [Hsu, 1995]*, 18(1):19–26.

[Moss, 1982] Moss, J. (1982). Nested Transactions and Reliable Distributed Computing. In *Proc. of the 2nd. Symposium on Reliability in Distributed Software and Database Systems*, pages 33–39, Pittsburgh, PA. IEEE CS Press.

[Moss, 1987] Moss, J. (1987). Nested transactions: An introduction. In *[Bhargava, 1987]*. Van Nostrand Reinhold Company.

[OMG, 1995a] OMG (1995a). CORBAservices: Common Object Services Specification. Technical report, Object Management Group.

[OMG, 1995b] OMG (1995b). The Common Object Request Broker: Architecture and Specification, revision 2.0. Technical report, Object Management Group.

[Palaniswami, 1997] Palaniswami, D. (1997). WebWork: The Web-based Distributed Engine for the METEOR$_2$ Workflow Management System. Master's thesis, University of Georgia, Athens, GA. In preparation.

[Palaniswami et al., 1996] Palaniswami, D., Lynch, J., Shevchenko, I., Mattie, A., and Reed-Fourquet, L. (1996). Web-based Multi-Paradigm Workflow Automation for Efficient Healthcare Delivery. In *[Sheth, 1996]*, Athens, GA.

[Perry et al., 1996] Perry, D., Porter, A., Votta, L., and Wade, M. (1996). Evaluating Workflow and Process Automation in Wide-Area Software Development. In *[Sheth, 1996]*, Athens, GA.

[Reinwald and Mohan, 1996] Reinwald, B. and Mohan, C. (1996). Structured Workflow Management with Lotus Notes Release 4. In *Proc. of 41st. IEEE Computer Society Intnl. Conference*, pages 451–457, Santa Clara, CA.

[Rusinkiewicz and Sheth, 1995] Rusinkiewicz, M. and Sheth, A. (1995). Specification and Execution of Transactional Workflows. In Kim, W., editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*. ACM Press, New York, NY.

[Saastamoinen, 1995] Saastamoinen, H. (1995). *On the Handling of Exceptions in Information Systems*. PhD thesis, University of Jyvaskyla.

[Schuster et al., 1994] Schuster, H., Jablonski, S., Kirsche, T., and Bussler, C. (1994). A Client/Server Architecture for Distributed Workflow Management Systems. In *Proc. of the 3rd. Intnl. Conference on Parallel and Distributed Information Systems*, pages 253–256, Austin, TX.

[Sheth, 1995] Sheth, A. (1995). Tutorial Notes on Workflow Automation: Application, Technology and Research. Technical report, University of Georgia. presented at ACM SIGMOD, San Jose, CA, URL: http://LSDIS.cs.uga.edu /publications.

[Sheth, 1996] Sheth, A. (1996). Proc. of the NSF workshop on workflow and process automation in information systems. University of Georgia. URL: http://LSDIS.cs.uga.edu/activities/NSF-workflow.

[Sheth et al., 1996a] Sheth, A., Georgakopoulos, D., Joosten, S., Rusinkiewicz, M., Scacchi, W., Wileden, J., and Wolf, A. (1996a). Report from the NSF

Workshop on Workflow and Process Automation in Information Systems. Technical report, University of Georgia, UGA-CS-TR-96-003. URL: http://LSDIS.cs.uga.edu/activities/NSF-workflow.

[Sheth and Joosten, 1996] Sheth, A. and Joosten, S. (1996). Workshop on Workflow Management: Research, Technology, Products, Applications and Experiences.

[Sheth et al., 1996b] Sheth, A., Kochut, K. J., Miller, J., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., and Shevchenko, I. (1996b). Supporting State-Wide Immunization Tracking using Multi-Paradigm Workflow Technology. In *Proc. of the 22nd. Intnl. Conference on Very Large Data Bases*, Bombay, India.

[Sheth and Rusinkiewicz, 1993] Sheth, A. and Rusinkiewicz, M. (1993). On Transactional Workflows. *In [Hsu, 1993].*

[Shrivastava et al., 1991] Shrivastava, S. K., Dixon, G. N., and Parrington, G. D. (1991). An overview of Arjuna: A programming system for reliable distributed computing. *IEEE Software*, 8(1):66–73.

[Smith, 1993] Smith, T. (1993). The Future of Workflow Software. *INFORM*, pages 50–51.

[Tang and Veijalainen, 1995] Tang, J. and Veijalainen, J. (1995). Transaction-oriented Work-flow Concepts in Inter-organizational Environments. In *Proc. of the 4th. Intnl. Conference on Information and Knowledge Management*, Baltimore, MD.

[Technologies, 1995] Technologies, A. (1995). Metro Tour. Technical report, Action Technologies, Inc. URL: http://www.actiontech.com/.

[Vivier et al., 1996] Vivier, B., Haimowitz, I., and Luciano, J. (1996). Workflow Requirements for Electronic Commerce in a Distributed Health Care Enterprise. In *[Sheth, 1996]*, Athens, GA.

[Wachter and Reuter, 1992] Wachter, H. and Reuter, A. (1992). The ConTract model. In *[Elmagarmid, 1992]*, chapter 7. Morgan Kaufman.

[Wang, 1995] Wang, X. (1995). Implementation and Performance Evaluation of CORBA-Based Centralized Workflow Schedulers. Master's thesis, University of Georgia.

[Weikum, 1993] Weikum, G. (1993). Extending Transaction Management to Capture More Consistency With Better Performance. In *Proc. of the 9th. French Database Conference*, pages 27–30, Toulouse.

[Weikum and Schek, 1992] Weikum, G. and Schek, H. (1992). Concepts and applications of multilevel transactions and open-nested transactions. In *[Elmagarmid, 1992]*, chapter 13. Morgan Kaufmann.

[Wodtke et al., 1996] Wodtke, D., Weissenfels, J., Weikum, G., and Dittrich, A. K. (1996). The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In *Proc. of 12th. IEEE Intl. Conference on Data Engineering*, pages 556–565, New Orleans, LA.

[Worah, 1997] Worah, D. (1997). Error Handling and Recovery in the METEOR$_2$ Workflow Management System. Master's thesis, University of Georgia, Athens, GA. In preparation. URL: http://LSDIS.cs.uga.edu/.

[Worah and Sheth, 1996] Worah, D. and Sheth, A. (1996). What do Advanced Transaction Models Have to Offer for Workflows? In *Proc. of Intl. Workshop on Advanced Transaction Models and Architectures*, Goa, India.

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

Index

Appendix: A Normative Perspective

	Advanced Transaction Models	Workflow Systems
Theoretical Foundation	Usually good theoretical basis.	Weak dependency, except for scheduling components. Driven by practical considerations.
Granularity	Transactions.	Tasks, activities, or steps
Methodology	Data-centric. Emphasis on data consistency.	Process-centric. Emphasis on task coordination.
Correctness Criteria	Serializability (if possible).	Primitive, often limited to scheduling.
Failure Atomicity	Inherent.	Not part of most models.
Concurrency Control	Inherent.	Limited support.
Recovery	Well-defined. *Rollback* and *compensation.*	Insufficient support. *Forward recovery* when supported.
Error Handling	Limited.	Very limited.
Task/Activities	Supports transactions only.	Supports both human and application tasks.
Processing Entities	Usually DBMS.	Heterogeneous systems (e.g., DBMSs, TP monitors, legacy applications, humans)
Coordination Support	Limited.	Inherent.
Modeling Organizational Structure	Usually absent.	Varies significantly.
Worklists	No support.	Strong support.
Flexibility	Varied.	Good.
Implementation Status	Very few exist.	Numerous commercial products and few research prototypes.
Applicability to Non-DBMS applications	Very limited.	Extensive.

Dr. Amit Sheth directs the Large Scale Distributed Information Systems Lab (LSDIS, http://lsdis.cs.uga.edu) and is an Associate Professor of Computer Science at the University of Georgia. Earlier he worked for nine years in the R&D labs at Bellcore, Unisys, and Honeywell. His research interests include multiparadigm transactional workflow (project METEOR), management of heterogeneous digital data and semantic issues in global information systems (projects InfoHarness & InfoQuilt), and interoperable information systems involving

intelligent integration of collaboration, collaboration and information management technologies. Prof. Sheth has approximately 100 publications to his credit, given over 50 invited and colloquia talks and 15 tutorials and professional courses, and lead for international conferences and a workshop as a General/Program (Co-)Chair in the area of information system cooperation/interoperability, workflow management, and parallel and distributed information systems, has served on over thirty five program and organization committees, is on the editorial board of five journals, and has served twice as an ACM Lecturer.

Devashish Worah is a senior component-software engineer in the Professional Services group at I-Kinetics, Inc., Burlington, MA. He is involved with the analysis, design, and implementation of large-scale, CORBA and WWW-based distributed information systems, and in defining a methodology for transitioning legacy systems to object-based distributed environments. Prior to working at I-Kinetics, Mr. Worah was a research assistant at the Large Scale Distributed Information Systems Lab., at the University of Georgia (UGA), Athens, GA. He was a member of the METEOR2 Workflow Project team, and was instrumental in incorporating reliability features into the ORBWork WFMS. Mr. Worah designed and developed a state-wide immunization tracking workflow application prototype based on CORBA and WWW at the Connecticut Healthcare Research and Education Foundation, Wallingford, CT. He completed his B.S. in Computer Science and Mathematics at Georgia College and State University, Milledgeville, GA. He is working towards completing his Master's dissertation at UGA under the guidance of Prof. Amit Sheth. His current research interests include workflow technology, distributed object computing, reliability in distributed systems, and legacy integration.