CONCEPTS AND METHODS FOR THE OPTIMIZATION OF DISTRIBUTED DATA PROCESSING

S. Jablonski, T. Ruf, H. Wedekind

Department of Computer Science VI (Data Base Systems)
University of Erlangen-Nuernberg
Martensstrasse 3, D-8520 Erlangen

Abstract:

In this paper we introduce and discuss a model of distributed data processing. For this purpose, a typical application system is analyzed and divided into sub-applications. To fulfill the task of the global application, the sub-applications have to communicate in an appropriate manner by exchanging data resp. information. In our model the communication between sub-applications is split up into two steps: the offering of information by sending sub-applications, and its acceptance by receiving sub-applications. For both communication steps synchronous and asynchronous processing modes are defined. Supporting those different communication modes the cooperation between sub-applications can be defined very closely to the specific demands of the application system. This optimizes distributed data processing. At last we demonstrate the prototype implementation of a distributed data management system, which is based on the flexible communication mechanism described in the paper.

1 Introduction

1.1 Motivation

In a distributed computing environment there are basically two approaches supporting consistent distributed data management. First there are Distributed Data Base Management Systems (DDBMS) that comprise all nodes of a distributed computer system. Alternatively one can use a central Data Base Management System (DBMS), which is allocated at a central instance, called the server node. Although different in system architecture, both approaches bear the same conceptual problems due to the common notion of data consistency they are based upon. In DDBMS, the performance is normally quite bad, because serializability has to be guaranteed in order to preserve data consistency ([2]). For that purpose time consuming protocols, in particular distributed two-phase-commit-protocols [3], have to be performed. They are expensive, because high communication costs arise in a distributed environment. In the server concept the server itself is a potential bottleneck, because all data accesses have to pass the server. The extraction of data out of the server data base into local data pools doesn't improve the situation: data access to the data copies in the server is locked, while copies of those data are extracted. Supporting the classical concept of consistency, data updates have to be distributed and processed in all replicas of one logical data item as soon as they occur. This is also caused by the requirement of preserving the principle of serializability in classical data management approaches.

Besides the performance problems, both approaches - DDBMS and server DBMS - bear the significant disadvantage that they do not support site autonomy in a distributed environment. Site autonomy is the independence of computer nodes from other nodes of a distributed computing systems. Pursuing site autonomy implies local data access. This enables better system performance, because expensive remote data accesses can be avoided. If a data item has to be modified in a DDBMS, all nodes bearing copies of this data item are involved. This is conflicting with the requirement of site autonomy. In a DBMS server environment all nodes depend on the central node. If replicas are allowed, data updates have to be propagated synchronously as in the DDBMS environment. This is also in contrast to the notion of site autonomy defined above.

1.2 Related work

Some recent approaches in DDBMS research try to bypass the requirements of serializability and establish new processing strategies, which utilize knowledge about the underlying applications. For example, Garcia-Molina ([6]) introduces the notion of semantically consistent schedules.
This scheduling strategy supports a weaker notion of consistency as the one implied by serializability. It makes use of knowledge about the application by defining compatible transaction sets. Kumar and Stonebraker ([10]) define commutative transactions. A distributed transaction is regarded to be composed of a sub-transaction for each computer node involved. Sub-transactions of compatible transactions may be scheduled in arbitrary order at each node. Adiba and Lindsay ([11]) define the snapshot concept, which specifies read-only copies of data partitions. These data copies are extracted from the normal data processing of the base site and reflect a static view of the primary data for a specific moment in time. After some period, the snapshot will be refreshed, i.e., updates that were performed on the base data will be propagated to the snapshot data. As a result, the snapshot reflects a new state of the primary data pool after the refresh. The fact that the snapshot data are not up-to-date for a certain period of time is in accordance with the requirements of the application and can be tolerated therefore. While this is acceptable from the application's point of view, accessing these 'old' data versions clearly violates the traditional notion of consistency used in classical data management approaches.

All the methods of transaction processing mentioned above (similar approaches can be found in various publications) try to enrich the traditional notion of consistency with application knowledge. This enables performance improvements in executing distributed transactions. The most important disadvantage of the first two approaches ([6] and [10]) is the limited application area and the restricted usability of the new methods on the application level. First, the mechanisms proposed are only applicable for data types like integer or cardinal and for some specific arithmetical functions like addition, subtraction, and multiplication. Also it is difficult to define the compatible and commutative transaction sets, to recognize whether a transaction belongs to such a set has to be elaborated at runtime.

1.3 Overview

Our approach to optimize distributed data processing is based on the partitioning of data processing into a local and a global (distributed) part. Local data processing is restricted to the local computer node, while distributed processing involves remote nodes, too. Besides, data management is divided into data administration and data distribution. A local data administration system manages the data at a computer node (local access) and handles the data input and output from an application program to the storage system. The data distribution system has to perform all non-local data processing tasks, that means, it manages the data exchange between different computer nodes. In this paper we only consider the optimization of the global data processing issue by exploiting predefined features of the application. As the local data administration systems are fully autonomous in our approach, well-known mechanisms for performance improvements may be applied for the local task.

The realization of our approach in a distributed data management system is divided into a local and a global part according to the differentiation between local and global (distributed) data processing described above. The local task is performed by local data management systems like centralized database or file systems. Especially the local concurrency control and recovery issues are managed by these systems. The system for distributed data processing has to deliver functions for distributed concurrency control and recovery. Among other things, a strategy for global deadlock detection has to be implemented.

To enable the modelling of the specific application requirements on the data management level, we build up a tool-kit of different (synchronous and asynchronous) communication mechanisms between nodes of a distributed computer system. Individually using the communication mechanism which models the specific application best, the performance of the overall application system may be improved.

For any situation, the cheapest communication mechanism, which is still tolerable from an application point of view, may be applied. In contrast, using a DDBMS, only the very expensive synchronous communication mode is available. As a consequence of providing synchronous and asynchronous communication primitives on the user level, the traditional notion of consistency is replaced by a new concept of consistency, which is application oriented.

A very interesting advantage of our approach is its applicability to all data types and all kinds of operations, because our approach is state-oriented and not action-oriented (see Section 2). In addition to this, most part of the application knowledge can be processed in the definition and compilation phase of the system, and therefore doesn't affect the run-time performance of the system.

The rest of the paper is organized as follows. In Section 2 a model of a distributed application system is defined, especially the concept of distributed data processing is introduced. In Section 3 a mechanism for data distribution be-
tween the sub-applications of a global application system is demonstrated. The implementation of this mechanism is described in Section 4. The summary of Section 5 concludes the paper.

2 A Model of a Distributed Application System

2.1 Definition of Sub-applications

For the further discussion we want to define a model of a distributed application system. We divide an application into several sub-applications. Each sub-application is characterized by the fact that most work initiated in a sub-application is done within the borders of that sub-application. In order to demonstrate this we look at a CIM application system (Figure 1). To simplify matters, the whole CIM application system is divided into the sub-applications CAD (computer aided design), CAP (computer aided planning), and CAM (computer aided manufacturing). All work initiated in the CAD sub-system is mostly limited to its sphere of control. Only at specific events communication with other sub-applications, like the CAP sub-system, is necessary for exchanging information. This information exchange is necessary to fulfill the overall task of the CIM part production system. We analyze this in more detail later on. Nesting of sub-application is possible, but is neglectable for the concerns of this paper. The relationships between sub-applications and computer nodes may be m:n, but for the discussion in this paper, considering 1:1-relationships is sufficient.

The constituents of a sub-application are: actions, data, and local rules (Figure 2). Actions are the active elements of a sub-application. For example, transactions are a special kind of such actions. Data are the passive elements of a sub-application. They bear all the information stored for a sub-application. The execution of a local action is controlled by the local rules, which define the logging and locking behavior of an action. For instance, a data base transaction is characterized by the local rule that all data elements being accessed have to be locked and that before and after images have to be generated.

It has already been mentioned that for the overall task some communication between sub-applications is necessary (Figure 2). We define global rules which specify the communication between sub-applications. The global rules in Figure 2 define the amount of information to be exchanged between sub-application SAi and sub-application SAl. In our model only data bear information. The aim of an information exchange is to actualize replicated data that have been modified in a sub-application independently from others. Therefore, the global rules specify which data (or at least which data modifications) have to be exchanged between sub-applications. In addition to the kind of information to be exchanged, time and strategy for information passing are defined by the global rules. Section 3 describes the semantics and the structure of the global rules in more detail.

2.2 Definition of Distributed Data Processing

The basic constituent of data processing is a data processing action. Such an action is split up into four steps. The initiation step is the first one. The last one is called completion step. These two processing steps enclose the real data processing steps, which are divided into a local and a global processing phase. In the surrounding initiation and completion steps the local rules, which are specific for the sub-application, have to be fulfilled. For example, the execution of a data base transaction requires in the initiation step the extraction of before images of all modified data base pages. In the completion step, after images have to be derived. Additionally, in the initiation step all data base items to be accessed by a transaction have to be locked.
Those locks must be released in the completion step. If local data are accessed, this is performed in the local data processing phase. Obviously, all global data processing is delegated to the global processing phase. In our model global data processing means to prepare the local data of a sub-application for an exchange with some external sub-applications. For instance, a time stamp has to be generated to signal the last modification of a data item. If another sub-application is interested in the data modifications since some predefined time, those data can be extracted easily.

Now we are able to define a distributed data processing action\(^1\) as the combination of a primary action and a secondary action. The primary action is a local data processing action that is initiated unconditionally by an application. The effect of this action is a change in a set of local data. According to the assumption that another sub-application is interested in this information (modifications), another data processing action, the secondary action, has to be initiated in the corresponding sub-application to process that information in its own data pool. Each receiving sub-applications may define individually when the secondary action has to take place. We will discuss this issue in more detail in Section 3.

An example from the CIM application area will demonstrate the meaning of primary and secondary action resp. distributed data processing action. Suppose a new manufacturing order is generated in the CAP sub-application. From a data oriented point of view, this is an insertion of a data item into a file (relation) called 'manufacturing-orders' in the local processing phase. This activity makes up the primary action. In the global processing phase the CAP sub-application prepares the data modifications for a later exchange to the CAM sub-application. In the CAM sub-application, the primary action's effect has to be reflected. Therefore, a secondary action is initiated, which processes the offered information (data modifications) into the data pool of the CAM sub-application, i.e. inserts the new data item into the CAM data pool.

In our model the local and global data processing actions are separated strictly. Local data processing occurs within the local data processing phase of a data processing action. Global data processing is realized by the execution of primary and secondary data processing actions, which in combination form a distributed data processing action.

There are some interesting advantages induced by the separation of local and global data processing:

- All well-known and optimized scheduling strategies for local data processing at one computer node may be applied.
- For the global data processing task, application oriented scheduling strategies (see Section 3) are applicable.
- For local resp. global data processing, specialized optimization mechanisms can be defined independently.

Beyond these advantages, dividing an application area into sub-applications introduces the possibility of defining local rules for each sub-application individually. This enables the integration of simple file systems and data base system – they mostly differ in using different strategies for locking and logging – under one global data management system shell. Furthermore, the cooperation between different sub-applications may be specified independently for each pair of sub-applications in both directions. These advantageous features of our model for distributed data processing allow a very close modelling of an application system on the level of data (information) processing. This is a natural and promising way of optimizing the performance of such a system. Static, non-flexible mechanisms for modelling often cause much overhead; flexible mechanisms allow to choose the best-fitting strategy, which optimizes the overall system performance.

As we consider the cooperation between sub-applications as a very interesting, up to now neglected, and above all optimization bearing research area, we propose strategies for application oriented communication in the next section.

3 Application Oriented Data Distribution Mechanisms

For the following discussion we assume that two sub-applications want to communicate with another. In one of these sub-application a data set will be modified by a primary action. This may be considered as information generation. The other sub-application is interested in this modification. As Figure 3 demonstrates, the exchange of information can be split up into four steps. In the first step, a primary action generates new information that is interesting for the external sub-application. Next, it offers that information by putting it into a virtual buffer. This process

---

\(^1\) In this paper, we concentrate on modifying actions. Nevertheless, reading actions can easily be integrated in our model.
is initiated by one or more triggers, which are defined specifically for each application. When such a trigger fires in the corresponding sub-application, the offered information will be accepted and processed. It is important to note that this information exchange is regarded without considering protocolling. Only the principle framework for exchanging information between sub-applications is discussed here. In the next sub-sections strategies for offering and accepting information are demonstrated.

3.1 Offering Information about Data Modifications

In order to specify a combined model of data modifications exchange, we first have to discuss when data modifications of a given sub-application will be offered to other sub-applications. Two solution for this problem are possible: First, the information can be offered immediately when they are generated in the primary sub-application (Figure 4a). An alternative way of offering this information is to wait until an external event occurs (Figure 4b). This causes a trigger to be fired. It puts the information into the virtual communication buffer. Choosing the immediate propagation of modifications in the global phase of the primary action, the information is directly sent to the buffer. Applying the indirect way of propagation in the global phase of the primary action, the modified data have to be prepared first. A substitutional (primary) action, initiated by a trigger, selects the information (of several primary actions) and sends them to the buffer. The most interesting feature of this delayed data propagation is the aggregated and therefore cost-effective data exchange between sub-systems according to the requirements of the application. The events which initiate the data transfer are specified by the application programmer. This offers an appropriate way of optimizing the performance of the data management system. The application can be modeled in the data management system very closely. For example, the propagation of information may be delayed until midnight, because it is not necessary to offer it earlier. So, no other running processes are disturbed by the data exchange and the performance of the system will be enhanced.

3.2 Accepting Information about Data Modifications

Similar to the alternative strategies for propagating information about data modifications, the acceptance of that information can be performed in two ways. The first one is to accept the information as soon as it is offered, i.e. the information has arrived in the virtual buffer (Figure 5a). This information is always generated by a single primary action. Another way to handle offered information is to accept it when an application specific trigger has fired (Figure 5b). In this case, the information of the virtual buffer is related to several primary actions in general. It is very important that the initiation of a secondary action is always triggered. In the case of an immediate acceptance of information (Figure 5a) the arrival of that information fires a virtual trigger. In the case of Figure 5b the controlling function of the trigger is obvious. Also for the acceptance of information about data modifications the optimization of an application system is possible, because the best-fitting strategy is selectable.
3.3 Defining Distributed Data Processing Actions

The aim of this sub-section is to demonstrate how the different strategies of propagating and receiving modification information can be combined.

3.3.1 Immediate Synchronous Processing

If the immediate propagation and the immediate reception of modification information are combined, the immediate synchronous way of processing a distributed action results (Figure 6). The information about modifications is offered at once in the global processing step of the primary action, and the secondary action has to process this information synchronously. If the secondary action has finished, the primary action may end, too. The motivation for the immediate synchronous processing is to manipulate data simultaneously in different sub-applications. All DDBMS are characterized by this processing strategy. The strategy is very expensive, because primary and secondary actions have to be coordinated in order to run quasi-parallel. For achieving fault tolerance, a two-phase-commit-protocol is necessary to complete both actions in an atomic step. A typical application of this mechanism can be found in the banking area. If a certain amount of money is withdrawn from a banking account by a client, all branches (sub-applications) of the bank have to be informed immediately for avoiding the client to withdraw more money than he is allowed to.

3.3.2 Immediate Conditioned Processing

The second way of distributed data processing is to combine the immediate propagation and the delayed acceptance of modification information (Figure 7). In the global processing phase of the sending sub-application the primary action informs the external sub-application that new information was generated; then it finishes. For each receiving sub-application a trigger is defined that controls the reception of the information. As a result, the secondary action is responsible to extract the modification information out of the virtual buffer. The advantage of this immediate conditioned data processing strategy is the decoupling of the primary and the secondary action. Therefore, no cost-intensive protocols (like the two-phase-commit-protocol) have to be performed, which increases the performance of the overall system.

There are many applications areas where the cheap immediate conditioned processing mode is applicable. In the example of Section 2.2, manufacturing orders are sent from the CAP sub-application to the CAM sub-application. Normally, this does not have to happen as soon as a manufacturing order is generated in the CAP system. Typically, a production system is organized in production shifts. Therefore, the propagation and acceptance of information can be adapted to the shift organization. Orders that are processed during the present shift are not disturbed by the transmission of new orders for the next shift when the communication work is displaced to the end of a shift. Compared to an immediate exchange, the performance of the system is significantly increased by collecting various data modifications and transmitting them in a single block.

3.3.3 Indirect Synchronous Processing

The combination of delayed propagation and immediate reception of modification information makes up the indirect synchronous processing strategy (Figure 8). A substitutional primary action puts data modifications of former primary actions into the virtual buffer. A secondary action receives this information immediately and processes it in its data pool. Like in the immediate conditional processing mode, the decoupling of primary and secondary action is essential. The difference to the immediate conditional
processing is that the responsibility for transporting the modification information from one sub-application to another is transferred to the sending sub-application.

The example of Section 3.3.2 can also be considered as an example for the indirect synchronous communication mode. Now the CAP sub-application is responsible for the information transport, which may be more appropriate for organizational reasons.

3.3.4 Indirect Conditioned Processing

The fourth way of distributed data processing is to choose the indirect way of data propagation and the conditioned way of data reception. This strategy is called indirect conditioned data processing (Figure 9). Like in the previous two processing modes, the decoupling of primary and secondary action is an advantage of the indirect conditioned processing; high communication costs for applying synchronous communication primitives can be avoided. The responsibility for transporting the data from one sub-application to another is transferred to the receiving sub-application. The sending sub-application is only responsible to inform the other sub-application that new information has been generated.

An example for the application of this processing strategy is a system that is characterized by a sequence of processing steps. A preceding step can put information into the buffer at his own decision. The succeeding step can extract this information out of the buffer independently, too. So the sequential steps interfere with another as rarely as possible.

In Section 4 a distributed data management system is introduced, which realizes the various distributed data processing modes introduced above. The implementation of that system demonstrates the feasibility of applying the various distributed data processing strategies.

3.3.5 Conclusion

Having introduced the various modes of distributed data processing, the question for the optimization of distributed data processing arises. The described methods (immediate synchronous processing, immediate conditioned processing, indirect synchronous processing, and indirect conditioned processing) are characterized by a decreasing amount of synchronization effort. Synchronization is one of the most weighty tasks in coordinating distributed actions.

In the case of immediate synchronous processing, a distributed two-phase-commit-protocol is necessary to execute the communication. Therefore, this is a very time consuming way of coordinating the primary and secondary actions. Applying the other ways of distributed processing, the primary and secondary actions are separated. They need not to be synchronized by a distributed two-phase-commit-protocol. Using the mode 'immediate conditioned processing' a (light-weighted) distributed protocol is necessary to inform the external sub-applications about data modifications. In the case of indirect synchronous processing, a (light-weighted) distributed protocol is only required for the coordination between the substitutional primary action and the secondary actions; but this protocol doesn’t disturb the application processing. The most negligible influence on the application processing shows the indirect conditioned way of distributed data processing. The one and only distributed action is the information of the exter-
nal sub-applications about data modifications in the substitutional primary action.

With the mechanisms introduced above, an application can be optimized by applying the cheapest way of distributed data processing which is still acceptable. Therefore, our approach optimizes the application by modelling it as closely as possible in the underlying data management system. Using a DDBMS, only the immediate synchronous data processing mode is available, and this is the most expensive way of distributed data processing. The gap between the consistency request of the application and the realizing data processing mechanism of a DDBMS causes avoidable overhead.

4 Implementation of a Distributed Data Management System

In this section, an overview on a distributed data management system (DDMS) is provided. This system is implemented in a network of several VAX/VMS computers at the Institute for Data Base Systems at the University of Erlangen-Nuernberg. The objective of this installation is to demonstrate the feasibility of the distributed data processing mechanisms proposed. In addition to the basic concepts, an overview on the general architecture, and a discussion of the most important problems is given. A more detailed description of the DDMS concept and its application can be found in [9] and [8].

4.1 DDMS Concepts and Architecture

The DDMS approach has to fulfill some principle requirements. According to a partitioning of the data management task into the sub-functions data administration and data distribution, the DDMS only has to realize data distribution. Data administration is handled by the already existing local systems like data base systems or file systems.

The general architecture of the DDMS is shown in Figure 10. Some existing data administration systems are integrated into the DDMS by a global data distribution shell. The local data management systems may be very heterogeneous, because only the basic functions of data management (reading, storing, manipulating, and deleting of a tuple/record) have to be provided. Therefore, the DDMS also sets up an integration platform for data management in heterogeneous computing networks. For example, in the first DDMS prototype that has been implemented, simple file systems (ACCESS MANAGER, RMS) and relational data base systems (INGRES, RDB) are operated under the global shell of the DDMS simultaneously.

A special feature of the DDMS data distribution shell is the SQL-like global data manipulation and query language in all nodes of the computing network. The different data administration systems that are used for local data administration can be accessed by the same unique language; the heterogeneity of the different languages of the underlying sub-systems is hidden by that global DDMS interface.

A basic requirement of the DDMS is the integration of application knowledge into the data management system. This is realized by the definition of application oriented triggers, which control the data communication in the distributed system. Therefore the DDMS may be regarded as a special kind of an active data base management system ([4]). A trigger in the DDMS consists of three parts: events, conditions, and actions. The semantic of this construct is the following: if a predefined event occurs, all triggers related to this event are initiated and the corresponding conditions are evaluated. If these conditions are fulfilled, the trigger actions are executed.

In the current version of the DDMS, trigger actions are limited to data distribution functions. In the next version of the DDMS triggers may be used to invoke user defined actions as well (e.g. mailing). As the events and conditions that may be used right now are the same as those described in [4], this generalization will transform the DDMS into a complete active data base management system.

4.2 Operation of the DDMS

The installation and operation of the DDMS in a distributed environment is demonstrated in Figure 11. At a dedicated node of the distributed computer system a DDMS definition sub-system is allocated. Here, the DDMS system operator specifies the allocation and distribution of data in the distributed system. The definition system puts all this information into a global data dictionary. A sub-system of
the definition system generates the local data dictionaries and the user data bases at the various nodes of the distributed system automatically. Also a DDMS kernel is installed at each node. After this installation phase the different nodes in the system are operated totally independent from each other, as every node contains all the information necessary to perform its data administration and distribution.

The structure of the DDMS kernel is shown in Figure 12. Its most important part is the trigger system, where the triggers defined by the DDMS users are controlled. The trigger system consists of three major parts: the event controller supervises the application and waits for the predefined events. The condition evaluator proofs, whether trigger conditions hold. If the conditions are true, the corresponding actions are executed by the action handler. The communication system is responsible for the physical data transfer between the nodes of the distributed computer system. The local data dictionaries bear all information necessary for a node to handle the local and global data management tasks. As stated above, this results in each node being independent from all other nodes.

4.3 Problems and Future Research

The major problem of the DDMS approach arises from the permission of asynchronous distribution of data modifications. If it is possible that a data item that is replicated at two nodes is modified independently at both nodes, two different versions of that data item may exist at the same time. Two main questions arise: Which version is correct? How can differing versions be merged, so that one global version can be derived? The answers to these questions are discussed in [7]. We only summarize that discussion and present the main results here. First, at a determined moment in time, it is possible that for different sub-applications different versions of data items are correct. An example is the sample application described in Section 2.2. The CAM resp. the CAP sub-application see two different versions of the 'manufacturing_orders' data pool at a specific moment in time. For both sub-applications the local view is correct, because both applications are interested in different data: the CAP sub-application is mainly interested in future data (orders), while the CAM sub-application is interested in the current data (orders). Besides, in [7] some strategies are proposed to merge different versions of a data item with the objective to gain one global version of that data item.

The future research in the DDMS field concentrates on two topics. First, the application of the DDMS in a manufacturing environment, which has already been realized to a certain degree, has to be extended ([12]). Qualitative and quantitative performance measurements have to be derived from this example. Second, recovery mechanisms for the DDMS have to be developed. Some introducing ideas for recovery techniques have already been proposed in [5].

5 Summary

The major problem of conventional data management concepts for distributed computer systems (DDBMS and server DBMS) is their bad performance. This mostly depends on maintaining the principle of serializability, which results in supporting synchronous, quasi-parallel distributed data processing actions exclusively. For each data management action time consuming two-phase-protocols are required in these systems. Even in applications where a less restrictive communication protocol is applicable, the expensive synchronous way of cooperation has to be applied. Of course this decreases the performance of the data management functions and therefore of the whole application system. In contrast to that inflexible concept, we pro-
pose several alternative strategies for distributed data processing. Especially for asynchronous data processing strategies, several communication protocols are offered. Applying an asynchronous communication mode between sub-applications enhances the parallelism of data processing in these application areas: the coordination is reduced to a minimum, as it must only be performed at application-specific events which may occur quite seldom compared to the frequency of data accesses and manipulations.

The paper shows that consistent data management is possible without using a database system involving all the well-known performance problems. Data consistency does not depend exclusively on serializability. Instead of this restrictive and only extensionally defined concept, we introduce an intensional and application oriented notion of data consistency. By using a powerful trigger facility, knowledge of the application system can directly be integrated and applied for data management. So the behavior of the application system can be supported in an optimized way by the data management system. By exploiting application specific knowledge a substantial improvement of the system performance is possible in contrast to the optimization of some other optimization approaches ([11]).

The implementation of the DDMS demonstrates the realizability of the flexible data communication strategies proposed. The DDMS concepts also shows how to integrate existing heterogeneous data management systems into a global distributed data management system. The heterogeneity of the underlying system is transparent at the user interface of the DDMS, where a SQL-like data definition and manipulation language is offered.

References