

ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR₂

S. Das, K. Kochut, J. Miller, A. Sheth, D. Worah
Large Scale Distributed Information Systems Lab (<http://LSDIS.cs.uga.edu>)
The University of Georgia, Athens, GA 30602-7404
email: {*souvik,kochut,jam,amit,worah*}@cs.uga.edu

Abstract

Key limitations of the state-of-art workflow products and research prototypes include the lack of adequate support for functioning in heterogeneous environments that involve humans and automated tasks distributed across enterprises, limited scalability, and the lack of adequate support for dealing with errors and failures in real-world organizational settings. Emergence of network computing based on Web and distributed object management provide an attractive infrastructure to address these issues. Workflow management techniques developed in the METEOR₂ project are intended to reliably support coordination of user and automated tasks in real-world multi-enterprise heterogeneous computing environments. Key capabilities of the METEOR₂ workflow management system (WFMS) include a comprehensive toolkit for building workflows (map/data/task design) and supporting high-level process modeling, detailed workflow specification, and automatic code generation for its workflow enactment systems - WEBWork and ORBWork. In this paper, we discuss the design and implementation of ORBWork, reliable and fully distributed a CORBA-based enactment system for the METEOR₂ WFMS. In addition to providing coordination capabilities in a heterogeneous and distributed environment, ORBWork supports scalable software architecture, multi-database access, as well as an error detection and recovery framework that uses transactional concepts. The recovery framework is based on a hierarchical error model, and includes mechanisms for persistence, monitoring and recovery of workflow system components.

1 Introduction

A *workflow* is an activity involving the coordinated execution of multiple *tasks* performed by different processing entities [KS95]. A *workflow process* is an automated organizational process involving both human (manual) and automated tasks. *Workflow management* is the automated coordination, control and communication of work as is required to satisfy workflow processes [SGJ⁺96]. A *Workflow Management System* (WFMS) is a set of tools providing support for process definition, workflow enactment, and administration and monitoring of workflow processes [Hol94]. The *process definition* tool is a design-time component that is used to map an organizational process into a workflow process. The workflow process is based on a formalized workflow model that is used to capture data and control-flow between workflow tasks. The *workflow enactment* system (a subset of which is also referred to as a workflow engine) consists of run-time components that provide the execution environment for the workflow process. It is responsible for enforcing inter-task dependencies, task scheduling, workflow data management, and for ensuring a reliable execution environment. *Administrative and monitoring* tools are used for management of user and workgroup roles, defining policies (e.g., security, authentication), audit management, process monitoring, tracking, and reporting of data generated during workflow enactment.

WFMSs today, are being used to re-engineer, streamline, automate, and track organizational processes [JAD⁺94, GHS95, Fis95, SKM⁺96, BSR96]. There has been a growing acceptance of

workflow technology in numerous application domains such as telecommunications, software engineering, manufacturing, production, finance and banking, laboratory sciences, health care, shipping and office automation. The current state-of-the-art in WFMSs is dictated by the commercial market [AS96]; several hundred products that provide support for workflow management, in some form, exist in the market today [Sil95, Fis95, GHS95, SJ96, AAAM97]. These systems are primarily focused toward providing automation within the office environment with emphasis on coordinating human activities, and facilitating document routing, imaging, and reporting.

In the last few years, pervasive network connectivity, catalyzed by the explosive growth of the Internet has changed our computational landscape. Centralized, homogeneous, and desktop-oriented technologies have given way to more distributed, heterogeneous and network-centric ones. This has raised challenging requirements for workflow technology in terms of being required to support large scale multi-system applications, involving both humans and legacy systems, in heterogeneous, autonomous and distributed (HAD) environments. Some of the apparent weaknesses that need to be addressed by the workflow community include limited support for heterogeneous and distributed computing infrastructures, lack of a clear theoretical basis, undefined correctness criteria, limited support for synchronization of concurrent workflows, lack of interoperability, minimal scalability and availability, and lack of reliability in the presence of failures and exceptions [BDSS93, JNRS93, GHS95, AS96, KR96, LSV96, WS96, WS97, AAAM97].

Large scale workflow systems require integration of heterogeneous applications that execute on diverse systems; in addition, they need to be able to support data access and communication across distributed organizational settings independent of system and application-level heterogeneity. Emerging and maturing infrastructure technologies for communication and distributed object management (e.g., CORBA/IOP, DCOM/ActiveX, Java, Notes, and Web) have addressed many of these challenges, and are making it feasible to develop standards-based large scale, distributed application systems.

There has been considerable research in the synergy between workflow and transaction-based systems. Transaction processing [GR93], database management, and many (but not all) efforts related to Advanced Transaction Models (ATMs) [CR91, Elm92, GHKM94], 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. These are very useful concepts that could be applicable in workflow systems. The degree to which transaction-based concepts map to workflow systems has been an issue of active debate. 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 [GH94, GHKM94, CD96, BDG⁺94, Wei93, WR92]. However, it has also been observed [BDSS93, AAA⁺96, WS96, WS97] that ATMs have limited applicability in the context of workflows due to their inability to adequately model the rich requirements of today's organizational processes.

The METEOR₂ workflow project at the Large Scale Distributed Information Systems Lab., University of Georgia (LSDIS-UGA) is a continuation of the METEOR [KS95] effort at Bellcore. Research and development work is geared towards developing a multi-paradigm transactional WFMS capable of supporting large scale, mission critical, inter-enterprise workflow applications in HAD environments. Various task models have been defined to support integration of heterogeneous task types into the workflow model [KS95, Wan95, Mur95, MSKW96]. Task models include models for transactional, non-transactional, user and two-phase commit types of tasks. Graphical workflow designers have been developed to enable graphical representation of the workflow process, associated workflow data, and task models that map to actual user and application tasks. The design process is followed by semi-automatic code generation of an executable workflow application [KS95, MSKW96].

Several enactment systems have been designed and implemented based on various scheduling paradigms [Wan95, MSKW96, SKM⁺96]. These range from highly centralized ones to fully

distributed implementations using CORBA and Web technologies (either exclusively, or in combination) as infrastructure for workflow enactment. Two distributed workflow enactment systems (WEBWork and ORBWork) have been successfully used to support a comprehensive prototype of the state-wide immunization tracking workflow process involving multiple hospitals, and health-care providers [SKM⁺96]. In this paper, we discuss the design and implementation of ORBWork, a reliable distributed CORBA-based enactment system for the METEOR₂ WFMS. In this implementation, CORBA provides the necessary distribution and communication capabilities for the workflow components, and the Web makes it possible for humans to interact with the Object Request Broker (ORB) enabled enactment system. Error handling and failure recovery are inherent features of the enactment system. This is based on a well-defined error model and run-time support for detecting and recovering from logical and system errors.

The organization of this paper is as follows. In section 2 we present an overview of the state-of-art in WFMS with emphasis on workflow enactment system architectures, infrastructure technologies used to implement them, and their support for error handling and recovery. Section 3 discusses the METEOR₂ WFMS in terms of a phased approach used from workflow design to run-time enactment of an organizational process. The architecture of the ORBWork is discussed in section 4; this section also outlines the run-time model in terms of integration of tasks of various types, and the coordination of data and control flow between tasks during workflow enactment. We define our error model and mechanics for error detection and recovery in section 5. We also discuss the support for persistence, monitoring, and recovery of workflow system components in ORBWork. Finally, in section 6 we close this paper with our conclusions and a brief discussion of on-going work.

2 Relevant Work

Several hundred commercial products that provide some degree of workflow automation exist in the market today [Sil95]. Most of these solutions are information management tools that support relatively simple organizational processes in intranet environments (e.g., office automation, document-flow, and imaging). The research community, on the other hand, has been more aggressive in its agenda for workflow research [SGJ⁺96]. Various aspects of workflow processing have been addressed; some of these areas include defining *rich* workflow models and specification languages that can capture real-world organizational processes, investigating the role of ATMs in the context of workflow systems, assessing various workflow scheduling mechanisms, defining different workflow enactment system architectures, assessing the viability of different communication infrastructures for workflow enactment, and defining requirements and techniques for incorporating concurrency control and reliability semantics into large-scale WFMSs [Hsu93, Hsu95, GHS95, She95, Dog96, SJ96].

In spite of the standardization efforts of the Workflow Management Coalition (WfMC) [Hol94], workflow products and research prototypes differ widely in terms of their approach towards organizational process modeling, workflow enactment system architecture, run-time environment, and system-level functionality. In this section, we will limit our discussion to relevant work in the areas of workflow enactment system architecture, WFMS infrastructure, and support for error handling and recovery in WFMSs.

2.1 Workflow Enactment System Architecture

Most commercial workflow products today are based on client-server architectures that use a centralized workflow enactment system to support scheduling and workflow-level functionality. Information pertaining to enforcing inter-task dependencies, internal WFMS state, and the organizational structure (i.e., roles, policies, etc.) is usually stored in a centralized information repository - in most cases, a DBMS (e.g., WorkFlow from FileNet uses Oracle, Metro from Action Technolo-

gies uses Microsoft's SQL Server). The centralized architecture and scheduling mechanism eases administration and management of these products. However, these solutions are inappropriate for automation of inherently distributed inter-enterprise workflow processes. Moreover, the central components are subject to being performance bottlenecks and central points of failure, thereby limiting the scalability, availability, and reliability of these products. The workflow enactment system (or engine) architecture proposed by the WfMC [Hol94] itself suffers from this limitation as it is modeled as a centralized facility for the coordination of workflow processes.

Numerous architectures for workflow enactment systems have been proposed in research literature. The Mentor project [WWWD96] is aimed at developing a distributed middleware framework for workflow enactment. It is based on a client-server architecture that allows co-existence of multiple workflow servers that contain their own workflow engines. An object-based client-server WFMS architecture is used in the Mobile project [SJKB94]. It uses a layered messaging architecture that abstracts the operating system and communication level heterogeneity of the enactment environment. A client-server database architecture in a CORBA-based environment is used in the WIDE project [CGS97]. The scheduling mechanism in WIDE is based on a centralized rule execution module. The Exotica project extended the FlowMark WFMS to support a distributed client-server architecture based on a queue messaging mechanism [AAA⁺95], mobile clients with intermittent connections [AKA⁺95] and Lotus Notes [RM96a].

The INformation CArrier (INCA) workflow model [BMR96] was developed to support dynamic workflow processes that involve distributed processing entities that are relatively autonomous in nature. In this architecture, rules for workflow scheduling and internal data are encapsulated within a distributed object called an INCA. At run-time, the INCA is routed across various distributed processing entities based on the workflow process definition. One of the drawbacks of using this approach in a large-scale workflow application is its heavy dependence on mobile objects for encapsulating *all* workflow scheduling and task related data. This could result in gradual degradation of performance in complex workflow scenarios. The ORBWork enactment system discussed in this paper overcomes this problem by adopting a more scaleable architecture.

Five run-time architectures for workflow enactment systems have been defined in the METEOR₂ project [Wan95, MSKW96]. These range from a highly centralized model to a fully distributed one. The highly centralized enactment system contains a scheduler and task managers within one heavyweight process; in the fully distributed model, the scheduling is both logically, as well as physically separated thereby providing a scalable, and fault-tolerant architecture (for details, see Section 4.7).

2.2 Workflow Infrastructure

In the past, database technology has been used to support process management (using triggers and stored procedures) in relatively homogeneous workflow environments [DHL90, BSR96]. The need for distributed object-based infrastructure technology for development of large-scale enterprise-wide WFMS was voiced in [GHS95]. Today, there exists a variety of middleware technologies and standards to support distributed computing such as TP-monitors, DCE, OpenDoc, CORBA, DCOM, Notes, Web, and Java RMI.

Commercial workflow products are geared towards providing desktop-oriented solutions in relatively homogeneous environments (primarily Windows, WindowsNT and Unix). Many of these WFMSs rely on email and DBMS for communication between workflow components and users [GHS95]; others rely on desktop technologies (e.g., OLE) and system-specific functionality (e.g., TCP/IP sockets). The popularity of the Internet has resulted in an increase in the number of Web-based (i.e., HTML, HTTP, Java) workflow solutions from most vendors. However, use of the Web has been limited to its basic client-server form in providing a interface, serving documents, and supporting restricted programmatic capabilities using CGI scripting.

Distributed technology has been used in a number of research prototypes. A workflow implementation that uses a Distributed Object Management System to allow integration of HAD applications is discussed in [GHS95]. The Mentor WFMS uses CORBA as a standard mechanism for communication between the workflow enactment system and invoked applications; a TP-monitor is used to enable fault-tolerant communication between workflow servers and the workflow logs [WWWD96]. The integration of Lotus Notes with FlowMark has been researched to develop flexible workflow solutions [RM96b]. The METEOR₂ project uses Web and CORBA as primary infrastructure technologies to support for workflow enactment [Wan95, MSKW96, SKM⁺96]. WEBWork [Pal97] and ORBWork [SKM⁺96, Das97] are Web and CORBA-based distributed enactment systems for the METEOR₂ WFMS. The Object Management Group (OMG) is making an effort to standardize interfaces for a Workflow Facility that would provide a model for distributed workflow management across heterogeneous platforms and operating systems.

Research in WFMSs, so far, has been primarily focused towards modeling and coordination of workflow processes. Integration of heterogeneous tasks (applications) in a generalized manner has not been a major concern since most this has been achieved on a case-by-case basis. A few efforts have been dedicated towards defining formal models for integration of heterogeneous tasks types (e.g., transactional, non-transactional) into the WFMS [ANRS92, RS95, KS95, MSKW96]. Runtime support for heterogeneous tasks is achieved in the METEOR₂ WFMS [MSKW96, SKM⁺96] using task managers. Another approach that claims supports integration of heterogeneous tasks uses task-specific wrappers to enable interaction with the workflow layer [SJHB96]. Commercial WFMSs provide support for only a restricted subset of task types. Their level of integration support is restricted to applications that are compliant to specific low level protocols (e.g., DDE, OLE, IPC); for example, WorkFlo from FileNet provides application specific adaptors that would enable them to be integrated into the WFMS layer.

2.3 Workflow Reliability

Workflow reliability is a critical area of workflow research that is lacking practical solutions [GHS95, AS96, KR96, SGJ⁺96, WS97, AAAM97]. Valuable research addressing recovery has been done in database theory, transaction management, and ATMs [BHG87, KLS90, CR91, Elm92, GR93]. They have proposed solutions (although many of which have yet to be implemented) when the constituent tasks are transactional, or the processing entities provide a transactional interface. Error handling in database systems has typically been achieved by aborting transactions that result in an error. However, cancelling a workflow task would not always be appropriate or necessary in the event of failures. Transactional RPC mechanisms and TP-monitors [GR93] have been used in distributed transaction processing to guarantee reliable messaging between distributed processes. They have been used in workflow systems [WWWD96] to guarantee transactional messaging between the workflow components thereby increasing the level of fault-tolerance of the WFMS infrastructure.

ConTracts [WR92] were proposed as a mechanism for grouping transactions into a multi-transaction activity. The *steps* within a ConTract are forward recoverable. ConTracts provide relaxed isolation and atomicity so that it may be interrupted and re-instantiated in the event of errors or failures. A mechanism for dealing with errors in an ATM for *long running activities* was proposed in [DHL90, DHL91]. It supported forward error recovery, so that errors occurring in non-fatal transactions could be overcome by executing alternative transactions. In [Wei93], the author suggests that semantic transaction concepts be merged with workflow concepts to promote workflow systems that are consistent and reliable. The *Transaction Specification and Management Environment* (TSME) [GHKM94] is an advanced transaction framework for execution of workflows. In TSME, workflows are a collection of transactions corresponding to tasks. The TSME claims to support various ATMs to ensure correctness and reliability of various types of workflow processes. In the work on *nested process management systems* [CD96], the authors present a formal model

of recovery that utilizes backward recovery and 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 [Mos82], it is limiting for heterogeneous workflows that involve tasks that are non-transactional in nature.

The *METEOR* transactional workflow model [KS95] discusses error handling and recovery of workflows. Failures are modeled at the task level, are mapped to the implementation level using Task and Workflow Specification Languages (TSL and WFSL respectively). Forward recovery is achieved using logging, alternate tasks and compensation. Compensation has also been applied to tasks and groups of tasks (*spheres*) in the FlowMark WFMS to support partial backward recovery [Ley95]. A stated objective of the Exotica project is to develop workflow systems that are reliable, scalable, and available enough to deal with large, distributed, legacy application environments. The *Exotica* project [AAA⁺96] explores the role of advanced transaction management concepts in the context of workflows. They have investigated the effect of using recoverable message queues [AAA⁺95] for communication within a decentralized workflow system. In the *INCA* WFMS [BMR96], transactional semantics of procedures (or steps) are limited by the transaction support guaranteed by the underlying processing entity. 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. However, the INCA itself is neither atomic nor isolated in the traditional sense of the terms. The *Workflow Activity Model* (WAMO) discusses workflow recovery [EL96]. 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. The Workflow Reference Model proposed by the WfMC does not address workflow reliability issues such as error handling and recovery.

The study of WFMSs is inter-disciplinary, and stems from areas such as database management, transaction processing, distributed systems, software engineering, and organizational sciences [SGJ⁺96]. Error handling and recovery are equally critical in these domains, and numerous solutions have been suggested to address these problems [BHG87, Cri91, Saa95]. Recovery in object-based systems have been researched in the *Arjuna* [SDP91] project. The *Isis* toolkit [BR94] provides a framework for fault-tolerant distributed computing. Detecting failures in CORBA-based distributed systems is discussed in [Maf96]. The CORBA standard [OMG95b] includes specifications for Object Services [OMG95a] 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 HAD settings. In the ORBWork WFMS, we assimilate many of these concepts to define a unified error handling and recovery framework.

3 *METEOR*₂ Architecture

In this section we discuss the overall architecture of the *METEOR*₂ WFMS including how the different modules interact with each other. Currently completed modules include a designer, two workflow enactment systems - Web-based WEBWork and CORBA-based ORBWork, including the respective runtime code generators.

A good workflow system is ultimately used to increase the productivity and quality of products of an organization, and these aspects of organizational success critically depend on human participation and their acceptance of any computerized system they have to deal with. This fact has lead us to keep the ease-of-use of all participants including workflow application developers and end users in mind when designing all aspects of *METEOR*₂.

3.1 The Designer

The METEOR₂ WFMS has a designer that is a graphical user interface used to specify the entire map of the workflow, data objects manipulated by the workflow, as well as the detail of task invocation. The workflow designer has the capability to model complex and varied tasks in a very high-level conceptual and easy to use manner that shields the designer of the workflow from the underlying details of infrastructure or the runtime environment. It also aims at providing the user very few restrictions regarding the specification of the workflow. The designer assumes no particular implementation of the workflow runtime. It is a completely separate entity from the runtime. This has been done to separate the specification aspects from the runtime. The specified design can be stored in a pre-determined intermediate format for subsequent code generation.

3.1.1 Workflow Design

The METEOR₂ designer has two design modes. The first mode, called *Process Modeler*, is aimed as a tool for the management of a typical enterprise. A workflow design may be initiated at the high, organizational level, without devoting any thought to the implementation details. This designer mode focuses on high level specification issues without going into the procedural issues of the runtime. The *Process Modeler* stresses the *what* of the process rather than the *how*. The second mode, called *Workflow Builder*, is closer to the implementation aspects of creating a workflow application. This mode would be used by technical engineers or system analysts who would know the details of the underlying run time system. The design specification created with the *Process Modeler* would be refined in a more fine grained manner (using the *Workflow Builder* in order to create a complete workflow application. It is in this mode that we specify the entire map of the workflow including various tasks, task managers, and their interactions. We shall talk about the second mode of specification next.

As has been mentioned earlier, the tool has a graphical user interface. The designer has three components: the map designer, the data designer, and the task designer. These components can model:

- the workflow map expressing the ordering of tasks and the dependencies among them,
- the data objects manipulated and transmitted by the tasks, and
- the details of the individual tasks (workflow activities) and interface to external task development tools (e.g., Microsoft's FrontPage to design the interface of a user task, or a rapid application development tool).

3.1.2 The Map Designer

The map designer specifies the map of the designed workflow. It is assumed that the creator of the map is intimately familiar with the workflow application. If that is not the case, then the *Process Modeler* can be used by the creator to specify the process overview. This process overview must be then fine tuned for purpose of creating the complete workflow application.

The tasks in the workflow map are specified as transactional, non-transactional, composite or compound. Each task type has a different iconic representation in the designer tool.

The designer uses a graph to represent the workflow map. Nodes are used to represent tasks and arcs to represent the transitions between the tasks. It is assumed that the runtime scheduler will execute the workflow instance according to this graph.

A workflow task may have an arbitrary number of predecessor tasks. The workflow designer may introduce an AND-OR condition on the predecessor tasks, so that all of the predecessor tasks

in one of the AND-groups must complete before the task may start execution. The AND-OR condition is called an activation dependency.

A workflow task may have one or more outgoing transitions (the final task of the workflow has none). Each outgoing transition may have an associated data dependency. The transition is enabled only if that particular dependency is satisfied. The dependency can be completely arbitrary and may involve user inputs or attribute values in data objects.

The map designer has the capability to specify alternate routes that a workflow instance may take due to a failure of a particular task. For example, while designing a task the number of retries can be specified. If the task fails to complete after several retries then an alternate or a compensating route may be specified from the FAIL or ABORT state of the task. So, the abstraction of an alternating or a compensating task has been lifted up to the WFMS level.

3.1.3 Data Designer

The data designer is based on the Object Modeling Technique (OMT), however, it does not support the OMT model entirely. The designer of a workflow application can create a number of classes, encapsulating both the data representation (attributes) and operations on the data (methods) for various data elements manipulated by the tasks in the workflow. Similar classes can be grouped together into inheritance hierarchies. As in OMT, various relationships (binary, ternary, one-to-one, one-to-many, etc.) may be defined on groups of existing classes to reflect data type dependencies.

3.1.4 Task Designer

In a heterogeneous work environment typically various tasks have to be performed which differ vastly in their execution mechanisms. The METEOR₂ task designer can be used to specify invocation details of different task types. The task designer is capable of modeling typical invocation details such as the name of the existing executable and the required input parameters.

As we already mentioned, a workflow task may have a number of transitions leading to it from its predecessor tasks. Any of these transitions may have associated data objects which are being transferred to the task when the transition is taken.

Once the executable of the task is specified, a command line string may be created. The command line may include a collection of attributes from the available data objects. In case the task program requires its inputs to be provided by data files, the task designer can be used to provide a template of such data files. Such templates may include data object attributes and other text. The templates are instantiated by the workflow runtime once a specific workflow instance reaches the task.

Many tasks in a workflow may be user tasks. Since the communication with the user of a workflow application is done with the use of a Web browser, the task designer is capable of initiating the construction of an HTML form which will be used to transfer the data to and from the Web browser. Such a form is initially equipped with text input fields corresponding to attributes defined in the available data objects. An HTML authoring tool of choice (such as Microsoft's FrontPage) can then be used to finalize the design of the form.

3.2 The Intermediate Language

The workflow specification created in the designer is stored in an intermediate format called the *Workflow Intermediate Language (WIL)*. The WIL format is similar in structure and semantics to the *Workflow Process Definition Language (WPDL)* of the Workflow Management Coalition and supports most aspects of our earlier intermediate languages- Workflow Specification Language (WFSL) and Task Specification Language (TSL) [KS95]. A WIL specification includes all the predecessor-successor dependencies between the tasks as well as the data objects that are passed

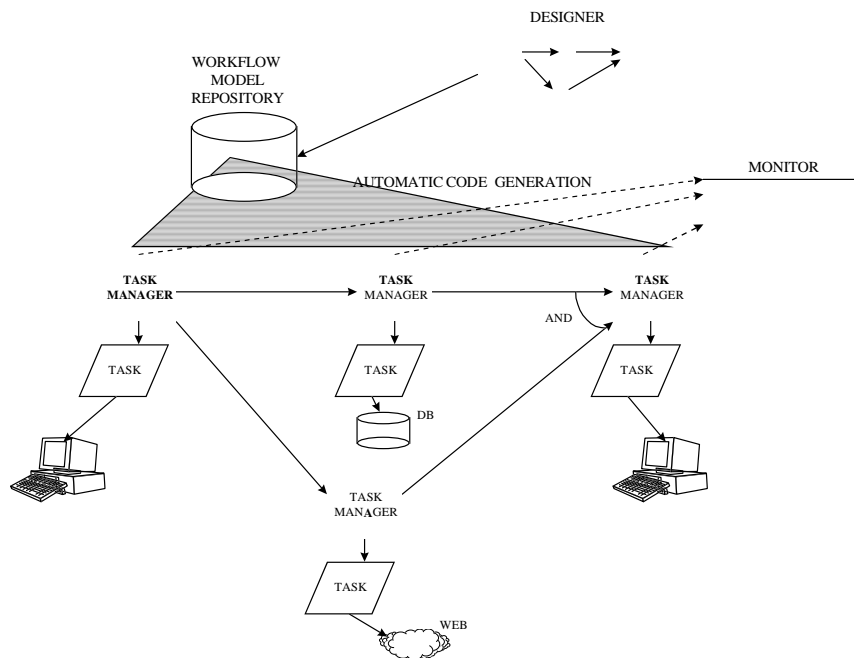


Figure 1: The METEOR₂ Architecture

among the different tasks. It also includes definitions of the data objects, and task invocation details. For more details of the WIL intermediate language see [Lin97].

3.3 The Runtime Code Generator

The functionality of the METEOR₂ workflow designer has been tailored so that it allows for almost complete automatic code generation for a complete workflow application. Of course, it is not possible to generate code for computer (automatic) tasks, since the individual tasks must be provided later by task developers (or possibly already exist as legacy applications). Each runtime (ORBWork and WEBWork) has a suitable code generator.

The code generator is a module that takes the WIL as input and outputs the run time code of a workflow application. The code generator outputs code for task managers, including their scheduling components, task invocation code, data object access routines, and the recovery mechanism. This mechanism allows for relatively easy way to incorporate legacy applications.

The code generator also outputs the code necessary to maintain and manipulate data objects, created by the data designer.

3.4 The Runtime System

The runtime system consists of the various task managers and their associated tasks, the user interfaces, the distributed recovery mechanism, the scheduler (distributed among task managers) and the various monitoring components. Figure 1 shows the various modules in METEOR₂ and their interactions. As can be seen in the picture, the designer is used to create a workflow specification. The specification is stored in a workflow model repository. The workflow code generator reads in this specification from the repository and generates the workflow application.

The goal of METEOR₂ was to keep the involvement of the end user to a bare minimum. The user interface hence has been kept very simple and intuitive. Due to the omnipresence of the World Wide Web we decided to have a Web browser as our user interface. All the run time modules of ORBWork will be discussed in the next two sections.

4 The ORBWork Workflow Enactment System

As has been mentioned earlier, there is a host of different workflow management systems available in the commercial and the research arena. These systems vary widely from their design philosophy and methodology and hence in their implementation strategies. The METEOR₂ model does not assume any particular architecture of the workflow enactment system. We have, however, opted to have a fully distributed architecture of the runtime system. We understand the additional overhead and problems a distributed implementation entails. But we strongly believe that a workflow system by nature should reflect a natural work environment. We evaluated some business organizations and their methods of doing work. We found that a lot of distribution and heterogeneity is inherent in those methods and the underlying process of work. Typically, in a modern work environment a lot of ad-hoc activities co-exist with several distributed design centers - albeit with some sort of a hierarchy that models the organizational structure. Work is done over different heterogeneous resources. To model such scenarios and applications correctly, a workflow system should also be built over a distributed environment that is also capable of handling heterogeneity in a seamless fashion. Right from the start, our philosophy has been to build a system where each and every module has sufficient independence to perform its job efficiently but, at the same time, is also responsible to some evaluation entities in the organization (which in our case would be a workflow monitor). Another significant driving force in favor of the distributed architecture is the absence of a single point of failure. This has a profound impact on error handling and recovery issues that will be discussed in detail in the next section.

4.1 Infrastructure Elements

ORBWork relies on CORBA to supply our basic communication infrastructure. CPRBA is also used in a more comprehensive manner to support interoperability and legacy application or data source wrapping. All our task managers, tasks (or wrappers around existing legacy tasks), the monitoring entity, and the recovery mechanism are CORBA objects. This was almost the natural choice once we decided on a distributed framework, since CORBA provides an infrastructure for facilitating development of reusable, portable and object-based software in a distributed and heterogeneous environment. In our prototype implementation, we have used VisiBroker for C++ (formerly ORBeline) from Visigenic, Inc. and Orbix from Iona Technologies, both CORBA 2.0 compliant ORB implementations.

A web browser is already the preferred user interface tool, and has lead to “webification” of enterprise applications. We also provide a browser-based interface for all end users to provide humans to participate seamlessly in workflows regardless of their location and (client) system platforms.

4.2 The Runtime System

The runtime system of the METEOR₂ system is divided into two parts corresponding to two types of components: Task Manager (controller/scheduler) and task (executable). The first part consisting of the implementations of the ORBWork’s Task Managers is automatically generated from the designer’s intermediate form, stored as WIL. This includes information necessary for task scheduling and data transfers (and more). The second part is the code implementing application specific tasks, which may be a legacy application code wrapped with the support of task designer component of our workflow designer, developed by a third-party application development tool which may be interfaced with the designer, or hand coded (e.g., a script written in a text editor).

The task manager design is highly object-oriented and consists of four components: task activation, task invocation and observer, error handling, and recovery (see Figure 2).

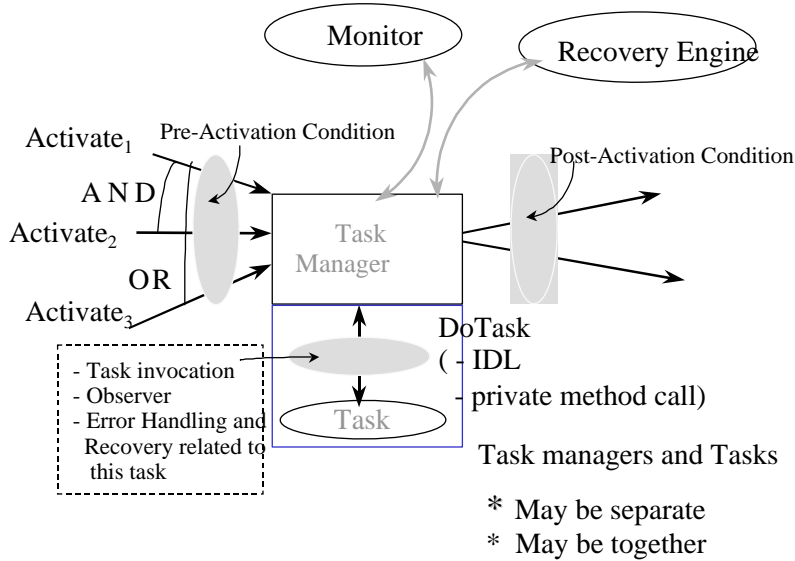


Figure 2: An abstraction of a task manager

The remainder of this section is primarily related to the first two modules of the task managers for supporting the distributed WFMS architecture. Failure handling and recovery are discussed in the next section.

4.3 The Hierarchy of Task Managers

METEOR's workflow model distinguishes among the different types of tasks depending upon their behavior. This has been done to ease modeling the semantics of the task. Correspondingly, ORB-Work supports different types of task managers. Depending on the task semantics (especially transactional or not), a task manager supports different functionality, in particular to deal with failures and corresponding recovery.

Each task manager is implemented as a CORBA object and exports certain public methods as an external interface. The architecture has a base task manager class which implements the basic functionality of a typical task manager. Other task managers derive from this base task manager class and implement their specific requirements.

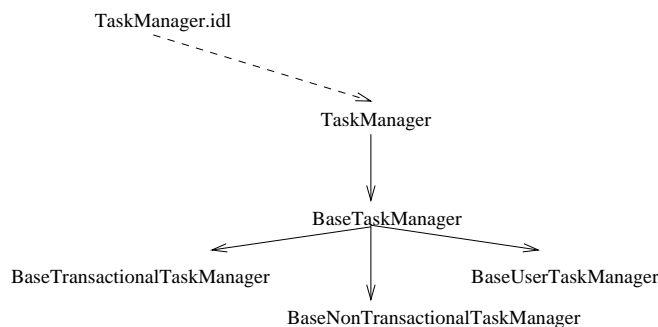


Figure 3: The Task Manager Hierarchy

As shown in Figure 3, the TaskManager IDL definition specifies the interface of the root class of the hierarchy of all task manager classes. When compiled by an IDL compiler, this definition generates the **TaskManager** class. The class **BaseTaskManager** is derived from TaskManager. BaseTaskManager includes a variety of attributes and methods needed in all task managers. Three other classes are derived from BaseTaskManager: **BaseTransactionalTaskManager**, **BaseNon-**

TransactionalTaskManager, and **BaseUserTaskManager**. All task managers used in a workflow runtime system are derived from one of these three classes. As the names suggest, task managers derived from **BaseTransactionalTaskManager**, **BaseNonTransactionalTaskManager**, and **BaseUserTaskManager** are used to control transactional, non-transactional, and user tasks, respectively. **BaseUserTaskManager** is intended to add properties that would enable it to handle specific functions needed in controlling user tasks. This task manager class hierarchy renders a high degree of modularity to the ORBWork's software architecture and implementation.

4.4 Task Activation and Distributed Scheduler

METEOR₂ does not have a single entity responsible for scheduling activation of task managers. Instead the scheduling information has been distributed among the task managers. The control dependencies are defined in the workflow designer and are rendered in the intermediate representation format (WIL). The workflow code generator reads the WIL specification and creates the task manager routines. This automatically generated code includes sections dealing with the scheduling logic. So, each task manager "is aware of" its immediate successors and hence is capable of activating the follow-up task managers once the task it controls terminates.

Each task manager is started (usually by its predecessor(s)) by a method called **Activate**. This is the only method call that carries the control (scheduling) information across the workflow instance. The **Activate** method carries necessary parameters that are sufficient for the successor task manager to proceed with its own work. One of the parameters is a list of names of data objects that the successor task manager would need to access for invoking its associated task. The exact timeline of events that happen once the task manager receives an **Activate** call will be discussed later.

Each task manager is equipped with a fragment of per task manager scheduling code which is subdivided into pre-activation, task activation, and post-activation parts. The pre-activation part must determine if and when the associated task should start execution, i.e. evaluate the task's activation dependency. A task's activation dependency may involve a number of predecessor tasks (with the control part either AND-ed or OR-ed). The activation dependency is expressed in the form of an AND-OR tree and may be normalized to a disjunction of conjunctions. In ORBWork, activation dependencies are compiled directly into fragments of C++ code and included with the task manager code, essentially acting as part of the scheduler of the overall workflow.

The task activation part handles the execution of the associated task. After preparing the task inputs, which might involve "unpacking" of the input data objects, and verifying that the task program may be executed by the specified processing entity (for example, if the specified host is up and running, and if the given executable exists), the task itself is started. After its completion, the final state of the workflow task is determined, and (possibly depending on the produced results) the output data objects are prepared.

In the post activation part, which corresponds to the workflow task being in one of its terminating states, the task manager's scheduling part attempts to initiate one or more successor tasks.

The task manager is responsible for saving the state of the data objects it uses. This is accomplished by calling the **Save** method provided by the data object, or by using the persistent object services of CORBA. This way, the task manager ensures that the WFMS has a saved image of the data objects before they are accessed by the individual task. The task manager also saves the state of the data objects after successful completion of the task. This way the system logs both a pre- and a post-image of each of the data objects accessed by the task.

To run the associated task, the task manager calls a method called **DoTask**. The task manager and the task can be either in two different processes or they can be in the same process. If the task and the task manager are different processes the **DoTask** method called by the task manager

would have to be a CORBA method invocation on a separate task object. This task object may be actually a CORBA object which performs the task or may be simply a wrapper for a legacy application task. Separation of task and task managers is better if viewed from recovery perspective because a failure of the task allows the task manager to perform a recovery step. Having tasks implemented as objects is very useful when legacy tasks need to be incorporated into the WFMS. The task objects can be implemented as wrappers around the legacy tasks and can be responsible for transferring data into the legacy application from the WFMS and out to the WFMS from the application. The details of the legacy application are then hidden from the WFMS.

After the task is successfully completed, the task manager calls the **Activate** method on the next task manager(s) (as specified in the design of the workflow). Each outgoing arc of a task manager (see Figure 2) may have a data dependency which has to be satisfied for the task manager to **Activate** the associated transition.

With the above in perspective, we can now lay down the sequence of events that take place once a task manager receives an **Activate** call from its predecessor. While processing the received **Activate** call, the task manager:

- Stores the identity of the caller and all the data object references it receives from it;
- Executes a function (atomically) to evaluate the AND-OR boolean condition. If the condition is true, it calls the **DoTask** method to execute the task. Otherwise, it just exits;
- If the boolean condition is successful, it saves each of the data objects that it is going to access;
- The task manager then makes a **DoTask** call for the actual execution of the task. The task manager then waits for the task finish running. The code returned by the task is analyzed by the task manager for potential error conditions. A timeout mechanism can also be incorporated for each individual task manager. The task manager can then timeout the execution of the task after the predefined time interval. This may be helpful in for a number of error handling situations.

4.5 The User Interface

METEOR₂ supports both user tasks involving participation of the users in a workflow application (e.g., to fill a form, validate data, carry out a task on a worklist, etc.), and automated tasks with no user involvement.

Each user task has a template of an HTML form associated with it. Such a template is first automatically created by the workflow designer. Each template contains information regarding which data object attributes a particular user task should display on the form. A template may (and most likely will) be refined by using any commercially available HTML authoring tool (e.g., FrontPage). When a workflow instance reaches a user task, the task's template is instantiated to include the field values taken from the data objects manipulated by this workflow instance. Specifically, the task manager of the user task retrieves the associated HTML form template, accesses the data objects mentioned in the template and then creates an instance of the template using values from the data objects. Through this mechanism, METEOR₂ is capable of separating the specification of data objects necessary for user tasks from their actual use at runtime.

As has been mentioned earlier, all user interaction with a running workflow is done with the use of a Web browser. The requests from the user are transferred from the user domain to the CORBA world. Currently, the communication between the ORBWork and a Web browser is done through a CGI script called the user-CORBA gateway. This script is responsible for taking all the user inputs and packaging them according to the needs of the task managers (CORBA servers). The user fills out an HTML form in a browser and then submits it, which calls the respective CGI

script referenced in the form. The script is, in fact, a CORBA client. The CGI script notifies the associated task manager object by calling the method `Done` (the data from the form are encapsulated within a CORBA object created for this purpose).

We are in the process of replacing this mechanism with facilities provided by Orbix Web (for ORBIX based runtime) and VisiBroker for Java (formerly BlackWidow; for VisiBroker based runtime).

4.6 Handling Workflow Data

Data that is manipulated by tasks participating in a workflow is represented as a collection of CORBA objects. The data object naming scheme associates each object to its workflow instance.

The data objects are specified with the use of the workflow designer and incorporated into the resulting intermediate representation (in WIL format). Then, the workflow code generator creates an appropriate IDL interface for each data object, as it processes the WIL specification.

Each data object supports the necessary functionality for providing its own persistence. This is done as a method, called **Save**, which is either implemented with the use of an externally available persistent storage mechanism (for example an object-oriented database system) or by utilizing persistent object services offered by ORB. The **Save** method is called by task managers on the data objects. To facilitate recovery, the data objects are responsible for saving and restoring their own states. Section 5 will discuss the recovery mechanisms of METEOR₂ in more detail.

4.7 Scalability of ORBWork

Scalability of the enactment system has been a key requirement for workflow systems that has been addressed in ORBWork. Partitioning the scheduling logic within the task manager components removes the need for a centralized scheduling mechanism. We have leveraged the capabilities of the ORB's location independence that allows us to place task managers and data objects on user-specified hosts. During the design process, the designer may indicate specific server nodes where a particular task manager should be placed. The task itself need not be located on the same server as the task manager. A task's placement can be determined at application installation time.

Typically, a workflow instance is a long duration activity; keeping this in mind, we adopted an *on-demand* activation policy for task managers and data objects at run-time. Communication between task managers is limited to method invocations that are light-weight in terms of exchange of data. Data object references are transferred between task managers; this approach is in contrast to that used by the INCA [BMR96] model that passes process and workflow data between processing entities. The recovery framework (see section 5.2) for ORBWork has also been defined in a scaleable manner by partitioning the recovery mechanism across local hosts, and by minimizing the dependence on low-level operating system-specific functionality of the local processing entities.

5 Reliability in ORBWork

ORBWork is subject to numerous errors and failures. These could result from causes that are internal to the WFMS, or due to external factors (such as human errors, workflow task failures, network and hardware failures, etc.). A reliable WFMS should ensure that a well defined framework exists for dealing with errors and recovering from failures.

The METEOR₂ workflow model and run-time architecture of ORBWork have been augmented with features for error handling and recovery. We have used a hierarchical approach to define our error model and recovery framework. This makes error handling and recovery framework more modular. ORBWork includes a layer of CORBA-based system components and failure detection mechanisms that increases its availability and allows recovery of components in a semi-automatic

manner. To be able to flexibly deal with arbitrary errors in a manner that is conformant with the overall goals of the organizational process, ORBWork allows a user to define errors and policies to deal with them.

5.1 The METEOR₂ Error Model

Issues related to error handling have been discussed in the METEOR model [KS95], but this was not followed by an implementation. In METEOR, different errors within tasks lead to a common failure state at the workflow level. In METEOR₂, we have extended this model to provide modeling for tasks at a lower level of granularity so that error-specific corrective policies can be specified by the workflow designer (see Section 3.1). Errors are detected and masked (if possible) as close to the point of occurrence as possible to prevent their propagation to other, unrelated components of the WFMS.

Errors in ORBWork could result from errors in the workflow enactment infrastructure, workflow system components, or due to errors within the individual tasks [WS97]. These errors can be categorized into three broad categories: 1) task errors, 2) task manager errors, and 3) workflow errors. Each of these categories could include both *logical* and *system* errors [KS95].

1. *Task Errors*: Task errors are specific to the individual tasks in the workflow process. They can be represented by the error codes (possibly exceptions) that are returned to the WFMS by the task (or task wrapper), or are sensed by the task manager (e.g., a task segmentation fault) during workflow execution. The task designer is used to describe these errors and the policies to be adopted for recovery. Task errors that have no error handling policies associated with them result in erroneous conditions within the task manager.
2. *Task Manager Errors*: Task manager errors are all errors that affect the normal mode of execution of a task manager. This includes all task errors that could not be resolved and errors resulting from disruptions in the normal mode of execution of the task manager. For example, the latter includes errors occurring during preparation of task inputs, submitting tasks for execution, etc. A task manager error that remains unhandled results in an appropriate workflow error.
3. *Workflow Errors*: This class of errors result from failures in enforcing inter-task dependencies, both data and control, between task managers as defined by the workflow map. Examples of such errors include errors within the task managers (e.g., memory overflow) that adversely affect the scheduling process, communication errors between the scheduler and the task managers, and communication errors between the task manager and the associated input or output data objects

All user-defined task errors are specified as sub-classes of task errors. The workflow designer can define a hierarchy of task-errors and appropriate error handling policies based on the semantics of the task. Task errors are returned to the task manager by the task wrapper that encapsulates the actual task, or by the task itself if it has been implemented accordingly. The **DoTask** (see Section 4) method called by the task manager either succeeds, or returns with a task error that is *detected* by the task manager. Task errors are handled by either activating error-handling tasks that have been specifically designed to handle a specific error or class of errors, or by activating an *alternate* task. In the case of using alternate tasks, we do not guarantee atomicity or isolation of the original task under the presumption that the user realizes the implications of this substitution. In some cases (e.g., non-transactional task execution), it might be necessary to *undo* the effects of a failed task. This could be achieved by implementing an alternate task in an appropriate manner, and by designing the workflow map so as to activate the alternate task if the original task results

in failure. Transactional tasks (e.g., DBMS transactions), due to their inherent characteristics, do not require explicit compensation.

Task manager errors are defined and handled as part of ORBWork. Some of the policies for dealing with task manager errors could be specified at workflow design time. For example, the `UnableToDoTask` task manager error could either be handled by retrying the task a maximum of N times, or by activating an *alternate* task manager that would be able to satisfy the workflow process.

ORBWork has built-in features for recovering from workflow errors. In the case of errors relating to scheduling of task managers on particular hosts, users can specify alternate hosts on which alternate (or the same) task manager could be started. This redundancy makes it possible to mask task scheduling errors involving failed machines, or resource limitations on a particular machine. Errors that cannot be automatically handled by the WFMS are reported to a human via a workflow monitor (see Figure 4) with necessary details such as the cause of the error, the name of the reporting component, and the hostname of the machine where the error occurred.

5.2 Recovery Framework in ORBWork

In this section, we outline the distributed recovery framework of ORBWork. The recovery model assumes a component-based architecture for the WFMS and a communication mechanism (in this case CORBA) that makes it possible to communicate transparent of location, platform, or implementation details of the components (see Figure 4). A semi-automated approach is used to deal with situations that cannot be handled automatically by the recovery framework. The workflow as a whole does not obey the ACID properties of traditional transactional models.

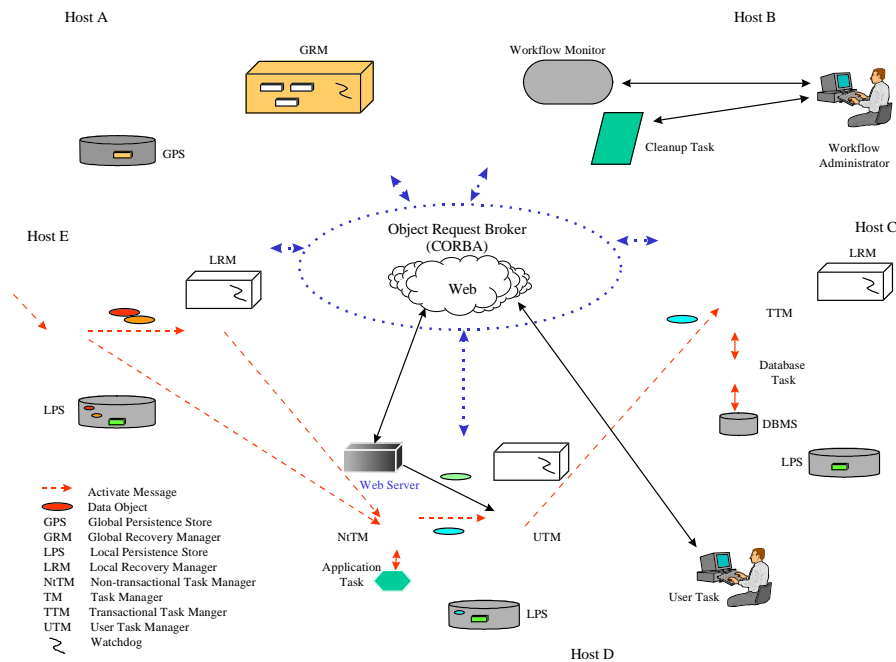


Figure 4: System Schematic for the Recovery Framework in ORBWork

Due to the unavailability of many of CORBA's object services (in particular, the Object Transaction Service) when we started the implementation of our current version, we have 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. Subsequently versions will increasingly use appropriate object services.

5.2.1 Detection of Failures

Failures in distributed systems are hard to sense 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, as in ORBWork. Failures are detected at two levels - at a local (per host) level, and at a more global (multiple hosts) level. To enable this, we have enhanced the workflow enactment system (described in section 4) with additional components in the form of *recovery managers* (see 5.2.2). The recovery managers are themselves CORBA objects and provide the functionality of *detecting* distributed component failures. Failures are monitored by continually polling critical components, or are sensed by *catching* IDL-exceptions during CORBA method invocations on object interfaces. At a programmatic level, we have used *try-catch* [Str91] blocks to clearly separate abnormal workflow execution and recovery actions from normal workflow execution as defined by the workflow map.

5.2.2 The Recovery Managers

A *Local Recovery Manager* (LRM) monitors other local workflow CORBA objects, such as task managers and data objects. It is shared across multiple instances of the workflow process. It also includes functionality in the form of a *workflow object factory* for activating (and reactivating) task manager, and data objects on the local host.

LRMs maintains a *watch-list* of currently registered task managers and data objects that execute on its host. On startup, task managers and data objects *register* with the LRM on their host, and *deregister* when they are no longer required within the workflow process instance. The LRM contains a *watchdog* thread that periodically polls each of the components on its *watch-list* to ensure their liveness. When a failed component is detected, the LRM reactivates it using its factory component. Restarted components have the ability to synchronize their own states and re-register with the LRM.

The LRM logs the critical actions that it performs into a local log (see 5.2.3 and 5.2.4). Critical actions include object registration and deregistration, failure detection, and object creation. This ensures that necessary data is available for forward recovery of the LRM in case of failure. In addition, it checkpoints its logical view of the local system (that is reflected in the current state of its watch-list) to the local log for improving the performance of its recovery process. Checkpointing is done in an atomic manner to ensure the integrity of the checkpointed data.

At the global level, LRM failures are detected by *Global Recovery Managers* (GRMs) that should ideally be activated on reliable hosts within the workflow execution environment. The GRM is responsible for ensuring the availability of LRMs within its domain of responsibility. It uses a *watch-list* and *watchdog* based failure detection mechanism for LRMs similar to that used by the LRMs. Repeated failures in detecting a LRM is reported to an *activation daemon* (available with most ORB implementations) on the relevant host. The activation daemon has the ability to restart CORBA servers locally. On recovery, the LRM would synchronize its state from the local persistence media. It would also create objects that might have failed during its window of recovery.

Failures with the GRMs do not affect the execution of task managers. This would only affect (and be detected by) a LRM that *catches* an exception thrown by the ORB during a registration attempt. In such a case, the activation daemon executing on the GRM's host would restart it. GRMs checkpoint their states in a manner similar to that used by LRMs, and can therefore be recovered to a consistent state upon restart. Using multiple recovery managers enhances redundancy and ensures that only a part of the workflow execution environment is affected by failures. Also, this distributes the overhead of monitoring (polling) objects across the recovery managers thereby reducing performance risks.

5.2.3 Persistence of State

We have used an object-oriented approach towards implementing persistence for the WFMS components. WFMS components have been designed in a manner such that they are responsible for logging their states to stable storage. In our current implementation, we have used a database engine for managing logs; in future we plan to use CORBA's object services. Our approach is very similar to the notion of recoverable objects in the distributed object-oriented framework of *Arjuna* [SDP91]. Recoverable data objects inherit from base interfaces called *data object* and *recoverable*. This gives them the ability to *save* and *restore* their state from stable storage. The persistent layer within each workflow component has been abstracted from the normal functionality of the component, thereby making it easier to map the internal object state to specific storage media (e.g., ASCII files, binary files, DBMS, etc.).

A *Local Persistence Store* (LPS) is used as the stable storage mechanism for logging local data that is critical for recovery purposes. Task managers, data objects, and LRMs use the LPS for logging their internal states. The *Global Persistence Store* (GPS) serves as the persistence storage for GRM relevant data. Logging is done at various stages within the workflow enactment process. Task managers log the state of their tasks including error codes returned by the task, inputs received from other task managers, and outputs returned by their tasks to dependant task managers. Data objects log the state of the workflow data attributes that they encapsulate.

The nature of the underlying persistence media is orthogonal to the functionality of ORBWork. Hence, one could ensure availability of the logged data by using replication techniques that are commonplace in DBMSs today.

5.2.4 Recovery of Task Managers

We have added functionality for logging and recovery to the task managers in ORBWork. A task manager is *initiated* for the first time in *normal* mode. The functionality of the task manager in *normal* mode has already been discussed in section 4. The state of a task manager at any instant in time can be described in terms of the following attributes:

1. State of input tokens (i.e., the input **Activate** messages that it receives from its predecessors) and the data object names associated with each token,
2. Status of the task that it is managing (i.e., *initial*, *execute*, *abort/fail*, *commit/done*); each of these primary states have been further sub-divided into secondary states to closely model execution and to enable error handling, and recovery. For example, the states are represented as (*initial*, *normal*), (*initial*, *end*), (*execute*, *begin*), (*execute*, *end*), etc.
3. State of output tokens (i.e., the output **Activate** messages that it needs to send to its following task managers) and output data objects for each token.

During *normal* mode of execution, the task manager logs changes in these state variables at appropriate stages of execution.

A task manager is initiated in *recover* mode by the LRM on its host if it is unable to detect the existence of the task manager object. We have defined the basic functionality for recovery for each task manager in the **BaseTaskManager** object. Only the recovery actions that are specific to the tasks that they manage are added to the derived task manager classes. During its entire window of recovery, the task manager flags itself as being *inconsistent*, thereby warning all predecessor task managers to wait until it has recovered, before sending *Activate* messages to it. The recovery process itself is performed in an *idempotent* manner by ensuring that the no external data is written or messages sent as part of the recovery process. A backward recovery mechanism is used to restore the state of the task manager by reconstructing (we have omitted the details due to lack of space)

its last logged state from the LPS. Based on its recreated state, the task manger performs the necessary actions to restore its consistency.

If the primary task status that is restored is *initial*, the task manager resets the *inconsistent* flag and is ready to receive activate messages. If the task status is (*execute, begin*), it implies that the task manager faced a system failure during task execution. In the case of a transactional task, the task could be resubmitted due to the ACID nature of the task itself. However, in the case of a non-transactional and user tasks, one needs to validate the current execution status of the task before performing any additional actions. This might involve implementing additional functionality within the task wrapper to allow for checking for multiple submissions for tasks, or reporting the probability of an inconsistent situation to the *Cleanup Task* (see section 5.2.6).

If the restored status of the task manager is (*execute, end*), the task manager retrieves the logged result of the task (which is either success, or the error code resulting from a task error) and proceeds to the success (*commit/done*) or failure state (*abort/fail*) respectively. If the primary restored state of the task manager corresponds to success or failure, the recovery actions involve restoring the control flow within the task manager to the point of activating the next task manager(s) (if any) that have not already been activated.

5.2.5 Recovery of Data Objects

During *normal* execution, logging of data objects is initiated by task managers that use these objects for input or output. Task managers invoke the *save* method on the data object's interface, causing it to save its data. Data objects are logged at the beginning and end of each task with the identity of the task manager that initiates the log. In the event of a task failure, the task manager would invoke the *restore* method on the data object's interface. This would result in the restoration of the *last* logged state of the data object from its log. In the case of initiation in *recover* mode by the LRM, the data objects restore their last logged state from the LPS, and declare themselves *consistent*.

5.2.6 Human-Assisted Recovery

In ORBWork, it might not be possible to mask all errors in an automated fashion (e.g., host failures, network partition failures, non-transactional task failures, global recovery unit and activation daemon failures, etc). In such cases, human assisted recovery is required to bring the WFMS to a consistent state.

The *Workflow Monitor* is a tool that is used as an administrative utility for critical failures that might need attention. Panic error messages that are generated during workflow enactment, are communicated to the monitor along with details regarding the whereabouts of the problem. The monitor can be extended to provide management capabilities for the workflow system.

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 WFMS to a consistent state after such irrecoverable failures.

6 Conclusion and Future Work

Researchers have recognized key requirements for workflow technology intended to support large scale and mission critical organizational processes spanning one or more enterprises. These include

ease of quickly developing executable workflow applications from process modeling and workflow specifications, support for human and automated (both transactional and nontransactional) tasks, support for heterogeneous distributed computing environments, integration with legacy information systems and application codes (that can be individual tasks in a workflow), support for long running processes, scalability, and handling of errors and failures. The METEOR₂ workflow management system addresses these issues. Specifically, the ORBWork enactment system exploits modern commercial distributed object management and Web-based infrastructure, and exploits well known techniques from transaction management to ease in developing a workflow management solution that addresses the above requirements. Details for other completed components of the METEOR₂ system, namely the workflow designer/builder, automatic code generation, another CORBA based (NEOWork which uses Sun's NEO) enactment system with a centralized scheduler, and a Web-only based distributed workflow enactment system (WEBWork) are not discussed here (see <http://lsdis.cs.uga.edu/workflow> for further details).

Our on-going research and development work include support for dynamic and adaptable workflows, a comprehensive repository to support workflow component reuse, support for security (authorization and access control), support for an API for invocation of METEOR₂ supported workflows by other software systems, moving of ORBWork to an ORBIX-based implementation to exploit CORBA's object transaction services, event handling services, etc., and increasing the use of Java and network computing based innovations.

The research funding for the METEOR₂ system explicitly encourages commercialization. Since the start of this project, we have worked with our industry partners in three ways: (a) to jointly develop workflow application requirements, (b) to develop significant demonstration prototypes (as exemplified by the application discussed in [SKM⁺96] which involve five Solaris and NT-based servers, three Web servers, and five databases, also see <http://lsdis.cs.uga.edu/demos>), with follow-on trials, and (c) to support technical testing and evaluation of METEOR's components in the partners' own settings. While we do not give further details for brevity, these activities have had significant positive affect on our research, and additional activities are planned with the LSDIS lab's new industry sponsors. Furthermore, METEOR's technology is being considered for commercial licensing.

Acknowledgements

METEOR team consists of Kemafor Anyanwu, Souvik Das, Prof. Krys Kochut (co-PI), Zonhwei Luo, Prof. John Miller (co-PI), Devanand Palaniswami, Kshitij Shah, Prof. Amit Sheth (PI), Devashish Worah, and Ke Zheng. Key past contributors includes, David Lin, Arun Murugan and Richard Wang.

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 System and Trials, Inc. consortium. See URL: <http://www.scra.org/hiit.html>. Additional partial support and donations are provided by Visigenic, Informix, Iona, Hewlett-Packard Labs, and Boeing.

References

- [AAA⁺95] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, M. Kamath, and R. Guenthoer. 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, August 1995.
- [AAA⁺96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, and R. Gunthor. Advanced Transaction Models in Workflow Contexts. In *Proc. of 12th. IEEE Intl. Conference on Data Engineering*, pages 574–581, New Orleans, LA, February 1996.

- [AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionalities and Limitations of Current Workflow Management Systems. Technical report, IBM Almaden Research Center, 1997. To appear in IEEE Expert.
- [AKA⁺95] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, C. Mohan, and R. Guenthoer. Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System. In S. Lauffmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd. Intl. Conference on Cooperative Information Systems*, pages 99–110, Vienna, Austria, May 1995.
- [ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. 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, August 1992.
- [AS96] G. Alonso and H.J. Schek. Research Issues in Large Workflow Management Systems. In [She96], Athens, GA, May 1996.
- [BDG⁺94] A. Biliris, S. Dar, N. Gehani, H. Jagadish, and K. Ramamritham. ASSET: A System for Supporting Extended Transactions. In *Proc. of ACM SIGMOD Conference on Management of Data*, pages 44–54, Minneapolis, MN, May 1994.
- [BDSS93] Y. Breitbart, A. Deacon, H. Schek, and A. Sheth. Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. *SIGMOD Record*, 22(3):23–30, September 1993.
- [BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
- [BMR96] D. Barbara, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management, Special Issue on Multidatabases*, 7(1):5–15, Winter 1996.
- [BR94] K. P. Birman and R. Van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.
- [BSR96] A. Bonner, A. Shruf, and S. Rozen. LabFlow-1: A Database Benchmark for High Throughput Workflow Management. In *Proc. of the 5th. Intl. Conference on Extending Database Technology*, pages 25–29, Avignon, France, March 1996.
- [CD96] Q. Chen and U. Dayal. A Transactional Nested Process Management System. In *Proc. of 12th. IEEE Intl. Conference on Data Engineering*, pages 566–573, New Orleans, LA, February 1996.
- [CGS97] S. Ceri, P. Grefen, and G. Sanchez. WIDE: A Distributed Architecture for Workflow Management. In *Proc. of 7th. Intl. Workshop on Research Issues in Data Engineering*, Birmingham, England, April 1997.
- [CR91] P. Chrysanthis and K. Ramamritham. A formalism for extended transaction models. In *Proc. of the 17th. Very Large Data Bases Conference*, 1991.
- [Cri91] F. Cristian. Understanding fault tolerant distributed systems. *Communications of the ACM*, 34(2):57–78, 1991.
- [Das97] S. Das. ORBWork: A Distributed CORBA-based Engine for the METEOR₂ Workflow Management System. Master’s thesis, University of Georgia, Athens, GA, 1997. In preparation. URL: <http://LSDIS.cs.uga.edu/>.
- [DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. In *Proc. of the ACM SIGMOD Conference on Management of Data*, 1990.
- [DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transactional Model for Long-running Activities. In *Proc. of the 17th. Intl. Conference on Very Large Data Bases*, pages 113–122, Barcelona, Spain, September 1991.
- [Dog96] A. Dogac. Special-Theme Issue: Multidatabases. In *Journal of Database Management*. Idea Group Publishing, Harrisburg, PA, Winter 1996.
- [EL96] J. Eder and W. Liebhart. Workflow Recovery. In *Proc. of the 1st. IFCIS Conference on Cooperative Information Systems*, Brussels, Belgium, June 1996.

- [Elm92] A. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.
- [Fis95] L. Fischer. *The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering*. Future Strategies, Inc., Alameda, CA, 2nd. edition, 1995.
- [GH94] D. Georgakopoulos and M.F. Hornick. A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows. *Intl. Journal of Intelligent and Cooperative Information Systems*, 3(3):599–617, 1994.
- [GHKM94] D. Georgakopoulos, M. Hornick, P. Krychniak, and F. Manola. Specification and Management of Extended Transactions in a Programmable Transaction Environment. In *Proc. of the 10th. Intl. Conference on Data Engineering*, pages 462–473, Houston, TX, February 1994.
- [GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–154, April 1995.
- [GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [Hol94] D. Hollingsworth. The Workflow Reference Model. Technical Report TC00-1003, Issue 1.1, The Workflow Management Coalition, Brussels, Belgium, November 1994.
- [Hsu93] M. Hsu, editor. *Special Issue on Workflow and Extended Transaction Systems*. IEEE Computer Society, Washington, DC, June 1993.
- [Hsu95] M. Hsu, editor. *Special Issue on Workflow Systems*. IEEE Computer Society, Washington, DC, March 1995.
- [JAD⁺94] S. Joosten, G. Aussems, M. Duitshof, R. Huffmeijer, and E. Mulder. *WA-12: An Empirical Study about the Practice of Workflow Management*. University of Twente, Enschede, The Netherlands, July 1994. Research Monograph.
- [JK97] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, 1997. To be published.
- [JNRS93] W. Jin, L. Ness, M. Rusinkiewicz, and A. Sheth. Concurrency Control and Recovery of Multi-database Work Flows in Telecommunication Applications. In *Proc. of ACM SIGMOD Conference*, May 1993.
- [KLS90] H. F. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proc. of the 16th. Intl. Conference on Very Large Data Bases*, Brisbane, Australia, 1990.
- [KR96] M. Kamath and K. Ramamritham. Bridging the gap between Transaction Management and Workflow Management. In *[She96]*, Athens, GA, May 1996.
- [KS95] N. Krishnakumar and A. Sheth. Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. *Distributed and Parallel Databases*, 3(2):155–186, April 1995.
- [Ley95] F. Leymann. Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. In *GI-Fachtagung Datenbanken in Büro Technik und Wissenschaft*, Dresden, Germany, 1995. Springer-Verlag.
- [Lin97] C. Lin. A Graphical Workflow Designer for the METEOR₂ Workflow Management System. Master's thesis, University of Georgia, Athens, GA, 1997. In preparation.
- [LSV96] F. Leymann, H. J. Schek, and G. Vossen. Transactional Workflows, 1996. Dagstuhl Seminar 9629.
- [Maf96] S. Maffei. PIRANHA: A Hunter of Crashed CORBA Objects. Technical report, Olsend & Associates, Zurich, January 1996.
- [Mos82] J. Moss. 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, July 1982. IEEE CS Press.
- [MSKW96] J. A. Miller, A. P. Sheth, K. J. Kochut, and X. Wang. CORBA-based Run-Time Architectures for Workflow Management Systems. *citedogac*, 7(1):16–27, Winter 1996.

- [Mur95] A. Murugan. Graphical Workflow Designer. Master's thesis, University of Georgia, 1995.
- [OMG95a] OMG. CORBAservices: Common Object Services Specification. Technical report, Object Management Group, March 1995.
- [OMG95b] OMG. The Common Object Request Broker: Architecture and Specification, revision 2.0. Technical report, Object Management Group, July 1995.
- [Pal97] D. Palaniswami. WEBWork: The Web-based Distributed Engine for the METEOR₂ Workflow Management System. Master's thesis, University of Georgia, Athens, GA, 1997. In preparation.
- [RM96a] B. Reinwald and C. Mohan. Structured Workflow Management with Lotus Notes Release 4. In *Proc. of 41st. IEEE Computer Society Intl. Conference*, pages 451–457, Santa Clara, CA, February 1996.
- [RM96b] B. Reinwald and C. Mohan. Structured Workflow Management with Lotus Notes Release 4. In *Proc. of the 41st. IEEE Computer Society Intl. Conference*, Santa Clara, February 1996.
- [RS95] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*. ACM Press, New York, NY, 1995.
- [Saa95] H. Saastamoinen. *On the Handling of Exceptions in Information Systems*. PhD thesis, University of Jyväskylä, 1995.
- [SDP91] S. K. Shrivastava, G. N. Dixon, and G. D. Parrington. An overview of Arjuna: A programming system for reliable distributed computing. *IEEE Software*, 8(1):66–73, January 1991.
- [SGJ+96] A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf. Report from the NSF Workshop on Workflow and Process Automation in Information Systems. Technical report, University of Georgia, UGA-CS-TR-96-003, July 1996. URL: <http://LSDIS.cs.uga.edu/activities/NSF-workflow>.
- [She95] A. Sheth. Tutorial Notes on Workflow Automation: Application, Technology and Research. Technical report, University of Georgia, May 1995. presented at ACM SIGMOD, San Jose, CA, URL: <http://LSDIS.cs.uga.edu/publications>.
- [She96] A. Sheth. Proc. of the NSF workshop on workflow and process automation in information systems. University of Georgia, May 1996. URL: <http://LSDIS.cs.uga.edu/activities/NSF-workflow>.
- [Sil95] B. R. Silver. The BIS Guide to Workflow Software: A Visual Comparison of Today's Leading Products. Technical report, BIS Strategic Decisions, Norwell, MA, September 1995.
- [SJ96] A. Sheth and S. Joosten. Workshop on Workflow Management: Research, Technology, Products, Applications and Experiences, August 1996.
- [SJHB96] H. Schuster, S. Jablonski, P. Heintz, and C. Bussler. A General Framework for the Execution of Heterogeneous Programs in Workflow Management Systems. In *Proc. of the 1st. IFCS Intl. Conference on Cooperative Information Systems*, Brussels, Belgium, June 1996.
- [SJKB94] H. Schuster, S. Jablonski, T. Kirsche, and C. Bussler. A Client/Server Architecture for Distributed Workflow Management Systems. In *Proc. of the 3rd. Intl. Conference on Parallel and Distributed Information Systems*, pages 253–256, Austin, TX, September 1994.
- [SKM+96] A. Sheth, K. J. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. Supporting State-Wide Immunization Tracking using Multi-Paradigm Workflow Technology. In *Proc. of the 22nd. Intl. Conference on Very Large Data Bases*, Bombay, India, September 1996.
- [Str91] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, 2nd edition, 1991.
- [Wan95] X. Wang. Implementation and Performance Evaluation of CORBA-Based Centralized Workflow Schedulers. Master's thesis, University of Georgia, August 1995.
- [Wei93] G. Weikum. Extending Transaction Management to Capture More Consistency With Better Performance. In *Proc. of the 9th. French Database Conference*, pages 27–30, Toulouse, 1993.
- [WR92] H. Wachter and A. Reuter. The ConTract model. In *[Elm92]*, chapter 7. Morgan Kaufman, 1992.

- [WS96] D. Worah and A. Sheth. What do Advanced Transaction Models Have to Offer for Workflows? In *Proc. of Intl. Workshop on Advanced Transaction Models and Architectures*, Goa, India, 1996.
- [WS97] D. Worah and A. Sheth. Transactions in Transactional Workflows. In *[JK97]*, chapter 1. Kluwer Academic Publishers, 1997.
- [WWWD96] D. Wodtke, J. Weissenfels, G. Weikum, and A. K. Dittrich. 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, February 1996.