In: Proceedings of the 5th European Concurrent Engineering Conference (ECEC'98), pages 223-230, Erlangen, Germany, April 1998.

COORDINATION IN CIM: BRINGING DATABASE FUNCTIONALITY TO APPLICATION SYSTEMS[*]

Heiko Schuldt Hans-Jörg Schek Markus Tresch[†]

Database Research Group
Institute of Information Systems
ETH Zentrum
CH-8092 Zürich, Switzerland
Email: {schuldt,schek,tresch@inf.ethz.ch}

KEYWORDS

Engineering process management, Workflow–management in CE, Change–notification; Collaborative CE environments, Information and application sharing; Networking and Distribution in CE, Architecture for building CE systems, Distributed computing environments.

ABSTRACT

Coordination in CIM means enforcing dependencies between autonomous application subsystems. We propose an architecture that generalizes multidatabase technology and consists of a global coordinator and local specialized agents, one for each subsystem to be coordinated. The agent associated to a subsystem not only monitors relevant subsystem activity but also guarantees certain properties of operations executed in a subsystem. Therefore, it has to provide database functionality for subsystems even in cases they do not rest upon databases.

The coordination itself is a synthesis of (advanced) transaction models and workflows. Aside from an introduction to the transactional coordination model, we concentrate in this paper on the analysis of different classes of subsystems and elaborate on properties required from the subsystem and its agent to ensure execution guarantees. Furthermore, we examine to which degree database functionality can be provided for application systems with a given set of properties.

Our examples are taken from an industrial project in the area of computer integrated manufacturing.

[*]This work has been partially supported by the Swiss Commission of Technology and Innovation (KTI) under contract number 3008.2

[†]Present address: Credit Suisse, Advanced Technologies, Postfach 100, CH–8070 Zürich

1 INTRODUCTION

The IT infrastructure of large companies is characterized by successively grown composite systems, consisting of different heterogeneous and autonomous application subsystems, each of them being specialized for certain application domains. The co-existence of these specialized applications can hardly be avoided, because they are an important factor for the processing of daily work. For historical as well as for organizational reasons, applications, located at physically distributed sites, are not made to cooperate with each other. A result of this inherent heterogeneity of applications of composite systems is both the distribution of application logic to several application systems and the distribution of data to several resource managers that are potentially running on distributed and heterogeneous hardware platforms.

Dependencies between subsystems in composite systems exist (e.g. dependencies due to replicated data at different resource managers or more complex dependencies in form of business processes). The goal of the coordination approach we will present in this paper is to enforce these dependencies even in case of failures or concurrency and to provide execution guarantees. In classical database systems, these properties are well-known and guarantees are provided by the ACID[‡] properties of transactions [9]. However, we have to face two problems when coordinating subsystems in composite systems: (i) We cannot assume that subsystems are databases. In general, subsystems are arbitrary application systems and we cannot even assume that they are built on top of a database system. (ii) Considering the processes needed to enforce dependencies within composite systems, classical transaction models are too restrictive. We therefore have to provide more general and flexible means to ensure execution guarantees in composite systems. Further-

[‡]**A**tomicity, **C**onsistency, **I**solation, and **D**urability.

more, we have to export database functionality even for non-database applications in order to integrate arbitrary subsystems within a composite system in the transactional coordination approach.

1.1 Coordination in CIM

A typical example of composite systems can be found in CIM (Computer Integrated Manufacturing) where different engineering tasks have to be coordinated and data has to be exchanged between various specialized tools.

The information flow of an enterprise can be classified in two parts: technical data and managerial data [19]. Systems of the managerial part include mainly applications for business engineering, stock management, but also production planning and control (PPC). The technical track consists for example of applications for supporting product design (CAD), the programming of numerically controlled machines (CAM), or quality control (CAQ).

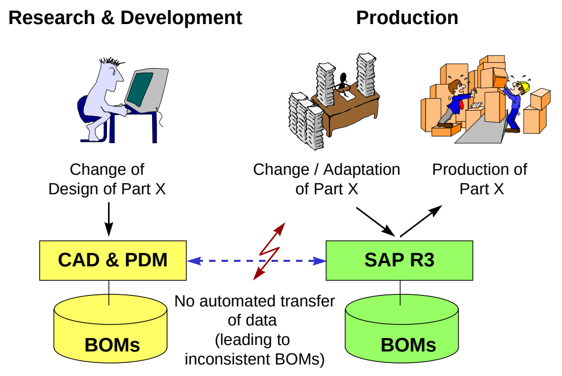
Research & Development
Production
Change of Design of Part X
Change / Adaptation of Part X
Production of Part X
CAD & PDM
SAP R3
BOMs
No automated transfer of data (leading to inconsistent BOMs)
BOMs

Figure 1: Coordination of subsystems in CIM is crucial to ensure consistent product data (e.g. BOMs) in CAD, PDM and PPC applications

Information about products is crucial in both parts. Therefore, product data is used both in subsystems of the technical track and of the managerial track. As each subsystem stores data in a separate, private repository and uses its own data model tailored to the specialties of the respective applications, many dependencies between subsystems exist. An example is depicted in Figure 1. An employee of the R&D department of an engineering company changes the design of product X (e.g. due to some fatal failures) in a CAD system leading to an update of the BOM (bill of materials) in the underlying database system. Then, a coordination process should be started to track this change for example to a PPC and to a business engineering system to update production plans and to apply changes to the purchase process of parts. Even if the R&D employee executes his operations locally and is unaware of operations that have to follow, all subsequent operations have to lead to a well-defined state. This is even more crucial if changes are executed not only at the R&D department but also at the production department (e.g. to adapt certain properties of a product) leading to concurrent coordination processes.

The coordination process itself therefore has to enforce dependencies within composite systems (e.g. system-wide consistency of product data) by guaranteeing that operations in related subsystems are executed with certain execution guarantees.

1.2 Related Work

In CIM, various different efforts have been taken to overcome the problem of the enforcement of (rather basic) dependencies: the avoidance of inconsistencies resulting in replicated product data that is stored under the control of heterogeneous and autonomous application systems.

A first approach is to skip the goal of keeping local autonomy of the participating application systems and to integrate local applications into global ones. Then, access to global data is only possible via the integrated applications where global integrity is ensured in the traditional way [2].

A further approach is to introduce an additional database outside the existing application systems to additionally replicate global data [3]. Global integrity constraints can then be defined on global data of the integration database. However, additional overhead has to be dealt with as replication leads to additional costs.

Synchronous communication mechanisms can also be exploited for keeping data consistent after local operations and for enforcing global integrity [10, 17]. Instead of the overhead of a centralized database, additional effort by defining inter-application communication has to be taken on. This leads, however, to a rather loose interconnection of application systems and does not allow to support complex coordination processes within composite systems.

In our own previous work, we introduced the agent/coordinator idea [15, 22] and concentrated first on the coordinator data model. As the provision of execution guarantees for more sophisticated coordination processes is crucial for various real–world application scenarios, we expanded this coordination approach by the introduction of a transactional coordination model [21] which is based on previous work of the database research group of ETH on composite transactions [1] and on the unifying theory of concurrency control and recovery [20].

A major contribution of the work presented in this paper is to show how agents can provide database functionality for CIM subsystems in order to realize transactional coordination [21, 12, 18] as a synthesis of advanced transaction models and workflow management for solving the inherent problems of composite systems.

Structure of the Paper. The remainder of the paper is organized as follows: In Section 2, we briefly introduce the coordination model on which transactional coordination is based. The architecture of our coordination approach is presented in section 3. We both analyze all components involved in this architecture and figure out the properties subsystems are supposed to have and the database functionality agents can add to them. The prototype system presented in Section 4 then describes how database functionality can be brought to two selected application systems in CIM by specialized agents. Section 5 finally concludes the paper.

2 TRANSACTIONAL COORDINATION MODEL

In this section, we introduce the model of our coordination approach called *transactional coordination* [21], which is based on both advanced transaction models [1, 20, 7, 23] and workflow management [18, 8].

The model is based on processes defined over distributed, heterogeneous and autonomous standalone applications within a composite systems. Formally, processes are directed acyclic graphs consisting of activities, an order between these activities (control flow) and an additional oder specifying alternative orders in case of failures. With this notion of processes we extend classical transaction models by providing a more general concept of both atomicity and isolation based on the notion of spheres [6] and by allowing a greater degree of flexibility by partial compensation and alternatives [11, 8].

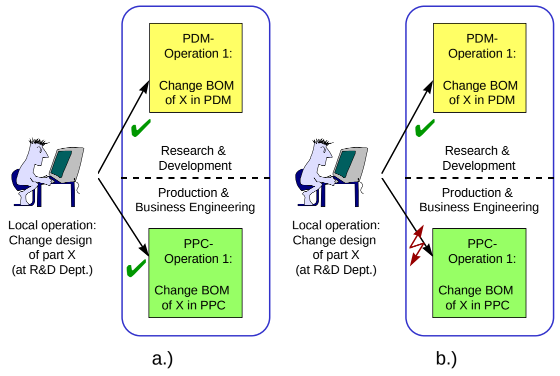

Figure 2: a.) Successful ACID coordination; b.) failed ACID coordination

In the classical sense, database transactions have ACID properties. This means, for instance, that all operations are executed successfully or none of them in order to avoid inconsistent states. However, it is not always reasonable to have this "all–or–nothing" semantics of atomicity. Consider a long running construction activity within a CAD system starting a subsequent coordination process to track changes to BOMs. Suppose further that, because of some subsystems' constraints, a BOM cannot be changed and the coordination process has to abort and to undo also the construction activity as no inconsistent states are allowed. This, of course, cannot be tolerated. Figure 2 a.) depicts a successful coordination process with ACID semantics. After a local change of a design, both associated BOMs in the PDM system and the PPC system have to be changed, too. If one of the coordination operations fails (Figure 2 b.) the "all-or-nothing" semantics of atomicity implies that no operation is visible at all. This means, that all operations including the initial (eventually long-running) design transaction have to be rolled back.

With partial compensation and alternative activities, we can now avoid to completely abort a process and to undo all activities even if they have already been committed by compensating only single activities until a point is reached from where another alternative can be executed. This leads to a more general notion of atomicity as only one of several possibilities has to be executed in order to terminate a coordination process successfully. This property is denominated by the notion of *sphere of atomicity* denoting a (sub)process where the successful execution of at least one alternative can be ensured (this guarantee can be derived from the properties of single activities).

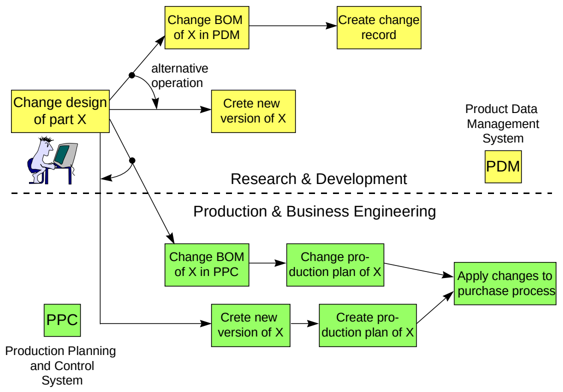
Change BOM of X in PDM
Create change record
alternative operation
Change design of part X
Crete new version of X
Product Data Management System
PDM
Research & Development
Production & Business Engineering
Change BOM of X in PPC
Change production plan of X
Apply changes to purchase process
PPC
Crete new version of X
Create production plan of X
Production Planning and Control System

Figure 3: Flexibility of the transactional coordination model by allowing alternative activities

In Figure 3, a coordination process is depicted allowing alternative activities in each subsystem. In case one activity fails (e.g. the change of BOM X in the PPC system), another alternative (the creation of a new version) can be executed and the whole process may still terminate successfully. In this case, the rollback of the initial design activity is avoided. Given that each time at least one of the alternatives can be executed successfully, this process is a sphere of atomicity.

The notion of spheres not only allows a more flexible definition of transactional properties but also their decoupling. This means, that spheres of isolation and spheres of atomicity can be defined independently (although they have to meet some restrictions in order to ensure correctness).

A sphere of isolation is a means to shield concurrent coordination processes in order to avoid incorrect interleavings. This is crucial when for example two different users simultaneously change related data elements (e.g. two replicas) at different application systems causing two parallel instances of the same coordination process. Then, the parallel execution of this processes has to be restricted and incorrect interleavings have to be avoided.

3 COORDINATION ARCHITECTURE

Autonomy of application subsystems leads to cases where local operations within one subsystem violate global consistency necessitating coordination processes within composite systems. In this section, we present the architecture of our coordination approach. In short, the approach is based on agents, sitting atop of subsystems and sending notices on local subsystem activities to a coordinator, which invokes global coordination processes.

In practice, the most common approach to guaranteeing system-wide consistency across application subsystems is to delegate responsibility to the user. He or she has to ensure that appropriate actions are taken within all affected subsystems. Configurable or even custom developed “information pump” software tools exist, that assist the user in pushing and translating data and operations between application subsystems. However, most of these tools are made to link only two distinct systems but are neither considered to automatically detect operations violating global consistency nor to automatically execute operations required to reestablish system-wide consistency.

Our approach goes far beyond. We do not only propose a general coordination architecture used as a basis to automatically reestablish system-wide consistency but we further elaborate on the extension to flexible failure handling and the realization of global execution guarantees on top of this architecture. The prerequisite to this is –as we will see in the following sections– to extend existing subsystems to be coordinated by special agents providing them with database functionality. The general coordination architecture used as the basis for the later realization of global execution guarantees is depicted in Figure 4.

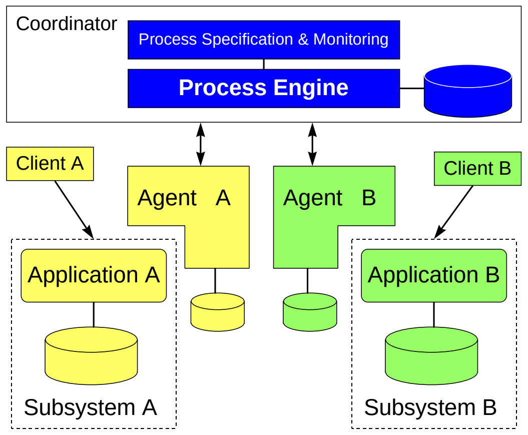

Figure 4: General architecture for coordinating autonomous subsystems

The goal of coordination is to enforce dependencies between application systems though data is manipulated locally via the respective applications. As the desired autonomy of applications allows such local operations eventually violating global consistency, we have to provide a special agent for each subsystem to detect local operations. Monitoring local operations violating global consistency is the prerequisite for the subsequent coordination processes, initiated by a global coordinator. Within this coordination processes, operations at the concerned subsystems (which are mediated by the respective agents) are executed for reestablishing consistency [15, 22].

The guarantees realized by a coordination process strongly depend on the services of the underlying agents, and they likewise depend on the willingness of the subsystems to cooperate. Different subsystems/agents may have different properties. In the following sections, we analyze the different properties of subsystems, the database functionality agents can add to them, and the services subsystems/agents can guarantee.

3.1 Coordinator

The task of the global coordinator is to enforce dependencies after they have been violated by a local operation by initiating and controlling the execution of a coordination process and invoking the necessary sequence of activities in the application subsystems. These operations in related subsystems have to reestablish a system-wide consistent state by means of the transactional coordination model described above.

The process specifications are stored in the repository of the coordinator. The specification and eventually also the simulation of coordination processes is realized by a special user interface. This modeling tool also provides means to check whether the definition of spheres is correct (whether the specified properties can be ensured by the coordinator). The execution of coordination processes is controlled by the process engine. Here, local operations are scheduled according to the transactional coordination model. Ordered invocations of subsystem operations are then delegated to the agents that have to keep orders determined by the coordinator. The coordinator itself has to control the correct execution of coordination processes and to react in cases where failures occur. Therefore, it needs a repository for persistent bookkeeping (e.g. logging) of executed activities of a coordination process. This bookkeeping together with the ability to maintain a dependency graph for conflicting activities is necessary to realize correctness and synchronization of concurrent global coordination processes.

However, the execution guarantees that can be provided by the coordinator strongly depend on the properties subsystems have and the services agents can provide on top of them.

3.2 Subsystems

Different subsystems together form the basis of the coordination architecture as shown in Figure 4. Each of these application subsystems manages its data separately and provides a specific data model (objects and operations). Parts of this data model specify activities that are exported to a coordination process. Moreover, different application subsystems may be based on different data sources (e.g. relational DBMS, object-oriented DBMS, file systems, etc.). As the underlying data sources differ, the support of application subsystems for the two coordination tasks *monitoring of local operations* and *local execution of operations initiated by the coordinator* differs, too.

To support the first coordination task, the local execution of operations initiated by the coordinator, an *ideal subsystem* has to provide by an API exactly the same set of operations that can be accessed via the subsystems' GUI. In order to enable the monitoring of local operations, an ideal subsystem should support trigger-like mechanisms at application level. Before a local operation is committed, these mechanisms take over the control enabling the notification of the coordinator, the transfer of data, and eventually the deferment of operations until the end of the coordination process.

In real-world scenarios, we will hardly find subsystems with these ideal properties. Therefore, special agents (see section 3.3) for each subsystem have to be provided exploiting the properties of the respective subsystem and extending it by adding database functionality. However, in order to enable the coordination of subsystems, certain basic characteristics have imperatively to be provided. A subsystem should both provide mechanisms for bringing in data from outside and for supporting the monitoring of local operations in some way. We assume both properties for given, thus imposing some minimal form of willingness of the subsystem to cooperate. Without this capability, there is no way of incorporating a subsystem into the coordination architecture.

The extraction of data can in general be realized at three different levels: application level, DBMS level (if a database management system is used by the respective subsystem) and data level.

Monitoring at data level requires very few prerequisites of the subsystem. It can be done for example by dumping the contents of the underlying data source in special files and to compare this dump with earlier versions (snapshot differential problem). However, this is very inefficient especially when huge amounts of data have to be inspected. Besides, information about data that has been changed does not necessarily enable the deduction of the initial operation as the schema of the dumped data may be unknown. Some application systems provide log files where all operations that have been executed are recorded (e.g. the PDM system WorkManager [5]). Such mechanisms rather support coordination as the semantics of that operations is available. Both monitoring approaches at data level described here have however the disadvantage that operations can be observed only after they have been committed.

If an application system is based on RDBMS, then triggers can be applied to notify about operations and eventually to defer a local operation. In this case, a mapping of database operations to operations at application level has to be available (this is the case if, for example, "pure" database systems are used for the management of product data). However, this task is in general very complicated and again does not necessarily provide information about the operation at application level being responsible for the monitored database operations. The reason is that application objects may be spread over several database relations whereas the schema of the database is unknown and meta information necessary for capturing the semantics of data is not directly available (as it is for example with SAP R/3 [4]).

The creation of a log file by database triggers and the examination of that log by the agent can be considered as hybrid approach as monitoring both at database and at data level is performed. This strategy is needed for systems like the PDM application WorkManager when triggers indicate that an operation has taken place and the log file provides the necessary information about this operation.

The third possibility, the monitoring at application level, is based on the largest set of prerequisites. In this case, either trigger mechanisms at application level have to be provided or the customization of application logic is required. The advantage of trigger mechanisms at application level (e.g. Pro/Engineer [16]) is that it enables the deferment of the execution of a local operation until the end of a coordination process. The customization of application logic may only enable the deferment of operations in some cases (depending on the degree of allowed customization), however the notification about local operations without their deferment is always possible with this approach (e.g. in SAP R/3).

We mentioned earlier that normally the schema of the database of a subsystem is unknown. Therefore, the agent may not be able to execute operations directly at the database level. If no API supporting the execution of local operations is available, only the information of a user who then has to execute operations manually can help in these cases. Though this seems to be a contradiction to the goal of automatically coordinating subsystems it is in some cases nevertheless the only possibility to support system-wide consistency.

3.3 Agents

The task of an agent is to detect local operations of a subsystem, to select those operations possibly affecting global consistency, and to overcome the inherent heterogeneity of composite systems by transforming objects of the data model of the subsystem to a global data model. Finally, the most important task of an agent is to execute operations of a coordination process initiated by the coordinator on its subsystem with certain execution guarantees. These guarantees have to be provided either by the subsystem itself or the agent in cases where the subsystem is not capable to do so. Agents can therefore be considered as database systems providing execution guarantees to the coordinator in a similar way component database management systems do for distributed transactions.

To overcome the inherent heterogeneity of composite systems, all agents must provide the same interface to the coordinator. The communication between agent and coordinator has to consist of notifications of local operations violating global consistency. Furthermore, the agent has to acknowledge (or nacknowledge) the success of local operations initiated by the coordinator. The coordinator itself must be given the possibility to invoke the local execution of operations within a subsystem and to compensate operations that have already been committed. At the end of a coordination process the coordinator has to notify all agents. They can then discard all information (e.g. log information or compensating operations) they had to keep in their repository.

The crucial question for agents now is: What properties can an agent guarantee for local operations executed in its subsystem. As already mentioned, these properties depend on the underlying subsystem's willingness to cooperate, e.g., the repositories and the APIs (see Section 3.2).

The main property agents have to provide is the atomic execution of local operations as atomicity at coordination level is desired. The easiest way to realize this is to defer local operations until the end of the coordination process and to include them in global coordination transactions. But as we have seen in the previous section, this deferment may not always be possible. The agent therefore has to provide additional functionality on top of a subsystem.

To ensure local atomicity even in cases the subsystem does not support the atomic execution of operations, different possibilities exist. If the subsystem is a database management system, the agent may rely in the database's transaction mechanism, if not, it may guarantee local atomicity and correctness by realizing by itself a limited form of transaction management. Depending both on the activity and the subsystem, the agent can either guarantee that an operation is *retriable* or *compensatable* in order to ensure the atomic execution of this operation (this terminology is taken from the flexible transaction model [14]).

An operation is retriable if it can be invoked multiple times until its execution has finished successfully exactly once (this is for example crucial when we have to cope with network failures or failures of the subsystem itself). If an agent provides retriable local operations, the coordinator has to execute these operations exactly once; the agent is then responsible to repeat local invocations until they finally succeed.

A further way to ensure atomicity of coordination processes is to provide local operations that can be compensated even after they have successively been committed. If for a particular operation, the underlying subsystem realizes correct compensation (and if this compensation can be called via the API of the subsystem), the agent will just have to execute this compensation activity. If such a compensation does not exist, the agent again has to implement it by itself, for example by providing information about the semantically correct compensation activity of already committed local operations. This can be realized by persistently storing information about both initial operation and corresponding compensation operation(s) in the repository of the agent at least until the end of a coordination process. The property of compensating local operations after commitment is crucial in cases where operations can only be monitored after they are executed. In case of failures of subsequent coordination activities, this compensation is the basic mechanism to provide execution guarantees at coordination level anyhow.

When local operations have been observed (either before or after they have been committed), the agent has to decide whether they are of global relevance or not. An agent has a repository with information about the configuration of the whole system or at least with information whether an object of his subsystem has related objects elsewhere or not. Furthermore, information about the transformation of local data to the global data model (and vice versa to execute coordinator initiated operations) has to be available in the private database of the agent.

In some cases where the underlying subsystem is not able to store data persistently at a certain point of time, this can be done temporarily by the repository of the agent in order to ensure certain execution guarantees.

4 PROTOTYPE SYSTEM

Based upon the architecture described in the previous section, a prototype system has been developed. Therefore, two application systems have been chosen to be integrated into the coordination architecture: WorkManager [5], a PDM system (product data management), and SAP R/3 [4], a PPC system (production planning and control) that additionally covers various tasks of business engineering.

Both systems have in common that they are built on top of a relational database management system. However, as in both cases the database schema is unknown and no meta data is available, the respective agents cannot rely directly on the underlying databases as they do not get the semantics of user operations.

WorkManager provides a log file, where all operations are recorded. This information can be exploited by the WorkManager agent. But as operations can only be monitored after their successful completion, the agent has to provide additional information about the compensation of those operations within its repository. The execution of local operations initiated by the coordinator can be achieved by using the APIs provided by WorkManager.

The execution of operations of SAP R/3 by the agent is also possible via the API the subsystem provides. Predefined programs written in the integrated 4GL programming language ABAP/4 can therefore be called. Monitoring of operations can be achieved by extending internal applications (also ABAP/4 programs) to a call to the agent providing it with data necessary for a subsequent coordination process. This extension has the advantage that both synchronous coordination (to monitor an operation before it is committed and to defer it) and asynchronous coordination (an operation will not be deferred) can be achieved.

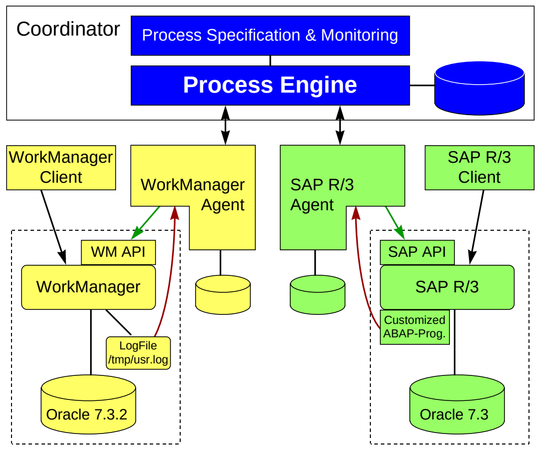

Figure 5: Prototype system incorporating SAP R/3 and WorkManager into the coordination architecture

These subsystems are connected via their agent to a global coordinator where inter-system dependencies and the reaction mechanisms to violations of overall consistency are managed and where coordination processes are controlled. The structure of the application systems and the way they are incorporated in the coordination architecture is depicted in Figure 5.

5 CONCLUSION

The contribution of this paper is twofold: We analyzed the properties of subsystems to figure out which database functionality can be added by special coordination agents in order to realize transactional execution guarantees of coordination processes and we elaborated on certain properties subsystems necessarily have to provide in order to integrate them into the coordination architecture we proposed.

Another emphasis of our work has been laid on the extension of transaction models to support coordination by decoupling transactional properties (spheres of atomicity and spheres of isolation) of coordination processes allowing to flexibly specify and enforce dependencies in composite systems.

A prototype system has been implemented that validates these concepts in the scenario of our industrial project partner. This prototype system includes the development of coordination agents of two subsystems in CIM: a product data management system and a production planning and control system. The agents prove the analysis of the prerequisites we presented for bringing database functionality to application systems.

In a related project, we investigate transactional workflows in a highly unstable and unpredictable distribution environment, like for example the Internet, to provide a framework for supporting virtual enterprises.

REFERENCES

[1] G. Alonso, S. Blott, A. Fessler, and H.-J. Schek. Correctness and Parallelism in Composite Systems. In *Proceedings of the ACM SIGMOD/PODS Conference on Management of Data*, Tucson, Arizona, May 12-15 1997.

[2] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Intergation of Heterogeneous Database Applications Through an Object-Oriented Interface. *Information Systems*, 14(5):407–420, 1989.

[3] V. Brosda. Data Integration of Heterogeneous Applications – a Technique for CIM-System Implementation. Technical Report TR 75.92.06, IBM Germany, Heidelberg Scientific Center, March 1992.

[4] R. Buck-Emden and J. Galimow. *Die Client/Server-Technologie des SAP-Systems R/3.* Addison-Wesley, 1996.

[5] CoCreate Software GmbH. *WorkManager, Release 3.5*, 1996.

[6] C. T. Davies. Data Processing Spheres of Control. *IBM Systems Journal*, 17(2):179–198, 1978.

[7] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabse Transaction Model for InterBase. In *Proceedings of the 16th VLDB Conference*, pages 507–518, Brisbane, Australia, 1990.

[8] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[9] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[10] S. Jablonski. Data Flow Management in Distributed CIM Systems. In *Proceedings of the 3rd International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pages 65–78, March 1992.

[11] S. Jablonski. Workflow-Management-Systeme: Motivation, Modellierung, Architektur. *Informatik Spektrum*, 18(1):13–24, 1995.

[12] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures.* Kluver Academic Publishers, 1997.

[13] W. Kim, editor. *Modern Database Systems: The Object Model, Interoperability and Beyond.* Addison-Wesley, 1995.

[14] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A Transaction Model for Multidatabase Systems. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993.

[15] M. Norrie, W. Schaad, H.-J. Schek, and M. Wunderli. CIM Through Database Coordination. In *Proceedings of the International Conference on Data and Knowledge Systems*, May 1994.

[16] Parametric Technology Corporation, Waltham, MA. *Pro/Engineer User Manual*, 1993.

[17] B. Reinwald. *Workflow-Management in verteilten Systemen: Entwurf und Betrieb geregelter arbeitsteiliger Anwendungssysteme.* Number 7 in Teubner-Texte zur Informatik. B. G. Teubner Verlagsgesellschaft, 1993.

[18] M. Rusinkiewicz and A. Sheth. *Specification and Execution of Transactional Workflows*, chapter 29. In: [13]. Addison-Wesley, 1995.

[19] A.-W. Scheer. *CIM – Towards the Factory of the Future.* Springer-Verlag, 3rd edition, 1994.

[20] H.-J. Schek, G. Weikum, and H. Ye. Towards a Unifying Theory of Concurrency Control and Recovery. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 300–311, June 1993.

[21] H. Schuldt, H.-J. Schek, and M. Tresch. Transactional Execution Guarantees for the Coordination of Subsystems. Technical report, Institute of Information Systems, ETH Zürich, 1998. In preparation.

[22] M. Wunderli. *Database Technology for the Coordination of CIM Subsystems.* PhD thesis, Swiss Federal Institute of Technology Zürich, 1996.

[23] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proceedings of the ACM SIGMOD Conference*, pages 67–78, 1994.