Applications Symposium (IDEAS'99), Montreal, Canada, August 1999.

Transactional Coordination Agents for Composite Systems*

Heiko Schuldt Hans-Jörg Schek Gustavo Alonso

Institute of Information Systems
Swiss Federal Institute of Technology (ETH)
ETH Zentrum, CH–8092 Zürich, Switzerland
{schuldt,schek,alonso}@inf.ethz.ch

Abstract

Composite systems are collections of autonomous, heterogeneous, and distributed software applications. In these systems, data dependencies are continuously violated by local operations and therefore, coordination processes are necessary to guarantee overall correctness and consistency. Such coordination processes must be endowed with some form of execution guarantees, which require the participating subsystems to have certain database functionality (such as atomicity of local operations, order-preservation and either compensation of operations or the deferment of their commit). However, this functionality is not present in many applications and must be implemented by a transactional coordination agent coupled with the application.

In this paper, we discuss the requirements to be met by the applications and their associated transactional coordination agents. We identify a minimal set of functionality the applications must provide in order to participate in transactional coordination processes and we also discuss how the missing database functionality can be added to arbitrary applications using transactional coordination agents. Then, we identify the structure of a generic transactional coordination agent and provide an implementation example of a transactional coordination agent tailored to SAP R/3.

1. Introduction

Coordination is necessary to keep track of dependencies between subsystems in environments involving distributed, heterogeneous, and autonomous subsystems. Local operations performed by users are not necessarily aware of side-effects and may violate consistency by introducing new data dependencies or making old ones disappear. A fundamental aspect of interoperability is, thus, to keep track of such dependencies and reestablish overall consistency whenever necessary. As different subsystems are involved and dependencies may be arbitrarily complex, we suggest to do this using coordination processes [29]. We also believe that transactional execution guarantees for coordination processes are crucial for this purpose. These transactional execution guarantees include correctness in case of failures (by partial compensation and appropriate alternative executions) and concurrency (when, for instance, different processes try to access shared resources simultaneously). In previous work, we have elaborated on correctness criteria for transactional coordination processes and identified the prerequisites to be met by the underlying subsystems in order to support transactional coordination [28]. There, coordination processes rely on the underlying subsystems to provide key transactional functionality, functionality which is not always present since we deal not only with database management systems (DBMSs) but with arbitrary applications. This functionality will be provided on top of the subsystems by *transactional coordination agents (TCA)*. The main contribution of this paper is the detailed discussion of the architecture and the principles of a generic TCA, the functionality TCAs have to provide as well as the basic prerequisites necessary to support this endeavor. Furthermore, we also present implementation issues of agent-based transactional coordination for a selected application system.

The paper is structured as follows: In section 2, we summarize transactional process management and motivate the process based coordination approach by a real-world scenario. Then, in section 3, we present the functionality needed for transactional coordination processes and discuss in detail how this can be provided by TCAs. In section 4, the implementation of a TCA for SAP R/3, an application system widely used for business management purposes, is presented. In section 5, related work is discussed; section 6 finally concludes the paper.

*Part of this work has been funded by the Swiss National Science Foundation under the project WISE (Workflow based Internet Services, http://www.inf.ethz.ch/department/IS/iks/research/wise.html) of the Swiss Priority Programme "Information and Communication Systems".

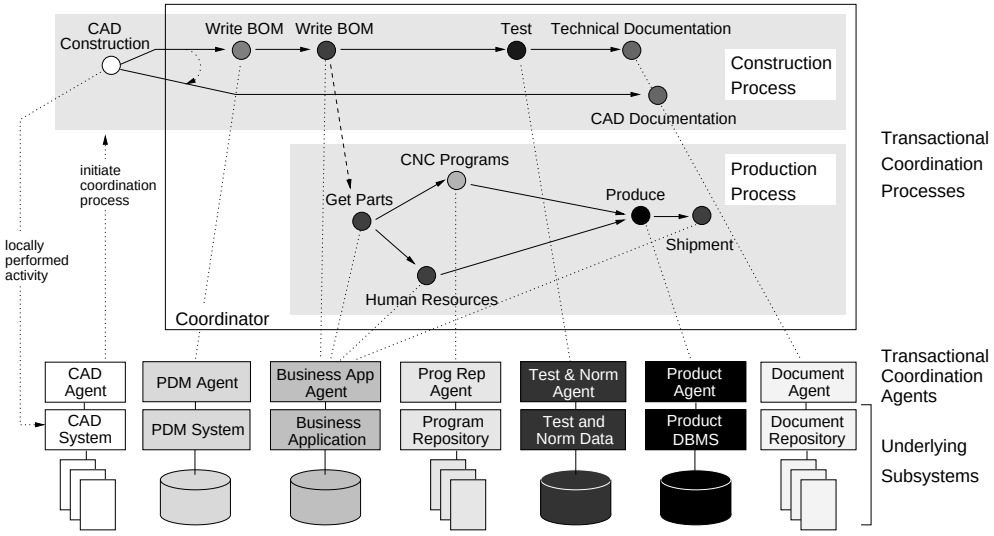

Figure 1. Transactional coordination processes in CIM environments

2. Transactional Coordination

In this section, we present a sample application for coordination in composite systems by transactional processes and motivate the necessity of transactional coordination agents to support this endeavor. In order to ensure readability independent of previous work, we will summarize the main ideas and aspects of transactional process management.

2.1. Motivation

Computer Integrated Manufacturing (CIM) environments often comprise a variety of different specialized application systems, such as computer aided design (CAD), product data management (PDM), or business application systems (as, e.g., SAP R/3) [22, 29]. In such environments, executions like the one shown in figure 1 are very common[1]. In this example, two processes are defined. A construction process is triggered by a local CAD construction operation; when the bill of materials (BOM) has been created, a second process, the production process, is started. In this case, production does not follow mass-production techniques but aims to customize each one of the products to deliver. Thus, the development of the product and its manufacture are strongly tied. Both processes run on a variety of subsystems, as shown in figure 1. The construction process contains all developing steps from the design of a new part to the final test and the subsequent technical documentation. It encompasses a CAD system, a product data management system (PDM), a test database as well as a technical documentation repository. The production process includes all manufacturing steps from the ordering of the necessary materials to the production floor including the necessary scheduling and the creation of computerized numerical control (CNC) programs.

These processes act both as the core business logic and as the basis for coordination of all the subsystems involved. Due to the complexity of the dependencies between all subsystems and because of the inherent heterogeneity of the composite systems, in general coordination cannot be realized by distributed transactions on top of all applications. Processes, in contrast, executed by a process support system [4] acting as coordinator provide the flexibility required for coordination. For instance, failures of single activities should not necessarily lead to a failure of the coordination effort. The possibility of executing alternatives in case of failures is an important characteristics of processes which is exploited by our coordination approach. In the following examples, we will show the benefits of transactional coordination processes and the requirements they impose on the subsystems to be coordinated.

Interaction In general, to support the execution of coordination processes, it has to be possible to interact with the underlying applications. For instance, local subsystems must be monitored to identify activities which violate dependencies. In our example, local activities executed within the CAD system have to be monitored in order to trigger the corresponding construction process. Similarly, the enactment of the coordination processes requires to be able to trigger activities at the underlying subsystems, for instance to compensate a given step, although concurrent local activities within the subsystem may exist.

[1] This example reflects the practice followed by one of our industrial partners in a recently concluded research project [29].

Correctness of Coordination Processes From a correctness point of view, although failures may occur and concurrency has to be considered, coordination processes must always terminate in a system-wide consistent state. Assume a failure during the construction process such that both a CAD construction and the associated BOM in the PDM system are created but, however, not the BOM in the business application system. This inconsistent state has to be avoided, as, due to the missing BOM in the business application, the respective part could never be produced.

Atomicity of Activities All activities executed by the coordinator have to be *atomic* in the sense that they are either executed completely or not at all in order to avoid situations where a coordination process ends up in an inconsistent state due to the undefined outcome of a non-atomic activity within a subsystem.

Compensation of Activities Correctness may be relatively easy to enforce if the underlying systems would offer the necessary functionality. For instance, some activities may be semantically compensatable after they have been successfully executed. If the respective subsystem provides a *compensation* activity, it can be exploited by the coordinator to deal with failures occurred during process enactment. Assume, for example, the "Test" activity of the construction process fails. In order to avoid inconsistencies, both "Write BOM" activities in the business application system and the PDM system have to be compensated. If compensation mechanisms exist, this is relatively easy to do. In this case, the complex objects representing the BOM have to be deleted in both systems. In some cases, restoring consistency is even easier, for instance, the "Get Parts" activity of the production process does not cause changes in the business application system and does not have to be compensated. However, unlike in database systems where all actions are undone as part of the rollback of a transaction, coordination processes should go beyond the "all-or-nothing" semantics.

Guaranteed Termination Compensation may not be provided by all subsystems or we may not want to compensate all activities of a process after a failure occurred. Suppose the test of a newly designed product fails. Although the BOMs have to be deleted (thus, the corresponding activities are compensated), it is wise to keep all work done within the CAD construction. Therefore, the construction step is not removed as it would occur if compensation were to be used but it is documented (as depicted in figure 1). Such behavior can be captured by replacing atomicity of coordination processes by the notion of *guaranteed termination* [28], i.e., the guarantee that the process will always reach a consistent state when it terminates.

Order Preservation Also, only consistent interaction between coordination processes can be allowed. In the example, the production process runs in parallel to the construction process in order to minimize delays. But since the two activities "Write BOM" and "Get Parts" conflict, arbitrary interleavings of both processes need to be avoided. The coordinator can easily establish these interleavings by ordering operations accordingly but this order needs to be respected by the underlying systems. One way to do so is to enforce *order preservation* so that the execution at the subsystems can be parallelized while matching the one indicated by the coordinator.

Commit Deferment Furthermore, interaction between the processes need to be closely watched. For instance, the "Produce" activity cannot be undone. As a result, it may not be executed before the construction process succeeds. Otherwise, a failure within the "Test" activity, for example, would lead to the compensation of the "Write BOM" activities and thus to the invalidation of the data the product has been built from. This would cause severe inconsistencies in the system as no valid construction and BOM for the part exists. The commit of the "Produce" activity therefore has to be *deferred* to avoid these incorrect interleavings of concurrent processes that cannot be resolved by compensation.

Retriability All activities that cannot be compensated effect the subsequent activities of the same process. Suppose the correct execution of the "Produce" activity of the production process. Suppose further the failure of the next activity, the shipment of products, for instance due to a temporary broken network connection to the business application managing the relevant customer addresses. Then, as not all previous activities can be undone, the production process would end up in an inconsistent state. Therefore, it has to be guaranteed that all activities of a process following a non-compensatable activity can be invoked repeatedly and succeed exactly once. In the example, after the recovery of the network connection, the customer address must be available.

2.2. Transactional Process Management

The notion of *transactional process management* has been suggested as a means to coordinate different subsystems [9, 3, 28]. The main components of transactional process management consist of a coordinator acting as top level scheduler and several transactional coordination agents (TCAs) —one for each participating subsystem— acting as lower level schedulers (as depicted in figure 1). Coordination processes encompass *activities* which are invocations in subsystems scheduled by the coordinator. These activities

differ in terms of their termination guarantees: they are either *compensatable, retriable*, or *pivot* (as in the flex transaction model [21, 35]). In contrast, local activities are the ones directly executed by local users within the subsystem and are in general not known to the coordinator[2]. The coordinators' task is to execute transactional processes in order to reestablish system-wide consistency when it has been violated by local activities. Even in case of failures and of concurrent access to shared resources, the coordinator must provide certain execution guarantees for these processes. Firstly, these execution guarantees include guaranteed termination, a more general notion of atomicity than the standard all or nothing semantics which is realized by partial compensation and alternative executions. Secondly, the correct parallelization of concurrent processes is required.

This paper continues the work presented in [3] with respect to process-based coordination. Starting with this idea, we focused in our previous work on the provision of execution guarantees for transactional processes and elaborated a correctness criterion, prefix-reducibility (PRED; details can be found in [28]). This correctness criterion addresses concurrency control and recovery simultaneously while considering the special structure that can be found in transactional processes. Furthermore, it also allows a high degree of parallelism by applying the ideas of the composite systems theory [1]. For the following discussion, it is however only important to understand the key aspects of transactional process management: The coordinator acts as a kind of transaction scheduler that is more general than a traditional database scheduler in that it i.) knows about semantic commutativity of activities, ii.) knows about properties of activities (compensatable, retriable, or pivot), and iii.) knows about alternative executions paths in case of failures. Based on this information, the coordinator ensures global correctness but only under the assumption that the activities within the coordination themselves provide the required functionality as discussed above in section 2.1. Therefore, certain database functionality is required at the underlying subsystems. Since we do not expect all subsystems to be databases, in order to deploy coordination processes, a *transactional coordination agent* is placed on top of each subsystem [29] as depicted in figure 1 so that the subsystem together with its TCA can be seen as a database and treated accordingly.

3. Transactional Coordination Agents

In what follows, we identify the functionality required to apply correct transactional process management with respect to the PRED criterion and discuss how it can be provided by a TCA. Furthermore, we identify which requisites the applications must meet in order to allow TCAs to implement the missing functionality. While the atomicity of activities as well as compensation and retriability are required for the provision of guaranteed termination of single transactional coordination processes, a generalized notion of the "all-or-nothing" semantics of atomicity, the deferment of commits and the order preservation are required in order to support correct concurrency control of transactional coordination processes. Although all approaches to support the different required properties are discussed independently, it has to be noted that some of them are closely related. A single feature of the underlying system sometimes allows the TCA to implement and to enforce more than one property.

3.1. Atomicity of Activities

Motivation: During local execution of activities, failures may occur. Both site and application failures may lead to undefined states where only some parts of an activity may have been executed. In order to avoid these undefined states, it has to be guaranteed that all activities are executed atomically in the sense that they are either executed completely or not at all (thus, failed activities must not leave any side-effects).

Approaches: Although atomicity can be ensured by certain subsystems, there also exists the possibility for TCAs to take over this task.

Subsystem solution: Ideally, atomicity is directly provided by the respective subsystem. This is the case for DBMSs but also for applications supporting the notion of transactions (e.g., SAP R/3 [25]) or providing only atomic service invocations. For these applications, no further work needs to be done by the TCA.

TCA solution: When a subsystem supports non-atomic activities, two prerequisites have to be met in order to allow its TCA to provide atomicity for activities executed by the coordinator: The subsystem first has to provide the necessary information about what has already been executed by a failed activity until the failure occurred, and second, the required interfaces to allow the coordination agent to undo all effects of the failed activity have to be available.

For the first requirement, log or trace files of the application are used. There, not only the single operations executed on data elements within an activity must be recorded but also the operations making these changes persistent. The PDM application WorkManager [10], for instance, which is based on a relational DBMS, persistently logs all SQL statements performed during an

[2] Aside of (global) and local activities, we also use the notion of operation. Operations are executed on the underlying data source of a subsystem as part of an activity.

activity in a specific trace file. When a failure occurs, the undo operations have to be executed only when a commit operation has been logged. The file based engineering application Pro/ENGINEER [23] provides a similar trace recording all executed file operations.

The second requirement is based on the observation that interaction with applications must always take place via their API. In general, the business logic of an application is not mapped to the underlying data source but is performed together with integrity checks at the application level. As operations executed directly at the DBMS level or at the data level are not aware of such constraints, they may have unknown side-effects leading to a violation of these application-specific constraints. Also, due to the inherent complexity of the applications to be considered, it is in most cases not possible to determine the appropriate operations at the data source level (e.g., since the schema of the underlying DBMS is totally unknown[3]) corresponding to an activity.

A special TCA solution to provide atomicity can be implemented when the underlying application system performs some kind of version management. In this case, the TCA does not have to know each local operation of a failed activity (requirement 1) but rather which previous version has to be recovered (req. 2).

3.2. Compensation

Motivation: Certain activities may be compensatable, i.e., can be semantically undone after they have been executed correctly[4]. When a failure occurs during the enactment of a coordination process, if backward recovery is possible (no non-compensatable activity has already been executed), compensation of all previous activities in reverse order is performed for recovery purposes.

Approaches: In general, two different approaches for the determination of compensating activities exist. In the registration approach, after appropriate configuration, activities for semantically compensating a given activity can be defined. The undo approach is, in contrast, more limited as it only allows compensation by step-wise executing undo operations.

[3]The underlying relational DBMS of SAP R/3 contains more than 8600 tables and thus, reasoning about the application semantics of operations monitored at DBMS level is not at all possible. Also, as business logic is not mapped to the DBMS, direct access to SAP R/3's DBMS may have dramatic consequences for the systems' consistency.

[4]This is similar to the semantical undo operations exploited within the saga model [13]. However, in the saga model, a compensation has to be available for each subtransaction whereas in the more general transactional process management, following the flex transaction model [21, 35], activities may exist that cannot be compensated.

Registration: When a subsystem is integrated into transactional coordination process management, a compensating activity has to be registered for each available compensatable activity and to be stored persistently together with the initial activity. Thus, for the management of metadata and also for logging purposes, each TCA requires a DBMS. When an activity has to be executed, the TCA has to perform logging in order to provide the necessary information for a subsequent invocation of its compensating activity. Considering the CIM example described in section 2.1, when the "Write BOM" activity is executed, the TCA of the PDM system has to log persistently which objects have been created in order to determine –together with the previously registered information about how to delete BOM entries– the correct compensation. This registration approach is exploited, for example, by the TCA implemented for SAP R/3 (see section 4).

Undo approach: Alternatively, when an application provides a log file or a trace file, resp., where all operations executed within an activity are logged, this information can be exploited for successively compensating the single operations performed within this activity by means of undo operations. Again, these undo operations must be available via the subsystem's API. Compensation by re-installation of previous versions also falls in this category.

3.3. Retriability

Motivation: When a non-compensatable activity of a coordination process has terminated successfully, backward recovery by successively compensating all previously executed activities is no longer possible. Therefore, to support the guaranteed termination property of coordination processes, it has to be ensured that all activities following a non-compensatable activity terminate successfully. However, since temporary failures have to be considered (e.g., due to network problems or the non-availability of a subsystem), these activities may have to be executed repeatedly until they finally succeed. The property of an activity to succeed exactly once after an arbitrary number of invocations is subsumed under the notion of retriability which is borrowed from the flex transaction model [21, 35].

Approaches: In general, the requirements to support the retriability of activities are not directly met by applications. However, it has to be guaranteed by the subsystem that such an activity terminates successfully. In contrast, certain failures occurring on invocation can be intercepted by the TCA.

Persistent Queues: When subsystems support persistent queuing mechanisms, it can be guaranteed that activi-

ties do not get lost. When network failures have to be considered, activities must be repeatedly invoked until they are put in the application's persistent queue.

Exactly once guarantee: Some subsystems guarantee that activities invoked repeatedly with the same identifier are executed exactly once which can then be exploited by the associated TCA. This property is, for example, available with SAP R/3's transactional remote function call (tRFC).

Repeated execution: When neither persistent queuing mechanisms nor exactly once guarantees are provided by the subsystem, the TCA has to execute an activity repeatedly until it succeeds. It must guarantee however that the effects of a failed activity are always undone before it is scheduled for the next time.

3.4. Commit Deferment

Motivation: When executing transactional coordination processes in parallel, consistent interaction between these processes has to be guaranteed. In the case of failures, non-compensatable activities are a problem. Consider, for instance, the CIM example presented in section 2.1. As dependencies between the construction and the production process being executed in parallel exist, the non-compensatable activity "Produce" can not be committed before the termination of the construction process. Otherwise, inconsistencies may arise as, for recovery purposes, the compensation of the construction process would also require the compensation of "Produce" which, however, cannot be performed.

Approaches: To be able to defer the commit of a local operation, the associated subsystem either has to support the two phase commit protocol [6] or it has at least to provide the functionality to emulate it. Otherwise, only the blocking of processes is possible.

Subsystem solution: The commitment of each non-compensatable activity has to be be deferred until it is safe to proceed. This imposes the strongest requirements on subsystems: they have to support the XA interface of the X/Open distributed transaction processing standard [15]. When this functionality exists, a two phase commit (2PC) protocol can be used to commit all non-compensatable activities of a process. When all subsystems agreed to commit successfully after the prepare-to-commit message, the global commit determined by the coordinator can be deferred according to the restrictions of the correctness criterion of transactional coordination [28].

2PC agent method: If subsystems do not directly support the XA interface but provide strict schedules[5] [6], 2PC functionality can be added to them by their TCA following the 2PC agent method [33].

Blocking solution: The restrictions on the execution of non-compensatable activities can be trivially achieved by blocking a process whenever a non-compensatable activity has to be scheduled. However, although this can be achieved for any subsystem, it leads in the worst case to a serialization of the coordination processes which is not a feasible solution.

Although the deferment of commits imposes the strongest prerequisites of subsystems, it has to be noted that they have only to be met when compensation of activities is not possible.

3.5. Order Preservation

Motivation: When concurrent processes access shared resources, conflicts between activities of different processes may occur. Two activities of different processes conflict if the order of their execution matters, i.e., when their return values are different for different orders of execution [27]. At the coordinator level, conflicting activities therefore have to be ordered. The coordinator imposes the so-called "weak" order borrowed from the composite systems theory [1]. It simply requires that the serialization order of two parallel activities does not contradict the imposed weak order. Thus, the weak order between two conflicting activities allows parallel execution as long as the result is the same as if both activities would have been executed sequentially.

Approaches: In the two possible approaches, this weak conflict order is either applied directly or it is transformed into a sequential order.

Weakly ordered execution: An ideal subsystem directly supports the weakly ordered execution of conflicting activities. A DBMS implementing, e.g., commit-order-serializability protocols [5] is such an ideal subsystem. In this case, the TCA can directly pass the conflicting activities to the subsystem with the commit order derived from the weak conflict order.

Sequential execution: If commit-order-serializability is not supported, the sequential execution of weakly ordered activities must be enforced. For all such systems, the associated TCA acts as a rather primitive scheduler which invokes conflicting activities sequentially.

[5] Strict schedules are provided, for example, by systems implementing a strict two phase locking (S2PL) protocol [6]. This is not only possible for DBMSs but also for application systems supporting locking functionality (e.g., SAP R/3 where the request and release of locks can be defined by application programmers, or check-in/check-out based CAD systems).

Although the transformation of the weak order in a sequential one decreases the degree of parallelism achieved, it can be done with any application. While order preservation is the least restrictive property to be provided by the subsystems and their TCAs, sequential execution suffices to guarantee correctness with any underlying application.

3.6. Local Concurrency

Motivation: Autonomy of the subsystems to be coordinated is an important issue as local users are normally not aware of dependencies that exist and thus of coordination that has to be performed. Therefore, two kinds of activities do exist within subsystems but must not lead to inconsistencies: Local activities executed by local users via the subsystems' GUI or API (these activities are not known to the coordinator) and global activities scheduled by the coordinator and executed by the TCA via the API of the subsystem. Thus, aside of concurrency control at the coordinator level, local concurrency within each subsystem has also to be considered.

Requirements: To support both global and local activities, all subsystems have to provide multi-user functionality. Ideally, a subsystem supports multiple parallel connections to its associated TCA (this is especially required when the weak conflict order is exploited) while allowing additional local users to work concurrently. In general, however, when activities are executed sequentially by the TCA, one connection via the subsystem's API is sufficient. Furthermore, each subsystem has to provide a limited form of concurrency control in order to avoid that local and global activities are executed on the same object simultaneously (e.g., check-in/check-out mechanisms ore more sophisticated locking protocols).

Approaches: Correctness in the presence of both local and global activities can be guaranteed either by the coordinator or by each TCA locally.

Coordinator approach: When the coordinator is notified by the TCA about local activities, correctness can be guaranteed by establishing a weak order between all global activities and the local activities within the same subsystem. The TCA then has to enforce this weak order.

TCA approach: When the coordinator is not aware of local activities, correctness has to be enforced by the TCAs. Each TCA then has to serialize global and local activities whereas the commit of global activities is deferred against all local activities while isolation is given up against all other global activities [26].

Blocking approach: Correctness of local and global activities can also be guaranteed when the local commit of each global activity is deferred until the commit of its associated transactional coordination process. This is the most restrictive solution but imposes only the elementary prerequisites to a subsystem and its TCA.

3.7. Interaction with Subsystems

Motivation: Aside of the database functionality to be provided on top of the subsystems, TCAs also have to provide the basic functionality to interact with the underlying applications. Since coordination processes have to be initiated after local activities violate global constraints or introduce new dependencies, TCAs have to perform monitoring to detect these local activities. In order to participate in transactional process management, all subsystems have to support the monitoring task and thus to provide the necessary information that can be exploited by the associated TCAs. Also, as already discussed in section 3.1, all subsystems have to provide an API offering all activities specified within the coordination processes.

Approaches: Monitoring can be supported by the subsystem either actively or passively [29]. The first approach, the active monitoring, is the most appropriate one. Since the passive approaches are very limited, they should only be considered when no active support is provided.

Active Monitoring: To support active monitoring at application level, a subsystem has to provide the possibility to pass control to its TCA by an appropriate call interface. Additionally, for these purposes, either trigger mechanisms (e.g., Pro/ENGINEER) have to be available or the customization of application logic (e.g., SAP R/3) has to be possible. In general, these mechanisms can be applied for all local activities; the TCA has then to filter the ones affecting global consistency. When a local operation is committed, these mechanisms enable notify the associated TCA which, in turn, forwards this notification to the coordinator. Ideally, control is passed to the TCA before a local activity terminates. These mechanisms then also allow to defer the commitment of local activities until it is safe to terminate them. Although it is based on the strongest prerequisites, the active monitoring approach is best suited for coordination purposes as it provides the full semantics of the activities to be observed.

Passive Monitoring: Passive monitoring does not impose any prerequisites to the application system as it is performed on the underlying data sources. Changes are detected by comparing a current snapshot of the data sources with a previous version or, alternatively,

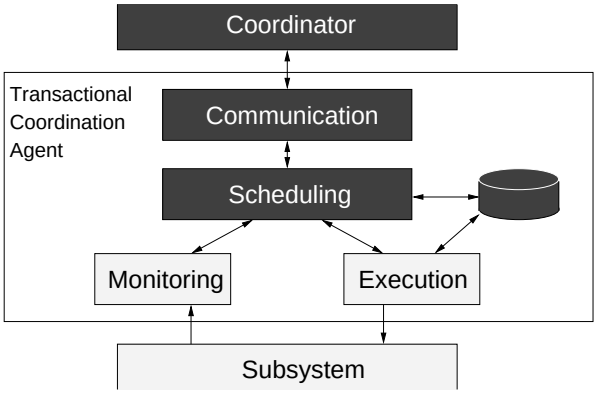

Figure 2. Structure of a generic TCA

by exploiting information from log files or trace files when available. However, aside of the asynchronicity of the passive monitoring approach, the semantics of an activity causing changes within the underlying data sources may not be captured. Thus, this approach is only suited for applications with rather simple and known schemata.

3.8. Structure of Generic TCA

With respect to the functionality that has to be provided by TCAs, four different modules of a generic TCA can be identified [34]: communication, scheduling, monitoring and execution (figure 2).

Monitoring The monitoring module covers the extraction of local operations out of the underlying application. As for this, existing interfaces or trigger mechanisms of the subsystem have to be exploited, the monitoring module has to be tailored to this subsystem.

Execution Activities specified by the coordinator have to be mapped to local operations. Again, subsystem-specific interfaces have to be exploited and thus, the execution module has to be tailored as well to the underlying subsystem. This is also true for all invocations of subsystem operations both for compensation purposes and for the provision of atomicity by step-wise undoing the effects of failed activities.

Scheduling Furthermore, scheduling of local operations with respect to the activities specified by the coordinator has to be performed by the scheduling module. This includes the preservation of the given orders as well as the provision of local atomicity, the guaranteed availability of compensating activities, and eventually the deferment of local commits. For these purposes, the TCA uses a DBMS for persistent logging of the activities executed within the subsystem together with the associated parameters for compensation purposes and the storage of necessary metadata. This includes, for example, the compensating activities registered during configuration. If the underlying subsystem is based on a database and if the TCA has access to this DBMS, it can be exploited; otherwise, a separate DBMS has to be made available for the TCA.

Communication Finally, a common communication protocol has to be supported for the interaction between the coordinator and the TCAs.

3.9. Analysis of Subsystem Properties

The discussion of the previous sections is summarized in table 1 where both the properties of an ideal subsystem from the point of view of the requirements of transactional process management and the minimal set of properties to be met by subsystems are listed. These basic properties are required to allow TCAs to enhance subsystems with the functionality necessary for participating in transactional coordination processes.

4. Transactional Coordination Agents in Practice: TCA for SAP R/3

SAP R/3 is one of the most commonly used application systems for business management purposes. It consists of specialized modules for certain application areas (e.g., production planning, logistics, or human resource management) and thus plays an important role in CIM areas as presented in section 2.1. The system is built in a client/server architecture and is based on a relational DBMS [7].

As a proof of concept of the ideas presented in this paper, a transactional coordination agent for SAP R/3 has been implemented [30]. The architecture of this TCA is depicted in figure 3. The agent specific parts are colored in light gray whereas the standard SAP R/3 system is depicted in white color. It is tightly integrated into the core system as, for instance, R/3's underlying database is used for the management of the TCAs' metadata and is accessed for this purpose by the standard DBMS interface of SAP R/3. This SAP R/3 TCA is very general in nature as it not only supports the execution of arbitrary functions within SAP R/3 but also complete workflow processes within the system. The kernel of the TCA is itself a SAP workflow process (meta process) in which the function or the workflow to be executed is embedded in a generic way by simply passing its name as parameter to the meta process. In what follows, we describe how the required TCA functionality identified in section 3 is implemented for this SAP R/3 agent.

	Ideal subsystem	Basic requirements
Atomicity	Atomicity is provided for all activities	Subsystem provides log files; appropriate operations available via API
Compensation	Compensation activities are provided via API; registration during configuration	Subsystem provides log files; operations for step-wise undo available via API
Retriability	Provides persistent queues or exactly once guarantee; activities must lead to success	Activities must lead to success; atomicity to be provided by TCA (repeated invocation)
Commit Deferment	Support of 2PC protocol (XA interface)	None, if all activities are compensatable, otherwise strict schedules
Order Preservation	Support of commit-order-serializability	None (sequential execution of coordination activities)
Concurrency	Multiple connections via API; guarantee of correct concurrency of local and global activities	One TCA connection via API; guarantee of correct concurrency of local and global activities
Execution	All activities are available via the subsystem's API	All coordination activities are available via the subsystem's API
Monitoring	Pass control to TCA; filtering performed within subsystem	Pass control to TCA for all activities (only in special cases: passive monitoring approach)

Table 1. Summary of subsystem properties to be met for transactional coordination processes

Interaction Communication with the coordinator takes place by exploiting the remote function call[6] (RFC) of SAP R/3 which is available via a set of C library functions. It supports both multiple parallel calls of R/3 functions from external applications as well as calls of external applications from within the R/3 system via the SAP gateway. The communication module (RFC library adapter) is thus a thin software layer transforming requests from the coordinator into RFC calls and vice versa. For execution purposes, the meta workflow of the TCA is called with a specification of the name of the function or process to be executed. In order to support the monitoring task, the R/3 transactions relevant for coordination purposes that are invoked by a local user have been slightly modified by adding a RFC call to the coordinator which can either be performed synchronously or asynchronously. This customization is possible because the ABAP/4 sources[7] of all R/3 transactions are available.

Atomicity SAP R/3 supports the notion of transactions and guarantees ACID properties for them. When a single function has to be executed as activity given by the coordinator, atomicity is guaranteed. When a complete workflow process has to be executed, however, appropriate failure handling mechanisms have to be implemented within this process to do rollback in the case of failures in order to provide the required all-or-nothing semantics of activities.

[6] SAP's implementation of the remote procedure call (RPC).

[7] ABAP/4 (Advanced Business Application Programming Language) is the 4^{th} generation programming language the greatest part of the R/3 system — aside of a small C kernel — is implemented in.

Compensation In general, the meta workflow encompasses only two activities: the function or process to be executed and its associated compensation. It thus follows the idea of an explicit registration of the compensation but has an essential advantage. In SAP R/3, the instances of workflow processes together with the associated parameters are stored persistently in the underlying DBMS. Thus, no additional effort has to be taken to log the parameters of an activity executed by the coordinator. After an activity (a function or a workflow process) of the meta workflow has been executed successfully, the meta process performs an idle wait, thus does not consume any resources. In the case the coordinator determines the necessity of compensation, an internal event in SAP R/3 is generated via a RFC library call and the next activity of the meta workflow, the compensation, can be executed without explicitly specifying the required parameters. Otherwise, when compensation no longer has to be considered, the meta process is terminated.

Order Preservation The requirement of order preservation is also met by the R/3 system although execution of transactions with respect to the weak conflict order is not supported. Each application object to be updated is in general exclusively locked during a R/3 transaction and thus, conflicting transactions are serialized in commit order.

Retriability The requirement of retriability of activities is achieved by exploiting SAP R/3's transactional remote function call (tRFC) mechanisms. After the RFC library

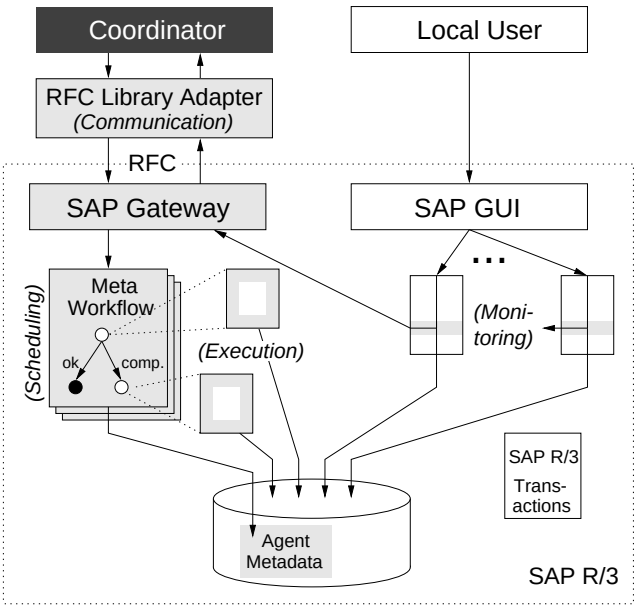

Figure 3. TCA for SAP R/3

adapter of the TCA has requested a tRFC identifier from the R/3 system, all invocations of a retriable activity are performed with this ID and the R/3 system guarantees that is is executed only once although multiply invoked.

Commit Deferment The deferment of commits with respect to the 2PC protocol is not available for external RFC calls (but only for RFC calls within the same R/3 system). However, this imposes no restrictions as compensation can be provided for all R/3 activities.

5. Related Work

The term *agent* is frequently used in a variety of contexts. Although there is no formal definition, we rely on a definition synthesized from several others [17]: *Agents are active, persistent (software) components that perceive, reason, act, and communicate.*

This definition is very general and subsumes the various directions and specializations of agent technology. Agents may be adaptive, mobile (itinerant), autonomous, or even intelligent, all comprised within the reasoning component of the agent. The ability to both act and perceive allows an agent to interact with its environment. For communication, various protocols such as, for instance, KQML [11], have been developed.

A very strong research direction in agent technology is the use of agents to facilitate interoperability in complex systems [14], however with the focus on query processing and without considering execution guarantees for this endeavor. In these approaches, agents model the resources they are associated with and use a common ontology for semantic integration [19]. Local models made known to other agents form the basis for negotiation among different agents and for the integration of models at a semantically higher level of abstraction. However, as in schema integration for federated database management systems [31], individual models have to be related to each other and incompatibilities resolved. Once this is done, interoperability can be achieved based on the ability of the agents to communicate with each other. This kind of agent technology exploits concepts also used for, e.g., mediators [32], and wrappers [12, 8]. Application agents as defined within the workflow reference model of the workflow management coalition [16] are used in a very similar way.

The provision of a common interface on top of heterogeneous subsystems is a fundamental task of coordination agents. Although our TCAs provide this functionality, schema integration is beyond the scope of this paper. We concentrate on the provision of database functionality even if this is not directly supported by the underlying applications. Thus, our TCAs are similar to two phase commit (2PC) agents proposed for federated database management systems [33] but provide considerably more database functionality than just atomic commitment.

6. Conclusion

In this paper, we have discussed the necessity of transactional coordination agents in order to enforce consistency in composite systems consisting of different heterogeneous, autonomous, and distributed applications. Coordination is performed based on processes defined on top of all application systems. To support execution guarantees of coordination processes, the underlying applications are required to provide transactional functionality. However, many systems do not meet these requirements. Thus, they must be enhanced by TCAs which, in turn, have to provide the missing functionality.

Based on a correctness criterion for transactional processes we have elaborated as part of previous work, we have identified both the properties that have to be provided by applications or their TCAs from the point of view of the coordinator as well as the minimal set of properties of applications in order to allow TCAs to implement the missing functionality. Then, we have discussed how subsystems with given properties can be enhanced by TCAs. This detailed classification provides the necessary information in order to first determine whether transactional coordination processes in composite systems with given applications can be applied and second, which functionality is necessary and how the required TCAs can be built. Finally, with the SAP R/3 TCA we have presented one out of several TCAs that

have been implemented for applications in the area of computer integrated manufacturing.

Our transactional coordination approach has, in addition to the CIM scenario discussed throughout this paper, also been applied to geographical information systems [24]. In future work, we will apply this coordination approach within the WISE project of ETH Zürich [2] to applications in the area of electronic commerce.

References

[1] G. Alonso, S. Blott, A. Feßler, and H.-J. Schek. Correctness and Parallelism in Composite Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'97)*, Tucson, Arizona, May 12-15 1997.

[2] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to Business E-Commerce. In *Proc. of the 9^{th} Int. Workshop on Research Issues in Data Engineering (RIDE-VE'99)*, Sydney, Australia, Mar. 1999.

[3] G. Alonso, C. Hagen, H.-J. Schek, and M. Tresch. Distributed Processing over Stand-alone Systems and Applications. In *Proc. of the 23^{rd} VLDB conference*, Aug. 1997.

[4] G. Alonso and C. Mohan. *Workflow Management: The Next Generation of Distributed Processing Tools*, chapter 1. In: [20]. Kluwer Academic Publishers, 1997.

[5] C. Beeri, P. Bernstein, and N. Goodman. A model for concurrency in nested transaction systems. *Journal of the ACM*, 36(2):230–269, Apr. 1989.

[6] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[7] R. Buck-Emden and J. Galimow. *SAP R/3 System: A Client/Server Technology*. Addison-Wesley, 1996.

[8] M. Carey, L. Haas, P. Schwarz, et al. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Proceedings of the 5^{th} Int. Workshop on Research Issues in Data Engineering (RIDE-DOM'95)*, Mar. 1995.

[9] Q. Chen and U. Dayal. A Transactional Nested Process Management System. In *Proc. of the 12^{th} Int. Conference on Data Engineering (ICDE'96)*, pages 566–573, 1996.

[10] CoCreate Software GmbH. *WorkManager, Rel. 3.5*, 1996.

[11] T. Finin, R. Fritzson, and D. McKay. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Nov. 1994.

[12] H. Garcia-Molina et al. The TSIMMIS Approach to Mediation: Data Models and Languages. In *Proceedings of the 2^{nd} International Workshop on Next Generation Information Technology Systems (NGITS'95)*, 1995.

[13] H. Garcia-Molina and K. Salem. Sagas. *ACM SIGMOD Record*, 16(3), 1987.

[14] M. Genesereth and S. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, July 1994.

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

[16] D. Hollingsworth. *Workflow Management Coalition: The Workflow Reference Model*. Workflow Management Coalition, Dec. 1993. Document TC00-1003.

[17] M. Huhns and M. Singh. *Agents and Multiagent Systems: Themes, Approaches and Challenges*, chapter 1, pages 1–27. In: [18]. Morgan Kaufmann, 1998.

[18] M. Huhns and M. Singh, editors. *Readings in Agents*. Morgan Kaufman Publishers, 1998.

[19] M. Huhns, M. Singh, and T. Ksiezyk. Global Information Management via Local Autonomous Agents. In *[18]*, pages 36–45. Morgan Kaufmann, 1998.

[20] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer, 1997.

[21] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A Transaction Model for Multidatabase Systems. In *Proceedings of the 12^{th} Int. Conference on Distributed Computing Systems (ICDCS'92)*, pages 56–63, June 1992.

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

[23] Parametric Technology Corporation, Waltham, MA, USA. *Pro/ENGINEER User Manual*, 1993.

[24] L. Relly, H. Schuldt, and H.-J. Schek. Exporting Database Functionality – The CONCERT Way. *IEEE Data Engineering Bulletin*, 21(3), 1998. Special Issue on Interoperability.

[25] SAP AG. *SAP R/3 Online Documentation*, 1996.

[26] W. Schaad. *Transactions in Heterogeneous Federated Database Systems*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zürich, 1996. In German.

[27] 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 (PODS'93)*, pages 300–311, June 1993.

[28] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, Philadelphia, Pennsylvania, USA, May 31-June 2 1999.

[29] H. Schuldt, H.-J. Schek, and M. Tresch. Coordination in CIM: Bringing Database Functionality to Application Systems. In *Proc. of the 5^{th} European Concurrent Engineering Conference (ECEC'98)*, Erlangen, Germany, Apr. 1998.

[30] C. Schuler. Design and Development of a Coordination Agent for the Integration of SAP R/3 in Workflow Processes. Diploma thesis, Database Research Group, Institute of Information Systems, ETH Zürich, July 1998. In German.

[31] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comp. Surveys*, 22(3):183 – 236, 1990.

[32] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[33] A. Wolski and J. Veijalainen. 2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase. In *Proc. of the IEEE PARBASE'90 Conference*, pages 321–330, Mar. 1990.

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

[35] 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.