A Practitioners Approach To Data Federation

Frank Leymann
IBM Software Group
Schönaicher Str. 220
D-71032 Böblingen
Germany
e-mail: ley1@de.ibm.com

Abstract: *During the last few years message queuing and workflow systems have been established as major elements of the software stack. In practice, this middleware is often used to cope with aspects of data federation: Message queuing technology is exploited for application integration. Workflows extend the notion of stored procedures in a federated environment and provide transaction management as well as monitoring features for inter-transaction integrity in such an environment.*

©-Notice: Many of the figures in this paper are taken from [LR] and are copyrighted by Prentice-Hall.

1 Introduction

In practice, the main purpose of dealing with federation is the "seamless" development of application systems that need to access multiple data sources to support a specific business function. This has two different major aspects: First, the integration of applications, and second, support for data warehousing. The basic requirement resulting from these aspects is the enablement of sharing data that are maintained via multiple heterogeneous data stores (DBMS, file systems etc.), or that originate from multiple data sources, respectively. The corresponding problems to solve are:

1. Masking heterogeneity of sources, i.e heterogeneity in terms of their schemes and CRUD methods. This facilitates the development of new application logic by easing the corresponding data manipulations.

2. Providing suitable LUW mechanisms. This facilitates application recovery.

3. Providing joint media recovery (something that is mostly open until today).

Different communities have different foci when attacking the problems of data federation: The research community wants to build an ideal world, e.g. an integrated schema of the federated databases, correctness criteria for federated transactions based on some variant of serializability etc. (e.g. [A], [BGS], [Li], [LMR], [SL]). Vendors of database systems want to expand the market space for their engines by adding "some selective" features, e.g. object-relational capabilities to extend type systems, support of the XA protocol to allow participation in global transactions, global optimization technology etc. (e.g. [MKRZ]).

Enterprise IT staff want to share data between applications (both, existing as well as newly written and even standard applications - see figure 1), i.e. no invasive actions on data or programs are admissible, no new database system must be required etc. (e.g. [S]).

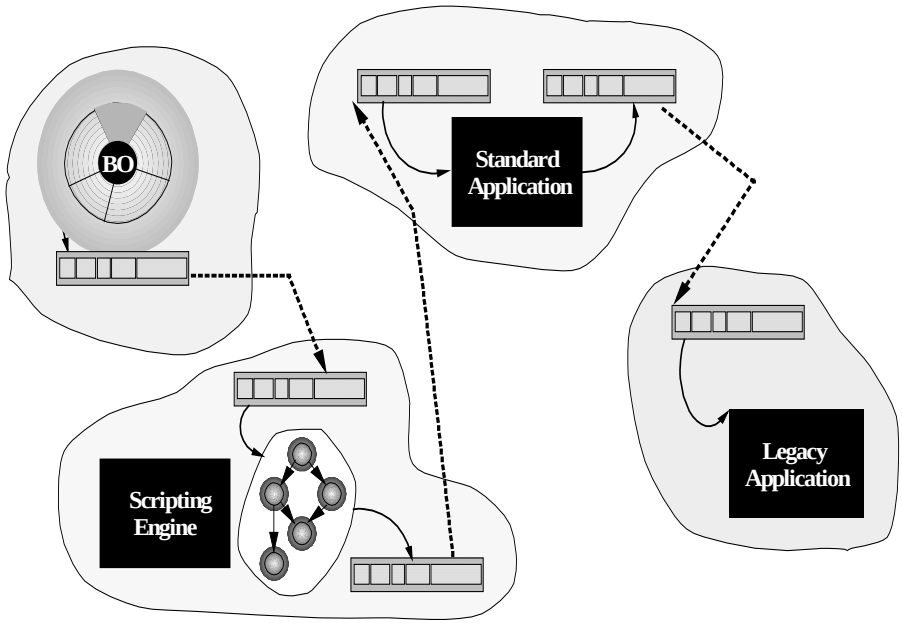

Figure 1: The Enterprise Application Integration Problem Area

Nowadays, the latter aspect is usually referred to as **E**nterprise **A**pplication **I**ntegration (EAI) or simply as application integration. It is the latter focus that we will deal with in this paper. The state-of-the-art in application integration within enterprises can be summarized as follows:

- No “pure/ideal” federated database system as envisioned by the research community is on the market, and consequently such systems don’t play any role in practice.

- Federated extensions of existing database systems are in use but often restricted to special purposes, e.g. for integrating relational databases with non-structured data like text, spatial, image, or for warehousing by providing a single source for extracting data from multiple databases.

- The use of message middleware for linking and bridging applications is widely accepted (the corresponding market is large and rapidly growing). Such middleware provides a means for asynchronous application-to-application communication via reliable message queuing, i.e. with explicit destination (i.e. a queue). Also, message brokering is more and more supported facilitating complex message transformations and a publish/subscribe paradigm (i.e. “multi-casting” of messages). Finally, complete “sequences” (i.e. partial orders) of applications are scripted together via workflows (both, transactional workflows and people-driven workflows).

The paper is organized as follows: In section 2 we sketch the fundamentals of message queuing middleware. Next, we outline the meaning of message brokering in information

federation. Section 4 sketches some relations between database systems and the publish/subscribe concept. Based on our application integration focus we argue in section 5 that application functions can be considered as “atoms” of data manipulations in federated environments and that workflows consequently correspond to stored procedures in these environments. Section 6 discusses the usage of workflow systems for managing federated transactions, and in section 7 we show how workflow systems can be used to ensure inter-transaction integrity.

2 Message Queuing

Since decades large enterprises built applications based on message queuing technology (see [BHL]). With the advent of generic message queuing systems like IBM MQSeries this technology became pervasive. Message queuing allows programs to reliably communicate in an asynchronous manner. The main benefit of asynchronous communication is that the communication partners are not required to be up and running at the same time. When combined with ensured delivery and message integrity this contributes to the overall availability of the implemented services (see below as well as [MD] and [LR]).

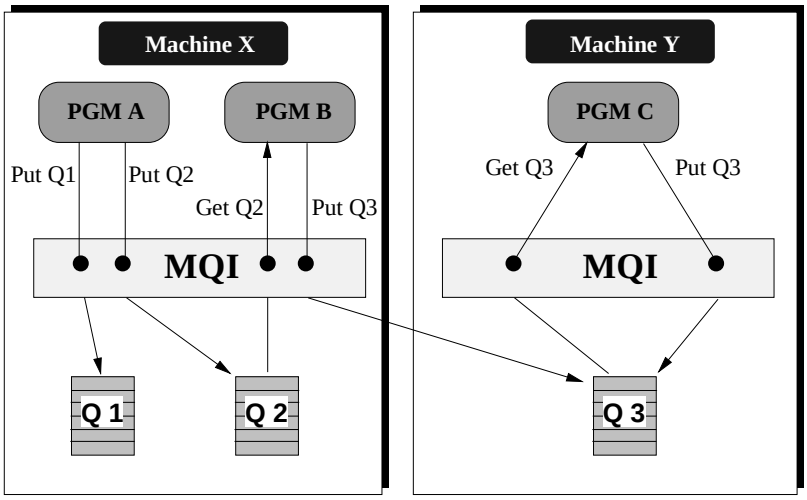

Figure 2: Message Queuing Interface

The interface a message queuing systems provides to manipulate messages in queues is often called the message queuing interface (MQI). Via the MQI application programs can put messages into local or remote queues, and they can get messages from local queues (see figure 2). Typically, the program that gets a message from a queue is different from the program that put the message into the queue. A message queuing system often has a client/server structure providing the MQI as client to a message queue manager (see figure 3). This allows an application to run on a machine separate from the message queue manager.

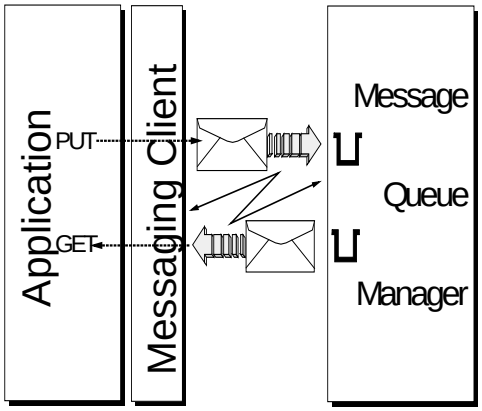

Figure 3: Client/Server Structure Of The MQI

The message queue manager facilitates all services to manipulate messages as well as to manage queues themselves. It hardens persistent messages and ensures recoverability even when processing transient messages. Furthermore, it guarantees delivery of messages even if the target queue is managed by a remote queue manager. For this purpose messages are moved between two message queue managers based on a protocol similar to the two phase commit protocol. Also, the message queue manager hides all idiosyncrasies of the underlying network infrastructure from the program that makes use of the MQI (see figure 4).

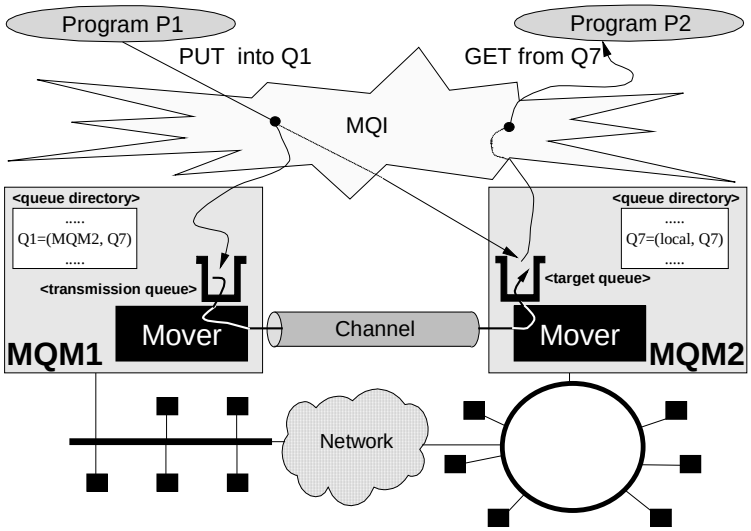

Figure 4: Plumbing Underlying Message Delivery

Via the MQI messages can be manipulated within transactions: A mixture of put and get requests destined to any collection of local or remote queues can be processed in an ACID unit by simply bracketing the requests by `BEGIN` and `COMMIT` or `ABORT` verbs, respectively (see figure 5).

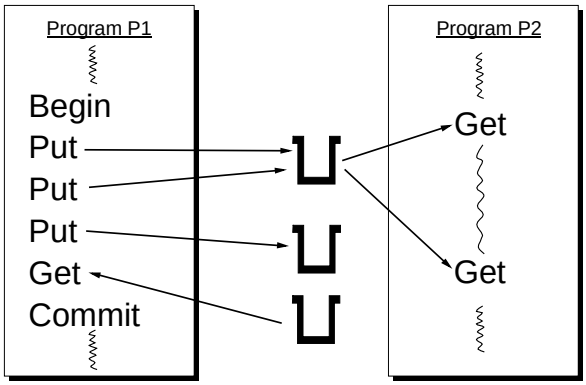

Figure 5: Message Queuing And Transaction Brackets

A message queue manager is also a resource manager that participates in a two phase commit protocol coordinated by a transaction manager. This allows to mix MQI requests with requests to other resource managers (like SQL calls) in a single transaction. In doing so, *message integrity* can be ensured (see figure 6): Assume a program (referred to as "server" in figure 6) that provides services to its clients. A client wants to be sure that its service request is processed by the server exactly once. For this purpose the client puts a persistent request message into the server's input queue within a transaction. The server gets a request message out of this queue, processes it (e.g. by manipulating data in a database) and puts its (persistent) response into the invoking client's response queue, all of this within a single two phase commit protected transaction; i.e. if, for example, the database manipulations fail the response message will never be delivered and the original request message is restored in the server's input queue and can be processed again. Finally the invoking client will receive its response which it might process in a separate transaction.

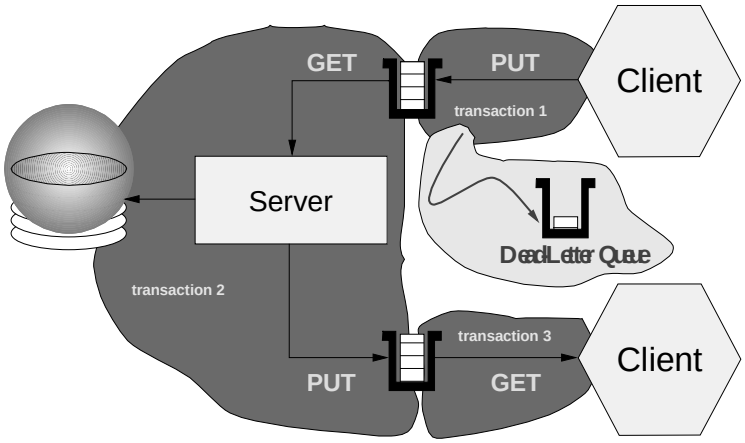

Figure 6: Message Integrity

Note, that in practice this processing is a bit more subtle: Although the message queuing system guarantees to never loose a message the reception of which it acknowledged it might happen that it cannot deliver the message to the specified target queue. For example, the target queue might not exist, it might be full etc.. Also, the message might be "poisoned", i.e. the server might always end abnormally when trying to process the message. In these and

similar cases the subject message is delivered into a so-called “dead letter queue” from which it can be processed by a special program (e.g. the message can be analyzed, perhaps corrected, and send again). In this sense, message integrity does not ensure exactly once processing but at most once processing (e.g. if the message cannot be corrected) with precise exception semantics.

Figure 7 depicts the general structure of a message. The message header contains information like the queue to which the server (see above) has to reply its response. The message body contains the proper information of the message. Typically, the body itself is structured into two parts, the type of the request to be served by the receiving server and the data needed to process the request proper.

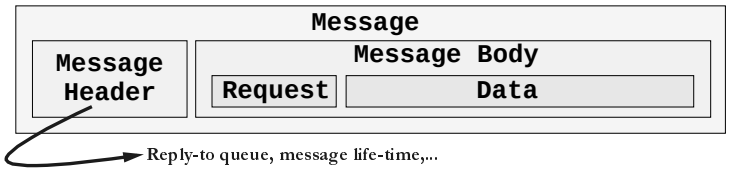

Figure 7: Message Structure

While the message header is defined by each message queuing system implementation the syntax and semantics of the message body is application specific. For example, standard application vendors define messages that allow to kickoff functions of their systems from the outside. Furthermore, standard consortia define messages for electronic data interchange (EDI). Recently, XML (extensible markup language) is more and more used to define messages.

Often, messages have no request part defined, i.e. only data is transferred via the corresponding messages. In doing so, information is disseminated between a collection of applications. This allows to separate “data sharing” aspects of federation and “data access” aspects: Via messaging, applications can share data content without sharing the same access mechanism, i.e. the interface to physically access the data in a specific store. This way messaging enables federation via exchanging data between disparate or unlike applications.

3 Data Federation Via Message Brokering

Within message queuing systems the destination of a message has to be defined explicitly. Furthermore, the assumption is that the sender and receiver of a message agreed on the same structure of the message (body). More recently, the concept of a *message broker* has been introduced (see [S]) which proposes to disseminate messages without specifying their targets and without assuming the identical structure of the message sent and the message received. To achieve this, a message broker allows to define rules for routing messages based on their content in multiple steps to various recipients and to define the format of the message a given recipient wants to receive the message in (see figure 8).

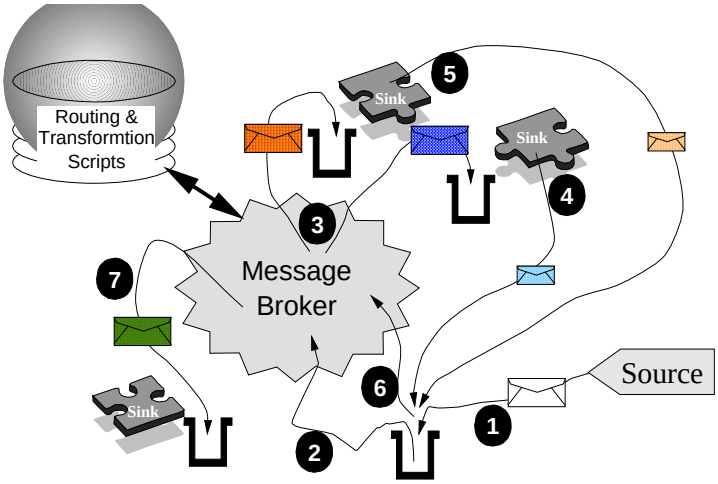

Figure 8: Message Broker Scenario

Sources of messages are referred to as *publishers* while sinks or recipients of messages are called *subscribers*. When a publisher submits a message to the broker, the broker determines the recipients based on their subscriptions. Basically, a *subscription* represents the interest of a recipient in a particular message based on the original content of the published message and defines the format and content of the message the recipient wants to receive. For example, the content of the message to be received might be different from the original content because a subscriber wants to have the message annotated with other data or wants it to be cleansed, and the format might be different because the publisher passed an XML message to the broker while the subscriber want to receive the message as a specific in-memory data structure (e.g. the COMMAREA of a CICS program). Because of this, message transformations might become quite complicated multi-step scripts. The message formats and associated transformation rules/functions and scripts are stored together with the subscriptions in the message dictionary of the broker. The broker might also manage a message warehouse to hold messages for replay or analysis (see figure 9).

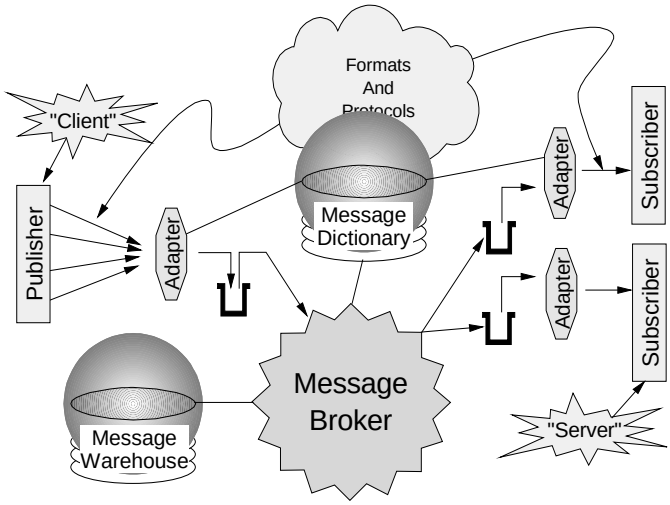

Figure 9: Message Broker Structure

The sketched publish/subscribe paradigm can be used to realize enterprise application integration, i.e. the aspect of federation we are focussing on: A message that gets published might represent data that must be reflected in various formats in other databases, or certain applications must be run based on this message to ensure that the publishing application's processing maintains integrity of the overall environment. The corresponding database systems or applications are subscribers in this scenario.

To a certain degree a publisher can be perceived as a client and the subscribers as servers processing the request of the client. The message broker then becomes a piece of middleware for dynamic integration of heterogeneous applications. It facilitates frequent changes of the applications to be integrated. The exchange and transformation of messages mediated by the message broker can be perceived as a continuous process of schema integration, i.e. the global external schema is maintained "incrementally". In contrast to this, the creation of a global external schema is traditionally seen as a one time shot or at least a process that is performed very infrequently [SL].

4 Publish/Subscribe And Database Systems

As a special case, database systems might become publishers as well as subscribers in message broker environments [FLRS]. As a result, databases might be kept in sync, integrity might be maintained, etc. based on this technique. Thus, a message broker provides plumbing that is beneficial for data federation.

Figure 10 depicts an example [BCM+]: On the left side of the figure various publishers send information about stocks. The information provided differs from publisher to publisher. On the right side various subscribes request to receive the published messages in different formats. In addition, the first subscriber requests to annotate published stock quotes by data stored in a database, while the second subscriber specifies a filter to receive high-value trades only. The message broker performs the annotation as part of the corresponding message transformation and it performs the filtering as specified as part of a subscription. Published messages can be stored in the message warehouse of the broker for future analysis via OLAP functions or mining tools, for example.

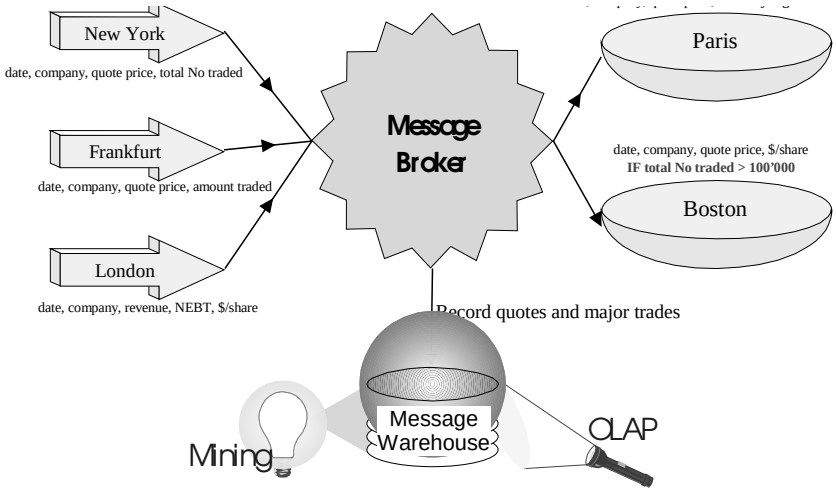

Figure 10: Information Brokering As Pubsub Scenario

The message broker might be a standalone system to which messages are published and from which subscriptions are requested. But in case data federation is emphasized adding message broker functionality to the participating database systems themselves does make a lot of sense.
Figure 11 shows on its left side a database system that can publish messages based on modifications of operational data. Based on subscriptions defined to the database system it filters data of interest to the various subscribers, transforms it into messages as requested in the various subscriptions and sends it to the message queues specified in the subscriptions. On the right hand side the figure shows a database system that has subscribed to receive certain "messages" (e.g. messages that are send by the other database system's publishing features). It receives the messages from its associated input queue and inserts it into the corresponding database.

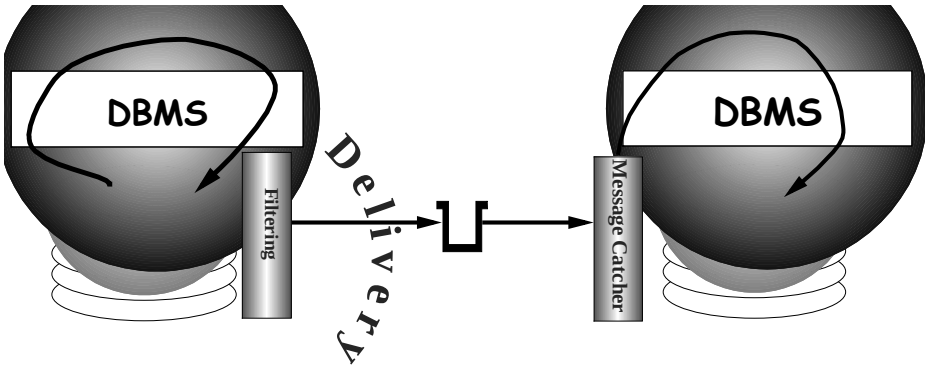

Figure 11: DBMS As Publisher and Subscriber

5 Federated Stored Procedures

Many applications of workflow systems can be subsumed under the term *complex request*. Figure 12 shows an example of a complex request: Based on an incoming message (from a queue or a terminal etc.) a workflow is started that first transforms the message into a format expected by programs that perform data modifications. These programs are then invoked by passing the expected messages to them. When the programs terminate another action is kicked-off, e.g. another workflow that performs more complicated actions.

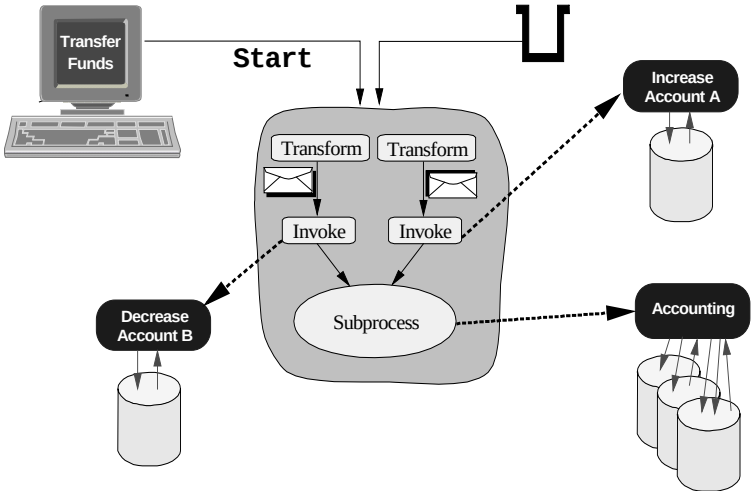

Figure 12: A Complex Request

The generic situation behind complex requests is as follows: Based on an incoming message (e.g. published by a DBMS!) several data stores must be manipulated to achieve a new consistent state in a collection of databases. For this purpose a whole pool of functions (application programs, transactions,...) is already available each of which manipulate a separate data store. A complex request is a script that ties together some of these functions into a unit that manipulates the complete collection of affected databases (see figure 13).

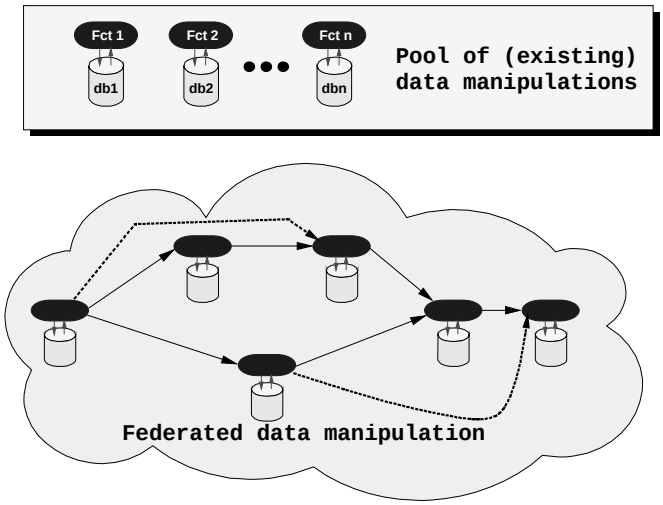

Figure 13: Scripting Existing Data Manipulations

To a certain degree, today's stored procedures as provided by most database systems are complex requests that tie together manipulations of different data stores (see figure 14): Instead of invoking a series of manipulation functions (i.e. SQL statements) each of which manipulate a certain data store (i.e. table) an application invokes a single script (i.e. stored procedure) that manipulates the data stores in a unit of work.

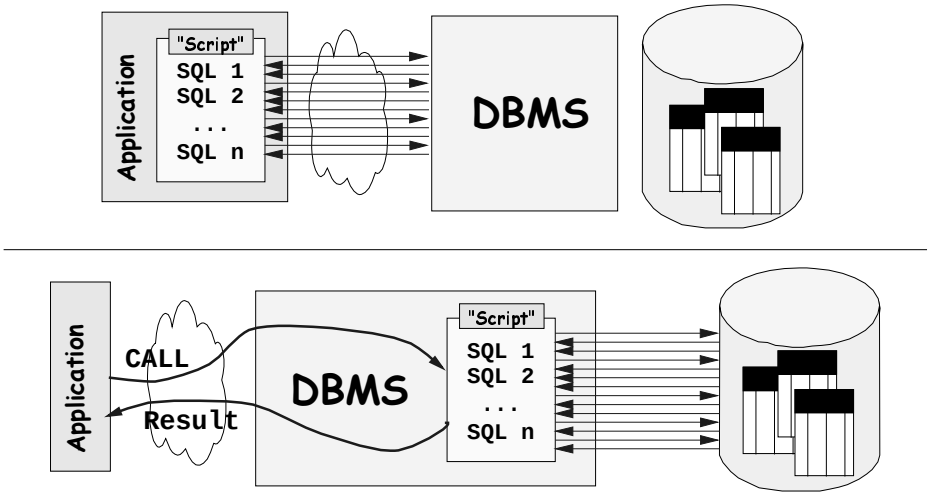

Figure 14: Stored Procedures As Data Manipulation Scripts

In federated environments complex requests must very often have additional properties: A complex request must have characteristics of some kind of extended transaction, i.e. it must be a “real” unit of work (cf. section 6). The control flow is an arbitrary DAG to allow for parallel manipulations of certain databases (speedup!), the data flow between the scripted programs may not match their control flow (allowing to pass data to a program that does not run immediately after the program that produced the data), the script must be statefull (to provide forward recovery), and certain exceptional situations must be handled via people involvement. Together, complex requests pass the boarder to production workflows (see [LR]). Thus, it is only natural to consider workflows as the federated equivalent to today’s stored procedures. Furthermore, as stored procedures are managed and run by the database system itself a federated database system should be extended to provide functionality of a workflow system (see figure 15). This will also allow to perceive federated database systems as scalable and robust runtime environments for data manipulation intensive application functions similar to what database systems provide wrt stored procedures already today (“TP-lite”).

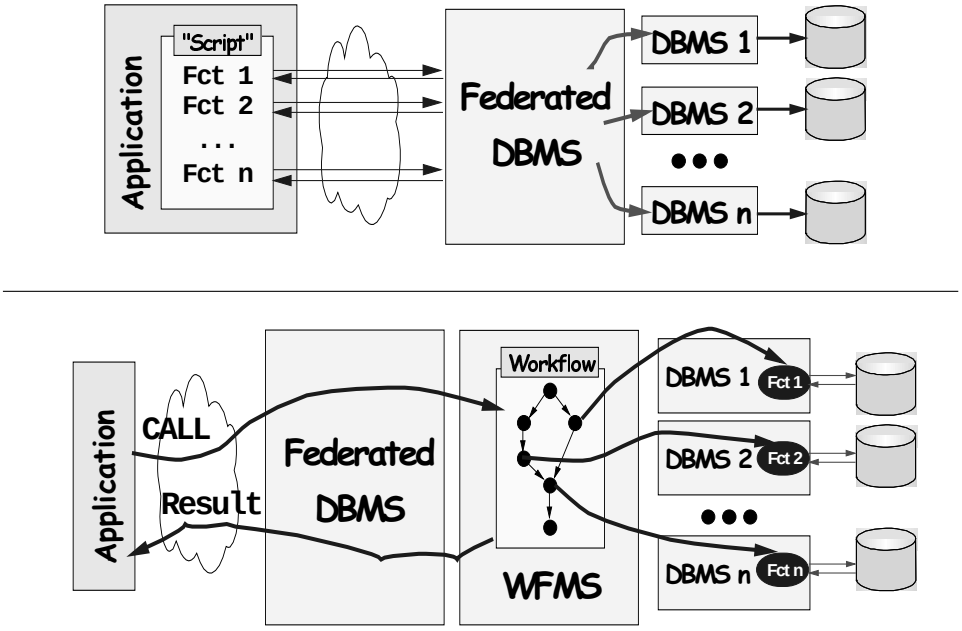

Figure 15: Complex Requests As “Federated Stored Procedure” Equivalent

6 Transaction Management In Federated Database Systems

[A] proposes to use nested transactions and multi-level transactions as transaction model suitable for manipulating federated databases. [BGS] argue that global serializability is unrealistic in federated environments and discuss several (weaker) correctness criteria; furthermore, they already suspect that workflow technology might be useful in managing federated transactions. [SST] and [SSA] suggest to use a certain combination of workflow technology and advanced transaction technology for realizing federated transactions in composite systems. We argue in this section that atomic spheres and compensation spheres in workflows as introduced in [L1] and [L2], respectively, can cover a broad spectrum of

federated transactions. Since all of the material is extensively covered in [LR] we will be even more succinct than in the preceding sections.

An atomic sphere [L2] is a collection of activities within a workflow (like {B,C} in figure 16) each of which is implemented by a classical (ACID) transaction and which must collectively be run as an atomic unit of work. Basically, the workflow system has to make sure that the atomic sphere is like a global transaction running the encompassed transactions as its subtransactions.

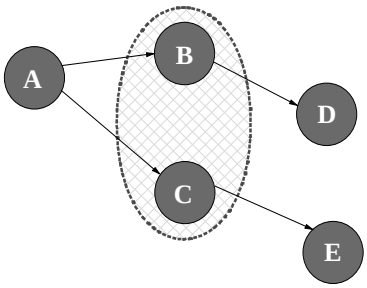

Figure 16: Atomic Sphere

Figure 17 sketches how the workflow system can achieve this: When the workflow system detects that it enters the atomic sphere for the first time (i.e. ❶) it begins a global transaction with the transaction manager of the environment. Next, within the context of this global transaction (i.e. ❷) it invokes the transactions included in the atomic sphere determined to be executed via navigation through the workflow. When the workflow system detects that it can leave the atomic sphere it registers itself as a participant of this global transaction and requests the transaction manager to commit the global transaction representing the atomic sphere (i.e. ❸). When the global transaction has been committed it leaves the atomic sphere and continues regular navigation (i.e. ❹), otherwise it tries to execute the atomic sphere again.

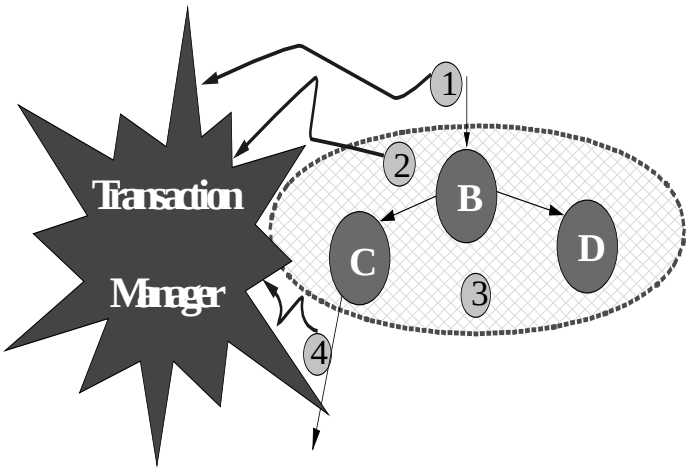

Figure 17: Processing Atomic Spheres

In some scenarios, atomic spheres (i.e. global transactions based on workflows) may not satisfy the requirements of a federated environment: For example, locks may be held for too long reducing the overall throughput, or transactions have to be mixed with non-transactional

programs into something that is worth being called a unit of work. Compensation spheres [L1] can often be used in these situations: A collection of activities within a workflow (see figure 18) can be aborted by running associated compensation actions. For this purpose, each single activity of the compensation sphere as well as the compensation sphere itself is associated with its appropriate compensation action. When aborting a compensation sphere either the compensation action of the compensation sphere is run, or the compensation actions of the encompassed activities are executed.

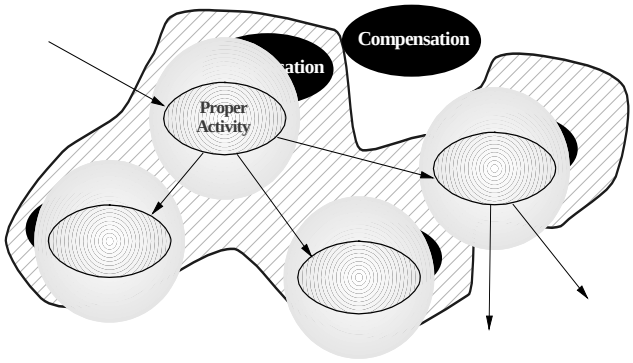

Figure 18: Compensation Sphere

Figure 19 shows how the latter mode of aborting a compensation sphere is performed: The workflow system will simply construct for the sphere S a workflow P(S) by reversing the edges of the subgraph induced by the sphere S in the original workflow and execute the compensation actions instead of the actions proper. Once this “repair workflow” has been run, the affected parts of the original workflow can be performed again.

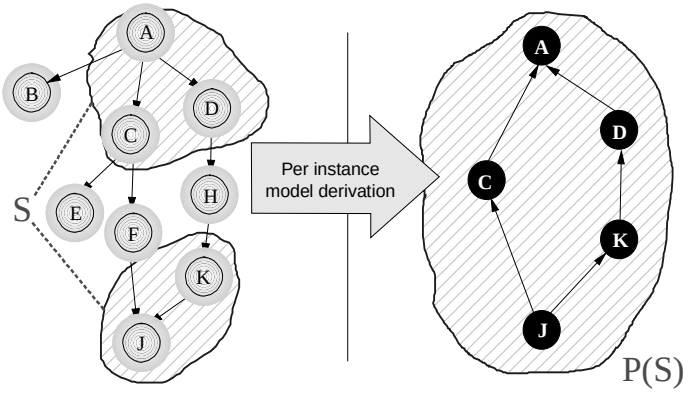

Figure 19: Rolling Back Compensation Spheres

Of importance similar to the sketched backward recovery features of workflows is forward recovery of workflows (cf. [L3]). By maintaining persistent state information about the navigation through the workflow as well as the execution state of each activity forward recovery of the workflow itself is achieved: Whenever a crash situation occurs the workflow system can consult its state database after restart to figure out the processing state of each workflow in the system. Thus, no work that has already be successfully run has to be redone and navigation continues where it has been left (see figure 20). As a result, each workflow

itself is forward recoverable (but without including user transactions into the scope of forward recovery!).

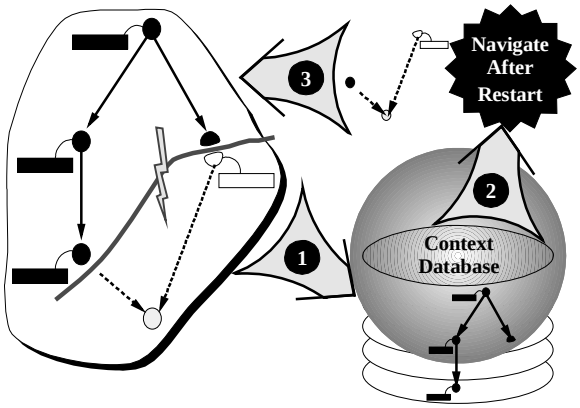

Figure 20: Forward Recovery Of Workflows

In order to ensure forward recovery of the resulting application the workflow system must be able to include the user transactions themselves into its owns internal transaction processing (cf. [L3]). This will appropriately cope with situations in which a user transaction has been active at the time the erroneous situation took place. Figure 21 shows how this can be realized in an effective manner: Instead of running a distributed transaction encompassing the various components of the workflow system as well as the user transaction, global transactions are only run based on collocation (avoiding network latency,...) and the global transactions are chained via persistent messaging. As result, the overall workflow application is forward recoverable.

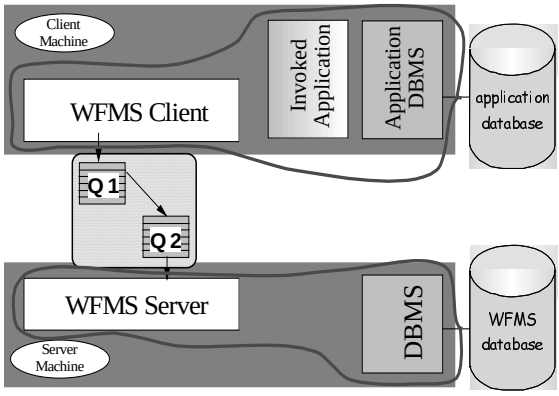

Figure 21: Combining User Transactions And System Transactions

In summary, workflow systems can ensure both, forward recovery as well as backward recovery of workflows and the resulting applications. When workflows are used to modify data in a federated database system by orchestrating collections of functions each of which manipulate data in a particular database system underlying the federated system a broad spectrum of federated transactions is covered. For example, an application of a federated database system can kickoff its required data manipulations as a workflow which is run by a workflow system belonging to the environment (see figure 22). The workflow system will invoke the corresponding manipulation functions. In case no errors occur the workflow is

"committed" otherwise it or even selective parts of it can be aborted by rolling back appropriate compensation spheres. Note, that complete workflows can be atomic spheres or compensation spheres in case finer granularity of backward recovery is not required.

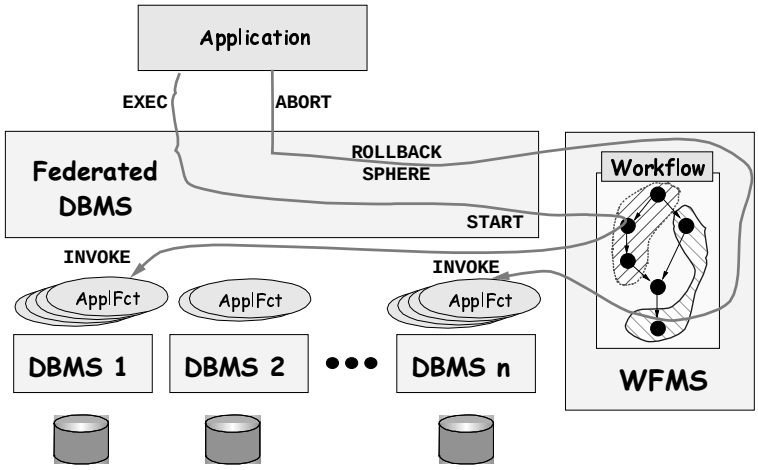

Figure 22: Managing Transactions In Federated Environments

7 Inter-Transaction Integrity

Semantic integrity constraints can be classified as static, transitional, or dynamic constraints (see figure 23). A *static* constraint describes the validity of each single database state: For example, the salary of a manager must always be higher than that of all of his employees. A *transitional* constraint governs the validity of the transition of two consecutive database states: For example, salaries can only increase. Static and transitional constraints together specify all possible series of valid database states; integrity monitors of today's database systems together with transaction technology cover a broad spectrum of static and transitional integrity constraints.

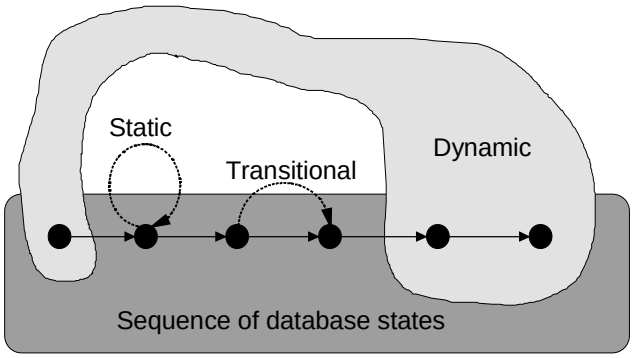

Figure 23: Categories Of Semantic Integrity Constraints

But not every possible series of database states is admissible: For example, the salary of an employee must only be increased if the employee got promoted (sometimes) before. *Dynamic* integrity constraints specify the validity of transitions between a collection of not necessarily consecutive database states out of a possible series of valid database states. To specify

dynamic integrity constraints temporal logic has been proposed more than a decade ago, algorithms to transform temporal logic expressions into something more tractable have been derived, and monitors based on these algorithms have been suggested (see [Li], for example). But no commercial system supports these kind of constraints today.

Transactions are used to manipulate database states and to perform transitions from one valid state to another. Thus, specifying when (in terms of context) a particular transaction may be executed means to specify when a particular database state transition may take place, which is a dynamic integrity constraint - in this context the term *inter-transaction integrity* constraint is more appropriate. Workflows define when transactions (more precisely: implementations of activities) may be executed, i.e. a workflow may be perceived as an inter-transaction integrity constraint. Figure 24 shows an example of such a constraint allowing the `promote` transaction only to be run if the `increase job code` transaction has been run sometimes before, if the `quarter result` of the employee was acceptable, and if a witness report has been requested.

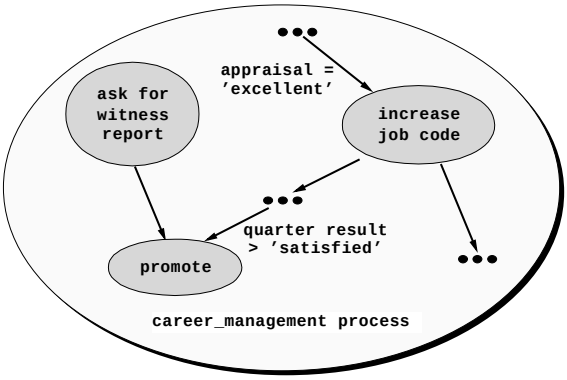

Figure 24: A Sample Inter-Transaction Integrity Constraint

Thus, in an environment in which transactions are exclusively invoked via a workflow system (called *clean* in figure 25) a broad spectrum of dynamic integrity constraints are automatically monitored. Note, that standard applications or custom built workflow applications provide environments that are clean. Thus, workflow systems are monitoring dynamic integrity constraints in certain environments already today.

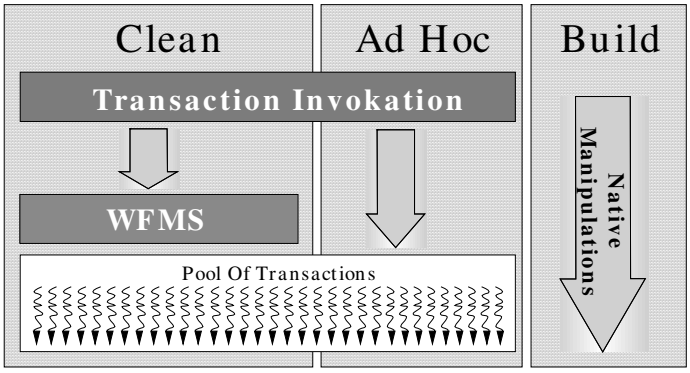

Figure 25: Categories Of Environments For Starting Transactions

What can be done in environments in which transactions are invoked outside the control of a workflow system (called *ad hoc* in figure 25)? In these situations a workflow system can be used as "admissibility checker" that is consulted by a database system (see figure 26). This assumes that each transaction that is restricted by a dynamic integrity constraint corresponds to an activity in a workflow that represents the associated constraint. Whenever a transaction is about to be started by the database system the workflow system is called to determine whether the subject transaction is admissible; if it is the transaction is actually executed otherwise it is rejected. When the transaction ends the database system informs the workflow system about the actual outcome of the transaction so that the state of the activity corresponding to the transaction can be updated in the workflow accordingly. (The interested reader is referred to [LR] for more details about the underlying mechanisms.)

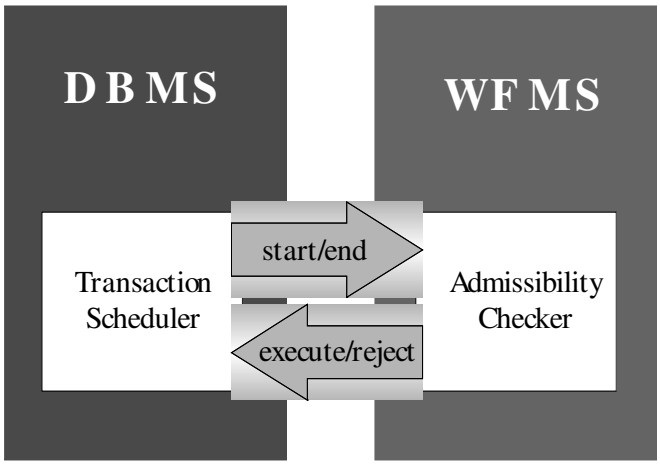

Figure 26: Handshaking For Monitoring Inter-Transaction Integrity Constraints

This shows that a workflow system may act as a monitoring component for inter-transaction integrity constraints in database systems, be it a stand-alone or a federated database system.

8 Conclusion And Outlook

In practice, enterprise application integration (EAI) is one of the predominant foci of information technology personnel today and a major reason for their interest in associated aspects of data federation. Message oriented middleware - i.e. message queuing, message transformation technology, and publish/subscribe mechanisms - are widely exploited today for reliable and flexible data exchange between applications: Schema integration is a continuous process in this environment. This way, data content is shared in a robust manner without sharing access mechanisms of data stores. Even database systems may become participants in such an environment.

> In order to maintain integrity of databases a message often must be processed by a whole series of applications which may even depend on the message content and other parameters: Today, such complex requests are realized in practice via workflows. The corresponding workflows are the equivalent of today's stored procedures in federated environments. Thus, a workflow system is a valuable component in such an

environment. Because workflow systems may also provide both, some kind of extended transaction features (i.e. workflow specific unit of work concepts) and monitoring of inter-transaction integrity constraints adding a workflow system as a component to federated environments seems to be imperative.

Many subjects presented in this paper deserve more investigations; to name just a few: The impact on architectures and systems structures resulting from adding workflow functionality to federated database systems. The role of database systems as publishers or subscribers, as well as efficient implementations of publish/subscribe mechanisms within database systems. Efficient identification of transactions governed by inter-transaction integrity constraints within database systems as well as the extension of using workflow technology for monitoring such constraints in "build" environments (see figure 25) where transactions are composed on the flight by power users using native manipulation functions.

9 References

[A] G. Attaluri, *Issues in managing long transactions and large objects in a multidatabase system*, Proc. CASCON'92 (Toronto, Canada, November 1992).

[BHL] B. Blakeley, H. Harris, R. Lewis, *Messaging and queuing using the MQI* (McGraw-Hill, 1995).

[BGS] Y. Breitbart, H. Garcia-Molina, A. Silverschatz, *Overview of multidatabase transaction management*, Proc. CASCON'92 (Toronto, Canada, November 1992).

[BCM+] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, D.C. Sturman, *An efficient multicast protocol for content-based publish-subscribe systems*, Proc. ICDCS'99.

[FLRS] J.-Ch. Freytag, F. Leymann, D. Roller, M. Stillger, *Publish/subscribe functions based on object-relational features*, in preparation.

[L1] F. Leymann, *Supporting business transactions via partial backward recovery in workflow management systems*, Proc. BTW'95 (Dresden, Germany, 3/22-24, 1995).

[L2] F. Leymann, *Workflows make objects really useful*, Proc. 6th Intl. Workshop on High Performance Transaction Systems (Asilomar, CA, 1995).

[L3] F. Leymann, *Transaction support for workflows* [in German], Informatik Forschung & Entwicklung 12 (1997) 82-90.

[Li] U.W. Lipeck, *Dynamic integrity of databases* [in German] (Springer, 1989).

[LR] F. Leymann, D. Roller, *Production workflow: concepts and techniques* (Prentice Hall PTR, 2000).

[LMR] W. Litwin, L. Mark, N. Roussopoulos, *Interoperability of multiple autonomous databases*, ACM Computing Surveys, 22(3) 1990.

[MD] C. Mohan, R. Dievendorff, *Recent work on distributed commit protocols and recoverable messaging and queuing*, Bulletin of the Technical Committee on Data Engineering 17 (1994) 22-28 .

[MKRZ] N.M. Mattos, J. Kleewein, M.T. Roth, K. Zeidenstein, *From object-relational to federated databases*, Proc. BTW'99 (Freiburg, Germany, March 1999).

[S] R. Schulte, *Message brokers: A focussed approach to application integration*, Gartner Group, Strategic Analysis Report SSA R-401-102, 1996.

[SL] A.P. Shet, J.A. Larson, *Federated database systems for managing distributed, heterogeneous and autonomous databases*, ACM Computing Surveys, 22(3) 1990.

[SST] H. Schuldt, H.-J. Schek, M. Tresch, *Coordination in CIM: Bringing database functionality to application systems*, Proc. ECEC'98 (Erlangen, Germany, April 1998).

[SSA] H. Schuldt, H.-J. Schek, G. Alonso, *Transactional coordination agents for composite systems*, Proc. IDEAS'99 (Montreal, Canada, August 1999).