Failure Handling in Transactional Workflows Utilizing CORBA 2.0

Casimir J. Crawley
Senior Project Engineer
Delco Electronics Corporation
Kokomo, Indiana

Omran Bukhres
Department of Computer Science
Purdue University
Indianapolis, Indiana

Abstract

Transactional workflows have been previously specified using commercially-available workflow management systems (WFMSs). WFMSs have facilitated this specification by providing task coordination and execution capabilities. However, these WFMSs presently have limitations in terms of heterogeneous distributed system integration, non-proprietary cross-platform support, flexible ACID property support, and database availability. A key weakness is their lack of recovery support which is required for most relaxed transactional models. Transactional workflows use compensating transactions for recovery after system aborts or task failures.

The Object Management Group (OMG) has specified the Common Object Request Broker Architecture (CORBA) as an open-architecture, object-oriented model which addresses the system integration and platform support issues. This paper makes two contributions towards workflow management systems. First, a CORBA-based architecture is presented. Second, a minor modification to CORBA's Interface Definition Language (IDL) grammar is also presented. The architecture resolves the system integration issue which hampers current WFMSs. It also provides high availability between heterogeneous databases and their clients. The IDL grammar modification provides a method by which compensating transactions in transactional workflows can be specified. This approach incorporates compensating subtransactions from relaxed transaction models with CORBA's cross-platform support and heterogeneous system integration capability to resolve semantic failures in transactional workflows.

keywords: advanced transactions, transactional workflows, databases, heterogeneity, CORBA, semantic failures, workflow transaction models

1 Introduction

Workflow management is an area of intense research in both academic and commercial circles. Businesses need to operate on a global scale in order to meet the competitive demands of the international marketplace. Companies have discovered workflow management as a key technology for integrating and automating their enterprise-wide operations. The academic community has responded by researching both traditional and non-traditional transaction management and applying its concepts to workflow management.

Rusinkiewicz and Sheth [16] define workflows as "activities involving the coordinated execution of multiple tasks by different processing entities". A *task* is simply a "*unit of work*" in some workflow environment while a *processing entity* is whatever entity (human or computer) performs the task [16, 17]. Workflows have also been described as models of the *business process* [14] where a business process describes an enterprise's business activities. Defining a business process into a workflow specification is termed *process modeling*. The Workflow Management Coalition was formed in 1993 with a key objective of standardizing these workflow specifications [8]. This coalition defines the business process as "a procedure where documents, information or tasks are passed between participants according to defined sets of rules to achieve, or contribute to, an overall business goal. " [18].

A workflow management system (WFMS) provides "the ability to specify, execute, report on, and dynamically control workflows involving multiple humans and HAD (heterogeneous, autonomous, distributed) systems" [8]. The Workflow Management Coalition defines a WFMS as "a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic" [18].

The current capabilities of commercial WFMSs are outlined in [8]. Current WFMSs lack the systems architecture and interoperability necessary for enterprise-wide integration via application programming interfaces (API) or other protocols. They also lack correctness and reliability support which provide concurrency control, recovery, and transaction coordination. Both of these capabilities are necessary for HAD system integration and transactional workflow support.

Figure 1 [9] characterizes workflows by plotting them on a graph with *human-oriented* workflow intervention versus *system-oriented* workflow intervention as dimensions. This intervention is needed to satisfy task control, task coordination, workflow correctness maintenance, and workflow reliability maintenance.

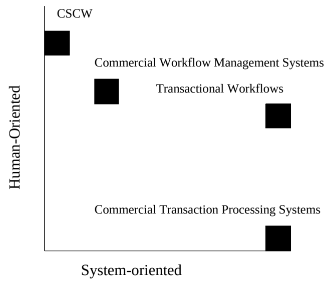

Figure 1: Workflow Characterization

Figure 1 [9] delimits workflows and contrasts them using four broad categories:

CSCW Computer supported cooperative work [8] describes workflows which utilize computers but still rely upon human intervention for all the previously mentioned requirements. These workflows have the highest degree of human-orientation with the lowest degree of system-orientation. The majority of present-day office desktop computing environments fit into this category.

Commercial Workflow Management Systems These present-day WFMSs provide system-oriented task control and coordination but are still heavily dependent upon human intervention for workflow correctness and reliability in the face of system aborts or task failures. The tasks are typically manual while the workflow state is recorded in a database.

Transactional Workflows These workflows are system-oriented in all the previously listed requirements and are derived from advanced transaction models. Transactional workflows combine the flexibility of commercial WFMSs while maintaining workflow correctness and reliability. These workflows and their examples are discussed in the next section.

Commercial Transactional Processing Systems Commercial transactional processing (TP) systems fully support all ACID properties. These properties and their implications are discussed in the next section.

Figure 1 demonstrates that transactional workflows fit between commercial TP systems and commercial WFMSs. Transactional workflows combine the recovery and concurrency handling features of TP systems with the distributed task control and coordination features of WFMSs.

Current WFMSs exhibit two main deficiencies: first, they lack recovery mechanisms necessary for the inevitable workflow system or semantic failures; and second, they lack the ability to integrate heterogeneous autonomous systems which are prevalent in the commercial environment. This paper addresses both deficiencies. It transplants the recovery features from advanced transactional models into workflow management systems by modifying CORBA's IDL to specify semantic failure handling. It also integrates heterogeneous systems by using CORBA's object-oriented, distributed, and open client-server architecture.

The rest of the paper consists of four sections. Section 2 provides a short background on advanced transactional models. It also describes semantic failures and sagas in order to illustrate the need for recovery support in workflows. Section 3 provides an overview of CORBA and then presents a CORBA-based architecture. It also identifies CORBA's IDL limitations as a workflow language and proposes an IDL grammar modification which enables failure handling. Section 4 discusses related work and shows how this paper fits into current research. Section 5 concludes this paper.

2 Advanced Transactional Models

Transactions have been categorized as *traditional* or *advanced*. A traditional transaction completely satisfies the classic *atomicity*, *consistency*, *isolation*, and *durability* (ACID) properties of conventional database management systems (DBMS) [1] and other systems such as TP monitors [8]. The ACID properties are fully supported by TP systems which regulate transaction execution, commitment, and recovery through the use of protocols.

The traditional transaction model's full and strict support of the ACID properties severely limits its usefulness in advanced applications such as multidatabase systems, WFMSs, and long-lived transaction systems [12]. As a consequence, *advanced transaction* models have been proposed in order to broaden the suitability of transactional concepts [3, 12]. Advanced transactions have been categorized into *extended transaction models* and *relaxed transaction models* [17]. Extended transaction models contain operations or *subtransactions* which form hierarchical structures within a given transaction. Two extended transaction examples are *nested* and *multi-level* transactions. Relaxed transaction models selectively relax the ACID properties of atomicity and isolation.

Applications which utilize transactional concepts in managing distributed and heterogeneous tasks have been identified as *multi-system transactional workflows* [17]. Further research has built upon this concept by implementing advanced transaction models in commercial WFMSs [2, 3]. The *Workflow Interface Definition Language* (WIDL) has been presented as a method of specifying workflows by adding intertask dependency specifications to CORBA's IDL[13]. This paper builds on the foundation of this previous research by further modifying IDL in order to accommodate relaxed transactional workflows.

2.1 Semantic Failures

Since workflows typically execute in a distributed computing environment, the potential for system downtime exists at numerous locations. Task aborts resulting from system downtime are termed *system failures*. *Semantic failures* occur when a workflow cannot execute as specified because of a task failure in the context of a business process. Examples include the overwithdrawal of money from a bank account or the delivery of an out-of-stock item [4].

It has been suggested that a workflow's *failure atomicity* requirements could be specified [16]. A workflow task failure at one site could trigger the execution of an equivalent task at another. These requirements can be articulated by defining the *acceptable termination states* of a given workflow. Any other termination state violates workflow failure atomicity requirements and is termed a *non-acceptable* *termination state*. Acceptable termination states are further categorized as *committed* or *aborted*. A committed acceptable termination state completely fulfills a workflow's business process objectives. An aborted acceptable termination state does not meet a workflow's business process objectives but does meet the specified failure atomicity requirements. If a given workflow task fails, previously committed task or subtransaction results may need undoing in order to satisfy failure atomicity requirements. Tasks which perform this function are termed *compensating tasks* or *compensating subtransactions* [10]. This paper's objective is to provide a means of specifying compensating subtransactions utilizing CORBA. This will enable advanced transactional models to be implemented in a CORBA-based workflow environment.

2.2 Sagas

Sagas were introduced in [7] as a transaction model which could handle the issues raised by long-lived transactions. A saga is formally defined as follows [12, 4, 7]: let $T_1, T_2, ..., T_n$ represent traditional subtransactions which fully support all ACID properties in saga T. Let $C_1, C_2, ..., C_n$ represent compensating subtransactions which correspond to each $T_1, T_2, ..., T_n$ traditional subtransaction. A system which implements the saga model must enforce either the execution sequence

$$T_1, T_2, ..., T_n$$

which results in a committed acceptable termination state, or the execution sequence

$$T_1, T_2, ..., T_j, C_j, ..., C_2, C_1$$

for some $1 \leq j < n$ which results in an aborted acceptable termination state. In summary, each subtransaction T_j has its corresponding C_j compensating subtransaction which semantically undoes any actions T_j executed and committed. The saga relaxes the atomicity of these long-lived transactions by allowing subtransaction access to shared resources even though each individual subtransaction fully supports the ACID properties.

The saga model demonstrates the need for compensating subtransactions since it provides a formal means of handling semantic failures. It is this necessity which motivates this paper's provision for the specification of such subtransactions utilizing CORBA.

3 CORBA

3.1 CORBA Overview

The Common Object Request Broker Architecture (CORBA) is the Object Management Group's (OMG) specification for an implementation-independent architecture which supports the integration of distributed and possibly heterogeneous systems through an object-oriented interface. A CORBA object can be client or a server depending upon the request sent to it.

The OMG [15] defines the entity requesting a service as the *client* and the entity providing the service as the *object*. The act of soliciting the service is termed a *request* while any request parameter is termed a *value*. An *object name* is a value which identifies an object. A value can be defined as an input, output, or both input and output parameter. A request may optionally return a single *result value* after its invocation. If a request invocation results in an abnormal condition, *exceptions* are returned to the client which can be user-defined or system generated *standard exceptions* [15]. This paper proposes the exploitation of this exception handling feature in order to facilitate compensating subtransactions. This proposal is detailed later in the paper.

Figure 2 [11] illustrates the structure of OMG's Reference Model. The ORB is the middleman which provides access to the other three OMG elements. As previously mentioned, the ORB also provides the connectivity between clients and objects. Therefore, the ORB can be thought of as a common software bus between distributed objects, systems, applications, and any CORBA-provided service. This model specifies an open architecture which provides an object-oriented, cross-platform approach which can integrate a distributed environment.

CORBA's ORB interface is outlined in Figure 3 [15]. A client can choose either a *static* or *dynamic* interface to an *Object Implementation*. The object implementation provides server object semantics by defining object instance data and method code [15]. An *Object Reference* specifies an object within an ORB and provides an object implementation handle to a client which is valid only for the lifetime of the client. The *Object Adapter* is the primary ORB service provider to object implementations. The Object Adapter performs various housekeeping duties such as object instantiation, reference management, class registration, and client request routing. A static interface for a given object implementation is defined prior to a client application's compilation using CORBA's Interface Definition Language (IDL). An IDL request can be either *synchronous* with the client waiting for an object response or *asynchronous* with the client signaled when an object has responded.

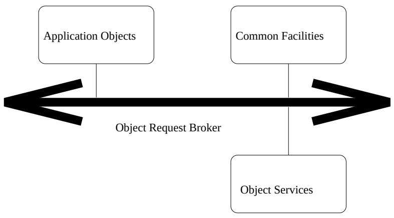

Figure 2: OMG Reference Model Diagram

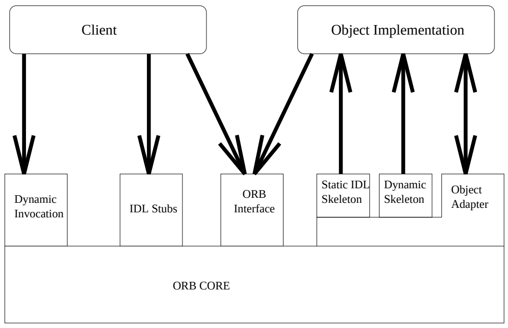

Figure 3: ORB Interface Diagram

It should be emphasized that IDL is a *definition*, not an implementation language. IDL simply specifies interfaces to objects and their implementations. A client or object application must still be realized using some implementation language. This is one of CORBA's key features which enables its cross-platform support for heterogeneous computing environments. As shown in Figure 3, IDL provides the request interface for clients via *stubs* and object implementations via *skeletons*. The stubs and skeletons are are basically templates generated by the IDL compiler from user-supplied IDL statements. These templates provide the framework from which object requests can be physically implemented.

3.2 CORBA-Based Architecture

Five different run-time CORBA-based architectures are presented in [13]: Highly Centralized, Synchronous Centralized, Asynchronous Centralized, Semi-Distributed, and Fully Distributed. Each architecture is classified by its scheduler implementation. The main entities in these architectures are task, task manager, scheduler, monitoring service, and dispatching service. These entities use synchronous and asynchronous IDL for communication depending upon the architecture. The asynchronous IDL is very attractive since it allows the entities to continue execution without waiting for a communication response.

Workflow architectures using IBM's Flowmark are described in [4]. Issues such as availability, reliability, and scalability are important factors in architecture configuration. The architecture decision criteria centers on shared database access between workflow tasks and workflow state maintenance by workflow servers. Four different architectures are identified: 1. centralized database, 2. distributed database, 3. replicated database, 4. interconnected workflow servers, and 5. clustered workflow servers.

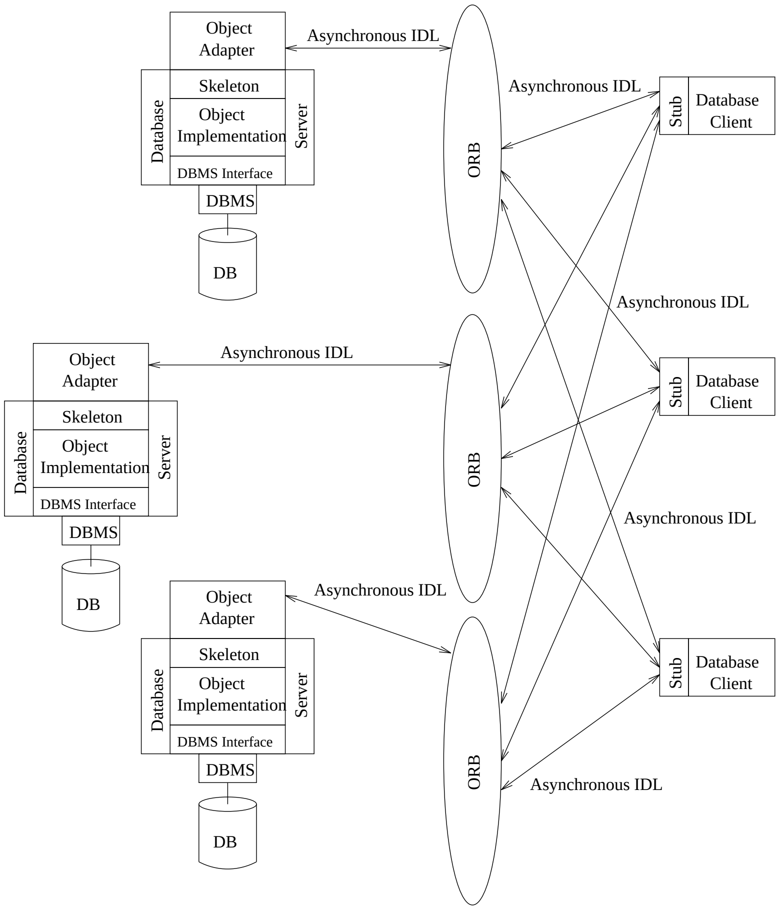

Figure 4: CORBA-Based Architecture

A centralized database is easily administered since there should be no data inconsistency. However, this solution scales poorly and is undesirable from a reliability and availability standpoint. A distributed database is more reliable and available but may not exist or is impractical in heterogeneous workflow environments. Replicated databases offer reliability and availability but workflow state maintenance becomes more complicated. This increased complication could potentially affect scalability. Interconnected workflow servers offer availability and reliability but could suffer performance-wise when workflow functions are executed. The authors finally propose a hybrid solution with clustered workflow servers and replicated databases. The replicated databases contain only persistent data or data that seldom changes. The workflow servers maintain the workflow state and are distributed in clusters around the replicated databases.

This hybrid solution offers the best trade-off between the competing interests of availability, scalability, and reliability. Consequently, this paper combines the best features of the architectures presented in [13] and [4]. The proposed architecture is shown in Figure 4. Each ORB provides an asynchronous IDL interface between the replicated database labeled DB and a Database Client's stub. A replicated database interfaces to the ORB via its Database Server. Each Database Server contains the required Object Adapter, Skeleton, and Object Implementation CORBA entities. The Object Implementation communicates with the replicated database's DBMS through the DBMS interface. Future research plans include a scheduler implementation using Workflow Interface Definition Language.

The replicated databases contain the persistent or seldom-changing data. Multiple ORB's provide database access to any Database Clients. The asynchronous IDL allows all entities to continue processing without waiting for a database response. As a consequence, the data replication and multiple ORBs offer a reliable, scalable, and available architecture which best serves a workflow environment.

3.3 CORBA Discussion

It should be emphasized that CORBA is a *specification*, not an implementation. This isolates CORBA from any particular computing platform which in turn allows the architecture to function in heterogeneous environments. A table that lists several CORBA implementations can be found in [13].

Four important requirements for general purpose workflow languages are outlined in [6]:

Advanced Control Mechanisms These mechanisms are described by the following specifications:

1. Task execution state and data dependency specification.
2. Sequential and parallel task execution specification.
3. External constraint specification.

System Integration Support This support is required for the integration of distributed and heterogeneous systems which are prevalent in workflow environments.

Object Model The object model is the best method of capturing data integration and system behavior semantics.

Transaction Capability This is required for the controlled relaxation of the ACID properties for the implementation of advanced transactional models. *Semantic compensation* is necessary for both Sagas and Flex Transactions.

CORBA certainly meets the needs of object model and system integration support. Indeed, CORBA's basic architecture is derived from the Object Management Group's Object Model Architecture [15]. This architecture's object-orientation was previously discussed. CORBA supports system integration by incorporating the client-server paradigm which focuses on "point-to-point" message passing between two objects [5].

CORBA is an excellent candidate for transactional workflow implementation since it is intended as a distributed, cross-platform, object-oriented, and open architecture. An architecture with these attributes offers great potential for becoming a pervasive technology. However, CORBA has limitations which must be addressed for such an implementation.

3.4 Specifying Compensating Subtransactions with CORBA IDL

CORBA lacks the advanced control mechanisms necessary for workflow support. IDL is focused on facilitating communications between clients and objects. The semantics of these communications must be interpreted at a client level. As such, CORBA's IDL lacks the expressiveness necessary to implement either transactional or non-transactional workflows [6]. Consequently, CORBA's IDL must be extended to support workflows. Miller, Sheth, et al., in [13] created the Workflow Interface Definition Language (WIDL) by proposing the "*enabled by*" clause addition to IDL. The syntax of the clause is repeated below:

```
enabled by ([<task_1>, <state_1>] && [<task_2, state_2>] && <condition>)
    [ "" | at <time> | after <time> ]
```

[<task_1>, <state_1>] represents a task-state vector predicate while <condition> represents a Boolean expression predicate. The clause can have any number of predicates and can optionally include time constraints specified by <time>. The significance of this work is the application of Event-Condition-Action (ECA) rules to CORBA's IDL. As a result, WIDL allows task state and data dependency specification, sequential and parallel task execution specification, and external constraint specification.

The "*raises*" expression lists all exceptions that might arise from the invocation of its accompanying method. IDL exceptions are data structures which may be returned to the client when an exceptional condition occurs. OMG IDL grammar uses a syntax that is similar to Extended Backus-Naur Form (EBNF). The existing "raises" expression is as follows:

```
<raises_expr> ::= "raises" "(" <scoped_name> { "," <scoped_name> }* ")"
```

where <scoped_name> is a previously defined exception. The expression's syntax is modified as follows:

```
<raises_expr> ::= "raises" "(" <scoped_name> { "," <scoped_name> }* ")"
              |   "raises" "(" <scoped_name> ","
                  { "[" <scoped_name> "," <identifier> "]" }+
                  { "," <scoped_name> ","
                  { "[" <scoped_name> "," <identifier> "]" }+ }* ")"
```

where the "[" <scoped_name>, <identifier> "]" predicate identifies a task-state vector which is enabled by the exception <scoped_name>. <scoped_name> in the "[" <scoped_name>, <identifier> "]" predicate must reference a previously defined interface name while <identifier> must reference a previously defined operation name within the scope of <scoped_name>'s interface. Now if an exception is raised as the result of a corresponding operation invocation, a compensating subtransaction task is enabled.

The example below illustrates the modified expression's implementation and is modified from the one outlined in [13]: The example WIDL is shown below:

```
struct Money_Transfer {
  long account_no;
  double money;
  long retries;
}; // Money Transfer

interface Withdrawal_Task : TTask {
  exception Overdrawn {
    string<80> errmsg;
  }
  long Execute(in Money_Transfer withdrawal)
    raises(Overdrawn, [Deposit_Task, Execute], [Deposit_Task, Commit]);
  long Abort();
```

```
  long Commit();
} // Withdrawal_Task

interface Deposit_Task : TTask {
  long Execute (in Money_Transfer deposit);
  long Abort();
  long Commit();
} // Deposit_Task
```

As in [13], `Withdrawal_Task` and `Deposit_Task` inherit an abstract base class named `TTask`. The structure `Money_Transfer` is an IDL constructed type which is used as the request parameter. The exception `Overdrawn` is raised when insufficient funds exist at the requested bank account. The task-state vectors `[Deposit_Task, Execute]` and `[Deposit_Task,Commit]` are enabled when `Overdrawn` is raised.

This is a simple example with two subtransactions which withdraw funds from two different banks. T_1 withdraws money from bank 1 and then T_2 withdraws money from bank 2. The compensating subtransaction in this example is C_1 which returns the withdrawn money to bank 1. The `Overdrawn` exception is raised when the bank 2 account is overdrawn. If this event occurs, the compensating subtransaction `Deposit_Task`'s task-state vectors `[Deposit_Task, Execute]` and `[Deposit_Task, Commit]` are enabled. As a consequence, `Deposit_Task` returns the first withdrawal to bank 1. This example illustrates the backward recovery feature of the modified IDL.

This is a simple example with a contingency subtransaction which deposits funds into an overdrawn account. T_1 withdraws money from a bank account and the contingency subtransaction C_1 deposits money into the bank account if the account is overdrawn. The `Overdrawn` exception is raised when the bank account is overdrawn. If this event occurs, the contingency subtransaction `Deposit_Task`'s task-state vectors `[Deposit_Task, Execute]` and `[Deposit_Task, Commit]` are enabled. As a consequence, `Deposit_Task` deposits money into the overdrawn account. This example illustrates the forward recovery feature of the modified IDL.

4 Related Work

An abstract model of task behavior using a state transition diagram was presented in [17]. This model was termed a *task skeleton*. State diagram nodes represented externally visible task states while transition edges represented events that forced a given task from one state to another. This model was extended in [13] for the classification of *transactional* and *non-transactional* tasks. Transactional tasks were identified as minimally supporting the atomicity property and maximally supporting all ACID properties. The externally visible states in transactional tasks were termed *initial*, *executing*, *aborted*, and *committed*. This paper incorporates this task model by adopting the task-state vector concept as part of its modified IDL. This work differs in that its modified IDL allows semantic failure handling.

The application of advanced transaction models into the FlowMark WFMS has been researched as part of IBM's Exotica project [2, 3, 14]. Sagas and flexible transactions have been implemented in FlowMark in order to demonstrate how semantic failures could be handled in a workflow environment [4]. Workflow performance issues such as availability, scalability, and reliability and their impact on workflow architectures were also explored. The CORBA-based architecture in this paper uses the replicated database and workflow server concepts from this research for an optimum configuration. It differs in that its implementation is CORBA rather than FlowMark-based.

In summary, the work presented in this paper complements the previous research work on transactional workflows. It is hoped that the combination of CORBA's flexibility with advanced transaction models' discipline will create a workflow environment which will solve the apparent problems in current workflow management systems.

5 Conclusions

The research in this paper has revealed that CORBA has an opportunity to become the glue which can bind distributed systems together. CORBA's IDL offers the correct level of abstraction with its object-oriented architecture for heterogeneous system integration. It has been demonstrated in this paper and in [13] that CORBA IDL needs only minor modifications for workflow applications. This is a tribute to its extensibility, scalability, and expressiveness. One of this paper's contributions is that its modified IDL augments WIDL so that when the two are combined a powerful means of specifying transactional workflows is created.

The other contribution of the paper is the CORBA-based architecture which provides a solid foundation for further research. Its database replication and distributed ORBs offer the best compromise between availability, scalability, and reliability. Further research will investigate scheduler implementation and decide whether the centralized versus distributed architecture is superior.

Workflow systems require an infrastructure for reliably capturing the states of business processes and tying together distributed applications and DBMSs. In general, advanced transactions models have been found to be the most natural choice for this infrastructure. As workflow systems scale to the enterprise level, the requirements on robustness, scalability, and distribution are linked to those required and developed in TP monitors and DBMSs. This system evolution will lead to the fusion of workflows with transactional concepts yielding transactional workflows as the final outcome.

References

[1] D. Agrawal and A. El Abbadi. Transaction management in database systems. In A. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 1, pages 1–31. Morgan Kaufmann, San Mateo, CA, 1992.

[2] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *Proceedings of the 12th International Conference on Data Engineering, New Orleans, LA*, Feb. 1996. Also available as *IBM Research Report RJ9970*, IBM Almaden Research Center, July 1995.

[3] G. Alonso, M. Kamath, A. El Abbadi, D. Agrawal, R. Günthör, and C. Mohan. Exotica: A project on advanced transaction management and workflow systems. *SIGOIS Bulletin (Special Issue on Business Process Management Systems: Concepts, Methods and Technology)*, 16(1):45–50, Aug. 1995.

[4] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan. Failure handling in large scale workflow management systems. IBM Research Report RJ9913, IBM Almaden Research Center, Nov. 1994.

[5] A. Chan and K. Harty. Building flexible distributed applications with the Technekron Enterprise Toolkit. *Data Engineering Bulletin (Special Issue on Infrastructure for Business Process Management)*, 18(1):33–40, Mar. 1995.

[6] A. Forst, E. Kuhn, and O. Bukhres. General purpose workflow languages. *Distributed and Parallel Databases*, 3(2):187–218, Apr. 1995.

[7] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM Conference on Management of Data*, pages 249–259, May 1987.

[8] 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):119–153, Apr. 1995.

[9] D. Georgakopoulus and M. Rusinkiewicz. From transactions to transactional workflows. 12th International Conference on Data Engineering tutorial notes, New Orleans, LA, Feb. 1996.

[10] J. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the 7th International Conference on VLDB*, Sept. 1981.

[11] S. Helal and R. Elmasri. Open standards developments for database interoperability. 12th International Conference on Data Engineering tutorial notes, Feb. 1996.

[12] Y. Leu, J. Mullen, and O. Bukhres. Introduction to advanced transaction models. In A. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 2, pages 33–52. Morgan Kaufmann, San Mateo, CA, 1992.

[13] J. Miller, A. Sheth, K. Kochut, and X. Wang. CORBA-based run-time architectures for workflow management systems. *Journal of Database Management, Special Issue on Multidatabases*, 7(1):16–17, 1996.

[14] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica: A research perspective on workflow management systems. *Data Engineering Bulletin (Special Issue on Infrastructure for Business Process Management)*, 18(1):19–26, Mar. 1995.

[15] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, revision 2.0 edition, Jul. 1995.

[16] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, chapter 29, pages 592–620. ACM Press, New York, NY, 1995.

[17] A. Sheth and M. Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, 16(2):1–4, Jun. 1993.

[18] The Workflow Management Coalition, Workgroup 1 A. *The Workflow Reference Model*, Feb. 1995. Document No. WFMC-WG01-1000.