Open access • Book Chapter • DOI:10.1007/BFB0022052

Two-Layer Transaction Management for Workflow Management Applications — Source link

Paul Grefen, Jochem Vonk, E.M. Boertjes, Peter M. G. Apers

Institutions: University of Twente

Published on: 01 Sep 1997 - Database and Expert Systems Applications

Topics: Distributed transaction, Transaction processing, Transaction processing system, Transaction data and Online transaction processing

Two-Layer Transaction Management for Workflow Management Applications[†]

Paul Grefen, Jochem Vonk, Erik Boertjes, Peter Apers
Center for Telematics and Information Technology
University of Twente
{grefen,vonk,boertjes,apers}@cs.utwente.nl

Abstract

Workflow management applications require advanced transaction management that is not offered by traditional database systems. For this reason, a number of extended transaction models has been proposed in the past. None of these models seems completely adequate, though, because workflow management requires different transactional semantics on different process levels. In the WIDE ESPRIT project, a two-layer transaction management approach has been adopted to cope with this problem. The approach consists of a transaction model built from an orthogonal combination of two existing models and a transaction management architecture with two independent transaction managers. This architecture is integrated into the next generation of the commercial FORO distributed workflow management system.

1. Introduction

Workflow management applications require transaction management functionality that goes beyond the traditional simple transaction model provided by current database management systems. In particular, support for long running activities with relaxed notions of isolation and atomicity and complex process structures is required. As indicated by the large number of proposed transaction models, no single model can effectively cope with the broad set of requirements imposed by complex workflow management applications. On a high level of granularity in these applications, a relaxed notion of transactionality is required to allow cooperativeness between multiple workflow tasks. On a lower level of granularity, stricter transactional notions are required to model business transactions that may involve complex process structures and multiple actors but require atomicity and isolation semantics.

In the WIDE ESPRIT project, the approach has been taken therefore to use a combination of modified existing transaction models, instead of inventing yet another new model. The result is an orthogonal two-layer transaction model that supports both high-level and low-level workflow semantics. The two-layer model is supported by two independent transaction manager modules, each of which manages one layer of the model. These modules are implemented on top of a commercial DBMS. The resulting transaction management architecture is integrated into the next generation of the FORO workflow management system (WFMS) with specific attention to distribution aspects and platform independence.

[†] The work presented in this paper is supported by the European Commission in the WIDE project (ESPRIT No. 20280). Partners in WIDE are Sema Group sae and Hospital General de Manresa in Spain, Politecnico di Milano in Italy, ING Bank and University of Twente in the Netherlands.

This paper is organized as follows. In Section 2, we first give an overview of related work. In Section 3, we present the process model underlying the workflow model constructed in the WIDE project. Section 4 discusses the transaction model dealing with the requirements following from the process model. The functional design of the software architecture supporting the transaction model is next presented in Section 5, the implementation in the context of the FORO WFMS in Section 6. We conclude the paper with a short discussion and outlook on future work.

2. Related work

In the past decade, numerous extended transaction models have been proposed for long running transactions [El92]. Examples are nested transactions [Da91], sagas [Ga87], and contracts [Re95]. General frameworks have been constructed, like ACTA [Ch94], that provide a conceptual framework for extended transaction models. Various extended transaction models have been proposed for use in workflow management contexts [Lo93]. In WIDE, we do not aim at the specification of yet another transaction model, but at the combined use of concepts from existing models. In contrast to many other proposals, we aim at an industry-strength implementation of extended transaction support.

In the Exotica project [Al96], advanced transaction models are emulated by means of the Flowmark WFMS, thereby trying to remove the need for advanced transaction support. In WIDE, we aim at advanced transaction support that is orthogonal to workflow management functionality. Although the basic ideas are quite different between Exotica and WIDE, some aspects are common. This will be made clearer in the sequel of this paper.

In WIDE, we provide extended transaction management on top of a commercial DBMS platform. In [Ba95], the reflective transaction framework is presented that provides extended transaction support using transaction adapters. There are a number of important differences to our work. The reflective framework provides flexible transaction semantics through reflection, whereas we provide flexibility through a two-layer model with multiple levels in each layer. Further, the reflective framework uses a transaction monitor with an open architecture (Transarc's Encina), where we use a closed database platform (Oracle). Finally, the reflective framework aims at a prototype realization, where we aim at integration into a commercial product.

3. The WIDE workflow management process model

In the WIDE project, an extended workflow model and language are developed with advanced process primitives like multitasks, various join operators, exceptions, etc. [C96a]. Important for this paper is the fact that a *multi-level process model* is used (see [Gr97] for an ER-diagram of the model) that allows for hierarchical decomposition of workflow processes with flexible transactional semantics.

The top level of a process hierarchy is formed by a complete workflow process. The bottom level consists of individual tasks, i.e. process parts that are not further decomposed in the workflow specification. Usually, an individual task is performed by a single actor in a short period of time. In the process hierarchy, the higher levels are long-running processes with cooperative characteristics and therefore require re-

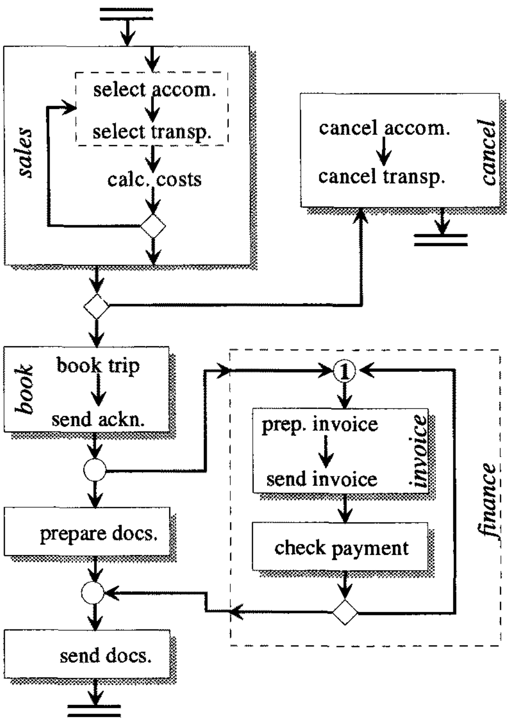

Figure 1: Example business process

laxed transactional semantics. The lower levels are relatively short-living processes requiring strict transactional semantics. The separation between the higher and lower levels is formed by the notion of *business transaction* in the workflow application. Process levels representing business transactions and their subprocesses have strict transactional semantics; their superprocesses have relaxed semantics.

In the WIDE model, the process levels above business transactions are represented by *non-atomic supertasks*. These supertasks do not behave strictly atomically, as this would imply the undoing of large amounts of work in case of an error. Also, they are not executed in strict isolation, as strict isolation would prevent the sharing of information as required in a workflow management environment. A rollback mechanism at this level is required though, to be able to undo a workflow to a certain point in case of errors. Rollback should offer application-oriented semantics, i.e. it should return a workflow to a state that is identical to a previous state from a business point of view, not necessarily from a database point of view.

The process levels associated with business transactions and below are represented by *atomic supertasks and tasks*. These supertasks should ideally be executed in strict atomicity and isolation. A rollback mechanism should offer complete undo to the pre-supertask state in case of critical errors. Besides normal atomic supertasks, *non-critical supertasks* need to be supported. Non-critical supertasks allow the definition of process parts that cannot cause critical errors and hence do not require rollback functionality.

The semantics of the two process layers are completely orthogonal: atomic supertasks are black-box "steps" in non-atomic supertasks. This means that changes can be made to one layer without affecting the other layer. There can be an arbitrary number of process levels in each of the two layers and each level can contain arbitrarily complex process structures. Consequently, application designers have a high level of freedom in structuring applications.

An example travel agency workflow process is shown in Figure 1 in a slightly adapted WIDE graphical workflow notation. In the figure, all boxes represent supertasks. Solid shadowed boxes represent business transactions, dotted boxes supertasks

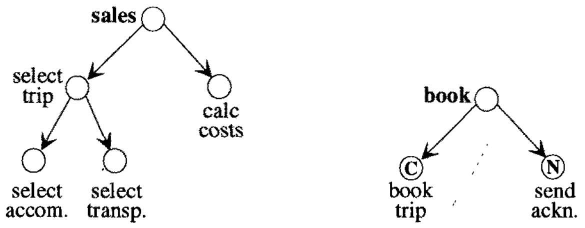

Figure 2: Example local transactions

above or below the level of business transactions. The ◊ symbol represents an or-split, the O symbol an and-split or and-join, and the ⦶ symbol an or-join.

4. A two-layer transaction model

In the WIDE project, the two-layer workflow process model described in the previous section is mapped onto a two-layer transaction model. In this transaction model, the upper layer is formed by *global transactions* providing the relaxed transactional semantics of process layers above business transactions and the lower level by *local transactions* providing the strict transactional semantics of business transactions. Below, we first discuss the local transaction layer, then the global transaction layer.

4.1 Local transactions

The local transaction layer of the WIDE model is used to support business transaction semantics in workflow processing. Business transactions require strict ACID transaction properties. They differ from traditional 'flat' ACID transactions, as supported by most commercial DBMSs, from the fact that they have a hierarchic structure consisting of subtransactions and basic actions. For this reason, we have chosen a nested transaction model for the local transaction layer in WIDE, partly based on nested transaction models (see e.g. [Da91]). The WIDE model provides flexible commit-dependency between subtransactions and their parents.

An example local transaction is the subprocess 'sales' from the example workflow, as depicted in Figure 2. This local transaction models selling a trip by selecting accommodation and transport details for a customer and providing the price tag for this selection. The local transaction consists of two subtransactions 'select trip' and 'calc costs'. The first subtransaction consists of two basic tasks; the second is a basic task by itself. Note that the control flow as shown in Figure 1 is not relevant for the local transaction concept, only the process hierarchy is taken into account.

In a local transaction, we can have critical and non-critical subtransactions. A critical subtransaction determines the success of its parent transaction: if the subtransaction aborts, its parent cannot commit. The success of a non-critical subtransaction does not affect the success of its parent. Figure 2 shows local transaction 'book' from the example workflow. In this transaction, subtransaction 'book trip' is critical, 'send ackn.' is noncritical, i.e., a failure in sending a booking acknowledgment does not abort the entire booking transaction, whereas a failure in the booking itself does.

Local WIDE transactions also provide a notion of intra-transaction concurrency control, used to obtain a mechanism for regulating data access between concurrent subtransactions of a local transaction. Intra-transaction concurrency control is per-

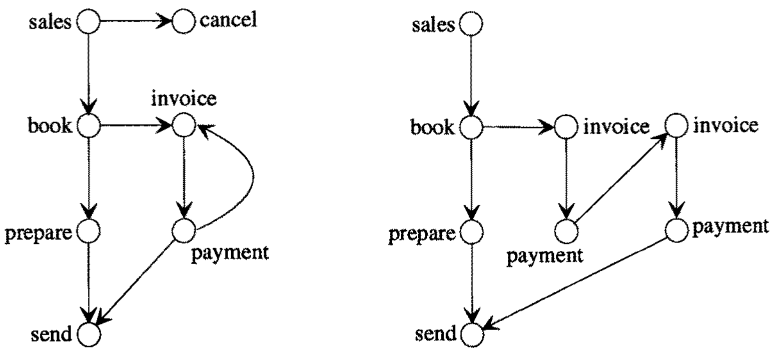

Figure 3: Global specification (left) and execution (right) graphs

formed on the granularity level of workflow data objects as defined in the WIDE information model [C96a], e.g. a workflow document or a folder containing multiple documents.

4.2 Global transactions

The global transaction layer of the WIDE transaction model requires relaxed notions of isolation and atomicity to cater for the needs of workflow processes above the business transaction level. Rollback on the global transaction layer should have application-specified semantics instead of the database-oriented semantics of the local transaction model. In this relaxed transactional context, local transactions must be black-box steps with respect to transactional semantics. For these reasons, we have chosen a global transaction model that is heavily based on the saga transaction model [Ga87], extended with a flexible mechanism for partial rollback.

A WIDE global transaction consists of a rooted directed graph of global transaction steps (local transactions). The graph is rooted as it can have only one starting step. It can have an arbitrary number of ending steps, and it can contain cycles. The graph represents the possible execution orders of the steps in the workflow process. The global transaction of the example workflow is shown in Figure 3. It is easily obtained by projecting the process structure in Figure 1 onto the business transaction structure.

Individual steps in the global transaction model conform to the ACID properties. Isolation in the global transaction, however, is relaxed with respect to the ACID model by making intermediate results in between steps visible to the context of the global transaction (i.e. steps commit their results to the shared database).

As we can have or-splits and cycles in a global transaction specification, the specification graph and the execution graph of a global transaction are different in general: paths that are not executed in an or-split are not in the execution graph and cycles are replaced by the instantiation of the iteration. Figure 3 shows an execution graph of the example specification graph. In this execution, the 'cancel' local transaction has not been executed and the 'invoice-payment' iteration has been executed twice. To reason about the dynamic properties of a global transaction in execution, the execution graph is considered, not the specification graph.

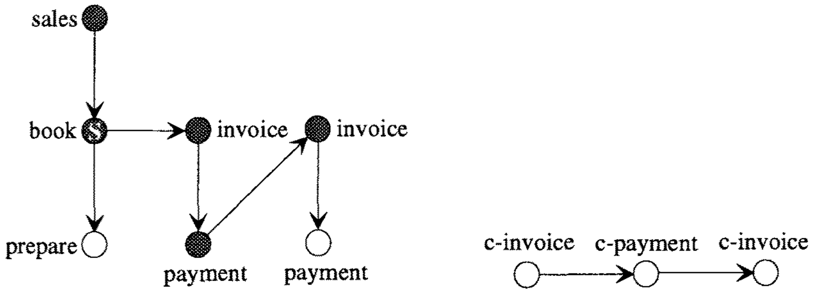

Figure 4: Partial execution graph (left) and compensating specification graph (right)

As in the saga model [Ga87], relaxed atomicity is obtained by using a compensation mechanism to provide rollback functionality. Rollback of global transactions is performed by executing compensating steps (local transactions) for the steps in the global transaction that have been committed (running, not-yet-committed steps can simply be aborted as they are atomic local transactions). Compensating steps are application-dependent and have to be specified by the application designer.

Complete rollback of a global transaction is often not desirable, as this may imply throwing away the results of a long workflow process. For this reason, we have introduced the notion of *savepoints* in global transactions. A savepoint is a step in a global transaction that is a safe place to begin forward recovery from. Unlike savepoints in the saga model [Ga87], global transaction savepoints do not require making checkpoints. Like the functionality of compensating steps, placement of savepoints in a global transaction is fully application-dependent.

An example of an execution requiring global rollback is shown in Figure 4. Here we see a partial execution of the specification graph in Figure 3. The grayed steps have been committed, two steps are being executed. Local transaction 'book' has been specified to be a savepoint. Now assume that running local transaction 'payment' raises an error that requires global rollback. Then all running local transactions are aborted (using the local transaction mechanism). Next, the execution graph needs to be compensated from the point where the error occurred until a savepoint is encountered (to the start of the graph if none is found). This means that compensation is performed by executing the dynamically constructed global transaction depicted in Figure 4. In this figure, the prefix 'c' for a local transaction indicates its compensating counterpart. Note that a very simple example is chosen for reasons of brevity. In general, compensating global transactions can have a complex structure.

5. A two-level transaction manager

The two-level transaction model outlined in the previous section is supported by a transaction manager architecture that is realized on top of the transaction service of a commercial DBMS. To provide portability, a high level of independence is required with respect to specific DBMSs. This implies making as few assumptions as possible about the transaction service of the underlying DBMS. To provide modularity in its construction and flexibility in its use, the overall transaction management architecture consists of independent global and local transaction management subarchitectures.

These two architectures are discussed on a functional level below. Their implementation in the context of the FORO WFMS is described in the next section.

5.1 Local transaction support

Local transaction support extends the basic 'flat' model of the underlying DBMS to the WIDE nested transaction model. As such, local transaction support can be seen as a transaction adapter as described in [Ba95]. A main difference between our situation and the situation described in [Ba95] is the fact that we deal with a closed DBMS architecture instead of a relatively open TP monitor architecture.

To be able to use the DBMS transaction management facilities in an effective and efficient way, each WIDE local transaction is mapped to a single DBMS transaction. A major issue in this mapping is the fact that possibly parallel subtransactions have to be mapped to a single sequential DBMS transaction. This results in database operations of multiple subtransactions being executed in an interleaved fashion.

Local transaction support handles abort of critical and non-critical subtransactions. If abort of a critical subtransaction leads to abort of the complete local transaction, this is easily performed by aborting the transaction on the DBMS level. If abort is limited to a subtransaction because a non-critical subtransaction is involved, partial rollback on the DBMS level is performed using the DBMS savepoint mechanism (distinguish these low-level DBMS savepoints from global transaction savepoints as discussed in Section 4.2). Interleaving of subtransactions as mentioned above complicates this situation, as it may lead to either the abortion of other subtransactions or the impossibility to abort a subtransaction.

Local transaction support provides simple mechanisms for intra-transaction isolation to allow the specification of parallel subtransactions operating on the same workflow data. The granularity of concurrency control is that of workflow objects, the scope is one local transaction.

5.2 Global transaction support

Global transaction support provides global transaction functionality as described in Section 4.2. Its main task is the construction of compensating global transactions when a global abort is requested, i.e. the construction of specification graphs containing compensating local transactions. Global transaction support uses local transactions in a black-box fashion, i.e. it sees global transaction steps the contents of which are completely irrelevant at the global transaction level. This ensures full independence from the underlying DBMS.

Global transaction support in WIDE bears some resemblance to the way sagas are supported in the Exotica project [Al96]. The main difference is the fact that Exotica is static (compile-time) in the construction of compensating structures, while WIDE is dynamic (enactment-time). WIDE allows cycles in process graphs, which requires analysis of the execution graph instead of the specification graph, implying dynamic compensation analysis. The introduction of global savepoints in the WIDE model provides additional flexibility in handling global aborts.

6. Implementation in the FORO architecture

In the WIDE project, the conceptual transaction management architecture outlined in the previous section is implemented on top of the Oracle DBMS and coupled to the FORO WFMS [Ce97] and an active rule management architecture [C96b]. The overall architecture is designed to be completely orthogonal with respect to transaction management, active rule management, workflow management, and data management. Distribution in the WIDE architecture is obtained through the use of a CORBA-compliant distributed object model [OM95], a client/server data management architecture, and a hierarchically distributed workflow server architecture [Ce97]. Independence from the underlying DBMS is obtained through an object/relation mapper, which maps object-oriented operations into relational primitives for the DBMS. The integration of extended transaction management with workflow and data management is shown in Figure 5. This figure clearly shows the independent subarchitectures.

6.1 Local transaction support

Local transaction management in WIDE is performed at a functional and a physical level to ensure maximum independence from the underlying DBMS. The functional level consists of a Local Transaction Manager (LTM) module and Local Transaction (LT) objects (see Figure 5). Each LT object is responsible for the functional management of a single local transaction. It manages the nested transaction structure and maps this to an abstract flat transaction model. The object is created dynamically by the LTM when the transaction starts. The LTM functions as a dispatcher of workflow events to the appropriate LT objects and of local transaction events to the workflow engine. Its main task is to keep both subarchitectures as independent as possible: through the use of the LTM, the workflow engine does not need to be aware of the existence of multiple LT objects. All transactional operations in the functional level are based on logical transaction identifiers. LTM and LT are fully independent from the underlying DBMS.

The physical level of local transaction support consists of the Local Transaction Interface (LTI) module. The LTI maps abstract flat transaction operations to the actual physical operations provided by the DBMS. Further, it maps logical transaction identifiers to physical transaction channels used for communication with the DBMS. With Oracle as database platform, the Oracle Call Interface (OCI) [Mc96] is used for this purpose [Gr97].

6.2 Global transaction support

The global transaction support (GTS) subarchitecture consists of a Global Transaction Manager (GTM) module and Global Transaction (GT) objects (see Figure 5). Each global transaction is managed by a GT object that is dynamically created at the start of the global transaction. The GT object is signaled by the workflow engine about events that may change the state of a global transaction, e.g. start and end of a global transaction step. The fact that a global transaction can have multiple active branches and iterative constructs implies that the process context of signaled events needs to be passed to the GT object as well. The main task of the GTM is to construct compensating global transactions as discussed in Section 5.2. The GTM is activated by the workflow engine when the engine raises a global rollback event. It uses the appropri-

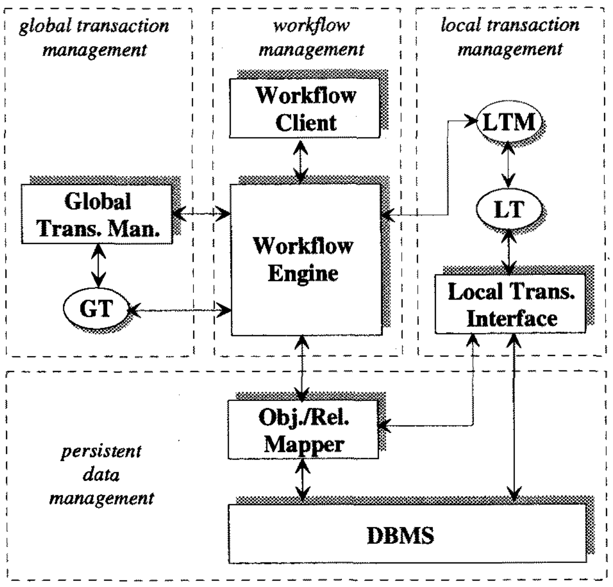

Figure 5: Integration in FORO architecture

ate GT object to obtain information on the current status of the global transaction, most notably the current execution graph and the compensating counterparts of executed steps. After the compensating global transaction has been constructed, it is passed to the GT object, which makes the information persistent.

Because a single GTM can serve multiple workflow engines at possibly remote sites, both GTM and GT objects are implemented as CORBA objects [OM95]. In this distributed objects approach, placement and clustering of processes can be handled transparently. The fact that the GTM routes all data access through the appropriate GT objects enables independent allocation of GTM processes and DBMS.

7. Conclusions

In designing a transaction model for workflow management, one is confronted with conflicting requirements. On the one hand, most transaction models are too heavily database-oriented to be non-restrictive to process requirements [Al96]. On the other hand, reliable data processing as obtained by the use of database-oriented transaction models is required in business applications. We have addressed this problem with a process-oriented upper layer providing flexibility towards process management and a database-oriented lower layer providing reliability towards data management. The 'interface-level' between the two layers can be chosen freely on an application-dependent basis.

Although the WIDE transaction support is presented in the context of workflow management, it is certainly not limited to this purpose. The transaction management architecture can easily be used in other environments where complex process support is important. The orthogonality of global and local transaction support allows the modification of one layer without affecting the other layer, or even omission of one layer if so required.

Most of the transaction support presented in this paper is at the time of writing this paper being implemented in the WIDE Version 1 system (except for the intra-transaction local concurrency control). After completion, it will undergo a thorough functional test at the end user sites in the WIDE consortium. In the WIDE Version 2 system, we plan to add support for distributed global transactions, handling of asynchronous global transaction aborts, a persistent local transaction mechanism and intra-transaction concurrency control for local transactions.

Acknowledgments

All members of the WIDE project are acknowledged for their contributions to the work presented in this paper. Special thanks go to Stefano Ceri of the Politecnico di Milano and Gabriel Sánchez of Sema Group at Madrid.

References

[Al96] G. Alonso et al.; *Advanced Transaction Models in Workflow Contexts*; Procs. Int. Conf. on Data Eng., 1996.

[Ba95] R. Barga, C. Pu; *A Practical and Modular Method to Implement Extended Transaction Models*; Procs. 21st Int. Conf. on Very Large Data Bases, 1995.

[C96a] F. Casati et al.; *WIDE: Workflow Model and Architecture*; CTIT TR 96-19; Univ. of Twente, 1996.

[C96b] F. Casati et al.; *Deriving Active Rules for Workflow Enactment*; Int. Conf. on Database and Expert System Appls.; Zürich, Switzerland, 1996.

[Ce97] S. Ceri, P. Grefen, G. Sánchez; *WIDE - A Distributed Architecture for Workflow Management*; Procs. 7th Int. Worksh. on Research Issues in Data Eng., 1997.

[Ch94] P.K. Chrysanthis, K. Ramamritham; *Synthesis of Extended Transaction Models using ACTA*; ACM Trans. on Database Systems, 19-3, 1994.

[Da91] U. Dayal, M. Hsu, R. Ladin; *A Transactional Model for Long-Running Activities*; Procs. 17th Int. Conf. on Very Large Databases, 1991.

[El92] A.K. Elmagarmid (Ed.); *Database Transaction Models for Advanced Applications*; Morgan Kaufmann; USA, 1992.

[Ga87] H. Garcia-Molina, K. Salem; *Sagas*; Procs. 1987 ACM SIGMOD Int. Conf. on Management of Data; USA, 1987.

[Gr97] P. Grefen, J. Vonk, E. Boertjes, P. Apers; *Two-Layer Transaction Management for Workflow Management Applications;* CTIT TR 97-07; Univ. of Twente, 1997.

[Lo93] D. Lomet (Ed.); *Special Issue on Workflow and Extended Transaction Systems*; IEEE Data Eng. Bull., June 1993.

[Mc96] D. McClanahan; *Oracle Developer's Guide*; Osborne McGraw-Hill, USA, 1996.

[OM95] Object Management Group; *The Common Object Request Broker: Architecture and Specification, Version 2.0*; Object Management Group, 1995.

[Re95] A. Reuter, F. Schwenkreis; *ConTracts - A Low-Level Mechanism for Building General-Purpose Workflow Management Systems*; IEEE Data Eng. Bull., 18-1, 1995.