Maintaining Global Integrity in Federated Relational Databases using Interactive Component Systems

Christopher Popfinger and Stefan Conrad

Institute of Computer Science
University of Düsseldorf
D-40225 Düsseldorf, Germany
{popfinger, conrad}@cs.uni-duesseldorf.de

Abstract. The maintenance of global integrity constraints in database federations is still a challenge since traditional integrity constraint management techniques cannot be applied to such a distributed management of data. In this paper we present a concept of global integrity maintenance by migrating the concepts of active database systems to a collection of interoperable relational databases. We introduce *Active Component Systems* which are able to interact with each other using direct connections established from within their database management systems. Global integrity constraints are decomposed into sets of *partial integrity constraints*, which are enforced directly by the affected Active Component Systems without the need of a global component.

1 Introduction

Many organizations or companies own a multitude of data sources, which generally emerged autonomously to fit the needs of a department or a certain group of users at a local site. Local applications produce or modify data that is often semantically related to data stored on a different autonomous source. One of the main challenges in the integration of data in such environments is the autonomy of the data sources. This autonomy implies the ability to choose its own database design and operational behavior. Local autonomy is tightly attached to the data ownership, i.e. who is responsible for the correctness, availability, and consistency of the shared data. Centralizing data means to limit local autonomy and revoke the responsibility from the local administrator, which is not reasonable in many cases. A federated architecture for decentralizing data has to balance both, the highest possible local autonomy and a reasonable degree of information sharing [1]. Hence, the architecture of a company wide information system has to be applicable to the data policy of the company and vice versa. To be more precise, the question of data ownership determines the composition of the company wide information platform, while it has to ensure a high level of consistency and fail-safety.

In many scenarios, local data is, and should be, exclusively manipulated by local applications, whereas global applications are used to display and analyze

this data without the need for global write transactions. For example, a company could store employee data in a database of the management department, whereas the research department maintains information about ongoing projects and researchers in its own database. Each department is responsible for the up-to-dateness and correctness of its data. Information about employees and research projects shall be integrated and displayed on the company's website. Obviously the data in both autonomous databases is interrelated, since all the researchers are employees of the company. Thus, we need to ensure that every entry in the research database has a corresponding entry in the management database. Interdependencies of data stored on multiple databases can be considered as integrity constraints expressed over a global schema that is derived from relevant schemata of the local databases. The maintenance of these global integrity constraints is still a problem since traditional integrity constraint management techniques cannot be applied to such a distributed management of data. Our work is developed in the context of relational databases, since this type of data source is widely used for data storage in practice. We assume an information system which comprises a collection of autonomous relational sources of various vendors running on different platforms. The databases store interdependent data that is accessed by local and global applications.

In this paper we present a concept for global integrity maintenance in such a federated relational database systems by extending the concepts of active database systems to a collection of interoperable relational databases. We introduce *Active Component Database Systems* (ACDBS), which are able to interact with each other using direct database connections established from within their database management systems. Interdependencies between the data sources are expressed as global integrity constraints and enforced using constraint checks that are entirely implemented on the ACDBSs without the need of a global component or federation layer. At the same time, we allow the component database systems to retain the greatest possible extent of local autonomy.

The remainder of the paper is organized as follows. We start with an introduction to Active Component Database Systems as the main components of our architecture in section 2. Section 3 defines partial integrity constraints as a new type of constraints suitable for ACDBS, followed by a detailed explanation of global integrity checking based on partial integrity constraints in section 4. Section 5 discusses properties of our concepts, while related work is presented in section 6. Section 7 concludes and draws up future work.

2 Active Component Database Systems

We start with an introduction to Active Component Database Systems (ACDBS) as the main concepts of our approach. An ACDBS is an autonomous component database system or component system (CDBS or CS) of a federated database system as described in [2]. The active functionality of this kind of component systems, which we are going to describe in the following, can be used to ensure consistency and to enforce business rules in both, tightly coupled and loosely

coupled federations. Within the classical notion of federated databases, the component systems do only have passive functionality regarding the federation. Like repositories, they provide access to their data and respond to data requests initiated by the clients. Such passive component database systems, with respect to the federation, operate isolated and do not have any knowledge of other CDBSs within the federation to which their data is related.

Active database systems, which are not automatically ACDBSs when participating in a federation, assist applications by migrating reactive behavior from the application to the DBMS. They are able to observe special requirements of applications and react in a convenient way if necessary to preserve data consistency and integrity. The integration of active behavior in relational database systems is not particularly new and most commercial database systems support ECA rules, whereas the execution of triggers is mainly activated by operations on the structure of the database (e.g. insert or update a tuple) than by user-defined operations [3]. Unfortunately, the ability to check constraints in active databases, especially the scope of trigger conditions and actions, has until recently been limited to the isolated databases they were defined at. Subsequent developments integrated special purpose programming languages (e.g. PL/SQL [4]) into the database management system to overcome some limitations of the query language and to provide a more complex programming solution for critical applications. But again, the scope of these extensions was strictly limited to the system borders of the database system, so an interaction with its environment was impossible. Thus, the support of ECA rules and triggers is necessary, but not sufficient for the concept we propose here.

Latest developments, especially in commercial database systems, take the functionality of active databases beyond former limits. The significant improvement, on which this work is based on, is the ability of modern active database systems to execute programs written in a standalone programming language from within triggers, user defined functions, or stored procedures.

Definition 1. *The ability of a database system to execute programs or methods from within its DBMS to interact with software or hardware components beyond its system border shall be called* enhanced activity*. A database with enhanced activity is an* Enhanced Active Database System *(EADBS). The execution of a program or method in this context shall be called an* External Program Call *(EPC).*

The execution of external programs (EPs) from inside the DBMS offers new perspectives to data management and processing in an information sharing environment. Besides the maintenance of global integrity constraints as presented here, it can be used to improve communication with other external components like database wrappers [5]. In this paper, we use the database connectivity of the programming language to add the following functionalities to a component database system of a federation:

Query the state of a remote database: The main functionality which is elementary for our approach is the ability of an CDBS to query a remote data

source *directly* during the execution of a database trigger. After a connection has been established by the EP, we can perform any read operation on the remote schema items we are allowed to access. Depending on the query language we can formulate complex queries with group and aggregate functions (e.g. like in SQL). The query result of the remote database is used locally to evaluate conditions of ECA rules. We call this kind of query a *remote state query*.

Manipulating a remote database: After a connection is set up by the program, a CDBS is basically able to modify the data stock of the remote database *directly* during the execution of a database trigger. Assuming the appropriate permissions, any operation supported by the query language can be executed including data insertions, updates, and deletions. Depending on the query language, a CDBS is thus basically able to modify even the schema of a remote database using for example `ALTER TABLE` statements in SQL. In the following, a manipulation of remote data or schema items from within a database trigger shall be called an *injected transaction*, since its execution depends on a triggering transaction on a local relation.

The programming language used for EPs has to provide the functionality to open and close a connection to a remote data source and execute queries upon that data stock. Furthermore, we must be able to pass parameters to the EP and to access the corresponding program output from inside the trigger. This output can be used to evaluate trigger conditions or to determine subsequent trigger actions. Since the EPCs are embedded straight inside the DBMS of the local system, we are able to delay or abort transactions depending on the state of another data source or the result of an injected transaction. Just like common triggers that exclusively use local data to evaluate their trigger conditions, the DBMS autonomously schedules the execution of the trigger that encapsulates the EP. In particular, we do not force a component system to provide an atomic commitment protocol like 2PC. From the point of view of the remote database, a query of another DBMS via the database server is handled like a request of an ordinary application.

Within recent commercial database systems a commonly supported programming language which meets the requirements just mentioned is Java (e.g. Java Stored Procedures or Java UDFs [6,4]). It contains JDBC, a common database connectivity framework, to provide a standardized interface for a multitude of different data sources like relational databases or even flat files. During the execution of an EP based on Java, we use JDBC to connect to remote data sources from within triggers. After a connection has been established, we execute queries using SQL as a standardized query language. Our concept can be adapted to other relational database systems supporting different programming languages that fulfill the requirements listed above.

Definition 2. *An* Active Component Database System (ACDBS) *is an EADBS, which actively participates in maintaining global integrity constraints in a federation. It is able to directly communicate with other component systems,*

to which its data is semantically related, and implements constraint checks to maintain consistency among this interdependent data.

Constraint checks performed by ACDBSs are entirely implemented and executed on local CDBSs, but require access to remote data. Since these checks cannot be expressed by either local or global constraints, we introduce *partial integrity constraints* as a new type of integrity constraints for ACDBS.

3 Partial Integrity Constraints

In this section, we discuss partial integrity constraints as the basic concept of our approach. As already mentioned, we assume a federation of relational databases. Each ACDBS in this federation has to meet two requirements concerning the programming language for encoding EPs: (1) It must be able to connect to other component systems of the federation using the database connectivity of the programming language and (2) it must support a query language understood by the other component systems to execute at least read operations on the remote data stock. In practice, two widely used standard database connectivity interfaces are JDBC and ODBC, which support a multitude of relational databases. An established query language for relational databases certainly is SQL. This enhanced functionality is used to implement constraint checking algorithms for partial integrity constraints, which are defined next.

3.1 Definition of Partial Integrity Constraints

We start with the following definitions similar to [7]:

Definition 3. *A federation F of relational component systems is a set of n interconnected database systems $\{S_1, ..S_n\}$. The database systems do not necessarily have to be located on physically different nodes of the network. Each system $S_i \in F$ manages a local database D_i. A local schema $\mathcal{D}_i$ of a database D_i comprises the schemata $\mathcal{R}^i_1, ...\mathcal{R}^i_{n_i}$ of the relations $R^i_1, ..., R^i_{n_i}$ stored in the database. The global database schema $\mathcal{G}$ of F is the set of all relational schemata $\mathcal{R}^i_j$ in F.*

We assume that a real-world object, that is modeled in a component database of F, is globally identified by a set of key attributes, i.e. a real-world object will have the same key attribute values when stored in different CDBSs. Otherwise we assume mapping functions to match real-world objects in different sources.

Definition 4. *A* local integrity constraint $I_L^{D_i}$ *is a boolean function over a local database schema $\mathcal{D}_i$, i.e. $I_L^{D_i} : \mathcal{D}_i \rightarrow \{true, false\}$. A* global integrity constraint *I_G is a boolean function over the global schema $\mathcal{G}$, i.e. $I_G : \mathcal{G} \rightarrow \{true, false\}$. It cannot be expressed over a local database schema $\mathcal{D}_i \in \mathcal{G}$.* Constraint checks *for $I_L^{D_i}$ and I_G are algorithms for evaluating $I_L^{D_i}$ and I_G respectively.*

After a global constraint I_G has been defined over $\mathcal{G}$, we identify a non-empty set $\mathcal{C} \subseteq F$ of component databases $c \in \mathcal{C}$ whose local schemata $\mathcal{R}_c$ are affected by I_G, i.e. data stored in the relations R_c on the component databases is semantically related. Thus, from the point of view of each component database c, I_G affects a relation managed locally and at least one remote relation managed by another component system. For example, if a key constraint is defined on a global attribute that is derived from multiple sources, each of the sources has to ensure the global uniqueness of the key attribute, when a new tuple is inserted locally. This means that I_G consists of a set of *partial integrity constraints*, which we define as follows:

Definition 5. *A* partial integrity constraint I_{R_c} *on an ACDBS* $c \in \mathcal{C}$ *is a boolean function, which is expressed over the local schema* $\mathcal{R}_c$ *and related schemata* $\mathcal{R}_{k_u}$ *for* $k_u \in \mathcal{C} \setminus \{c\}$*, i.e.* $I_{R_c} : \mathcal{R}_c \times \mathcal{R}_{k_1} \times \cdots \times \mathcal{R}_{k_v} \rightarrow \{true, false\}$ *for* $v \leq |\mathcal{C}| - 1$*. A constraint check for* I_{R_c} *is an algorithm for evaluating* I_{R_c}*, which is entirely implemented on* c *using external program calls to access the remote schemata* $\mathcal{R}_{k_u}$*.*

A partial integrity constraint consists of a local constraint check and one or more remote constraint checks on interrelated remote data, depending on the type of global constraint and the number of affected databases. It is used to express a global constraint from the local view of a single component database. An ACDBS, which implements a partial integrity constraint, has to ensure consistency of its local data depending on related data stored in other component databases. This means that it is responsible for checking a specific part of the corresponding global integrity constraint concerning local write operations on the interrelated data. Thus, a global integrity constraint is assured, iff all affected component systems enforce their corresponding partial integrity constraints, expressed as

$$I_G : \bigwedge_{c \in \mathcal{C}} I_{R_c}$$

The global constraint I_G consists of a conjunction of partial integrity constraints I_{R_c}, which are formulated as

$$I_{R_c} : local_{R_c} \wedge \bigwedge_{j \in \mathcal{C}'} remote_{R_c,R_j}$$

Depending on the type of global integrity constraint, each affected ACDBS c has to check an optional local condition $local_{R_c}$ and one or more remote conditions. $remote_{R_c,R_j}$ defines a pairwise dependency between the affected local relation R_c and one interrelated relation R_j on component j. Remote conditions do not necessarily have to be defined for each pair of local and remote databases in $\mathcal{C}$. Thus, $\mathcal{C}' \subseteq \mathcal{C} \setminus \{c\}$ denotes a subset of ACDBSs, which are required to check for a specific integrity constraint. A constraint check for I_{R_c} implements tests for the local condition $local_{R_c}$ and each remote condition $remote_{R_c,R_j}$. For aggregate constraints, a test for a local or remote condition results in the successful computation of a local or remote aggregate. Detailed examples for partial constraint checks are provided in section 4.

3.2 Specification of Partial Integrity Constraints as ECA Rules

Since our concept of global integrity maintenance is based on ACDBS with enhanced activity, we use database rules following the event-condition-action (ECA) paradigm to specify a partial integrity constraint I_{R_c} on a component database c as follows:

define rule *PartialIntegrityRule* I_{R_c}
on *event which modifies* R_c
if *a test for* $local_{R_c}$ *yields false or*
a test for $remote_{R_c,R_j}$ *yields false*
do *local and/or remote action(s) to ensure or*
restore a consistent global state

Such integrity rules can precisely define both: events that potentially violate the integrity of local and remote data, and corresponding reactions on these events to ensure or restore consistency in the entire system. The relevant events concerning data consistency are modifications of the data stock, i.e. insertions, updates, and deletions. According to the definition of partial constraints, each rule condition and rule action of a partial integrity rule can consist of a local and one or more remote checks. The local check $local_{R_c}$ exclusively uses and accesses local data, while the remote checks $remote_{R_c,R_j}$ exclusively process data stored on remote systems. The remote checks of a partial integrity rule are implemented using remote state queries (or injected transactions) provided by the ACDBS. Thus, we have the following options to call an external program during an integrity check:

During the evaluation of a trigger condition: An external program call during the evaluation of a trigger condition allows a DBMS to determine subsequent actions depending on the result of a remote state query or the result of an injected transaction. Thus, a locally executed constraint check can be conditioned by the state and behavior of a remote data source. In our concept, most of the constraint checks are implemented using remote state queries from within trigger conditions. The part of the trigger condition, which evaluates a condition using remote data shall be called *remote condition*.

During the execution of (a) trigger action(s): Besides the remote condition, an external program can be executed as a trigger action. A local transaction can thus trigger an injected transaction to manipulate a remote data source. This can be used to execute consistency restoration actions or to implement special constraints like cascading referential integrity. This part of the trigger action, which manipulates remote data using injected transaction shall be called *remote action*.

The specific combination of the time the corresponding EP is executed (i.e. during remote condition or remote action) and the time a partial integrity rule is evaluated (i.e. **before** or **after** a local transaction is committed) is significantly

affecting the behavior of the entire system. Please note, that we are certainly not limited to exclusively one of these combinations. During the evaluation of a partial integrity rule, external programs can be called from both, the remote condition and the remote action, before or after a local transaction is committed.

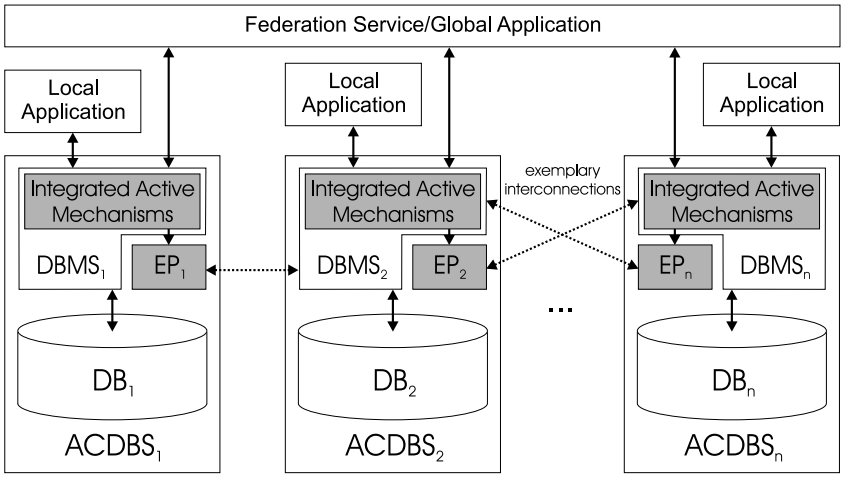

Fig. 1. Basic Architecture

Putting it all together, we present the architecture for global integrity checking in heterogeneous information systems using active component systems depicted in Fig. 1. The partial integrity checks are defined and implemented directly in the ACDBSs, building up an application independent communication layer to jointly ensure global consistency of interdependent data. The ACDBSs call EPs to check remote conditions or to execute remote actions of locally defined partial integrity constraints. Each transaction is checked according to local and partial integrity constraints, no matter if submitted by local or global applications. The maintenance of global integrity is thus migrated from a global application or federation layer to the underlaying active component systems.

3.3 System Interaction

We now give a schematic description of the interaction process between two ACDBSs during the execution of a partial integrity check. Consider two relations R and S on active component systems $ACDBS_1$ and $ACDBS_2$, which store interdependent data. To enforce a global constraint, we have to define and implement partial constraint checks on both component systems. Therefore we create the following objects on each ACDBS (see Fig. 2):

– An **external program (EP)** (here a Java method) to execute queries upon the remote data stock,

– a **user defined function (UDF)**, which is mapped to the external Java method, and
– a **trigger** which executes the UDF when relevant write operations occur on the relation.

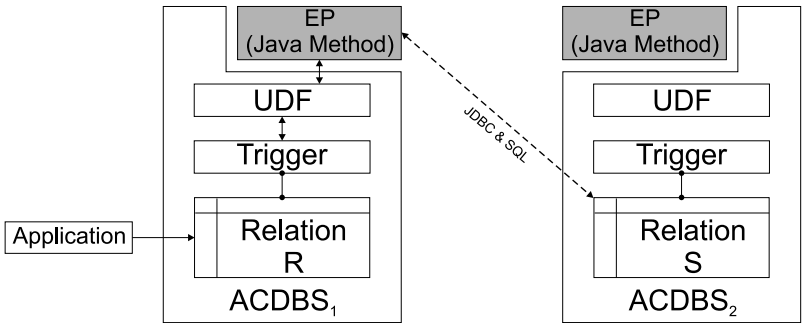

Fig. 2. Interaction between two Active Component Systems

We assume a global key constraint to be enforced on R and S, i.e. whenever a tuple is inserted or modified in R or S, we have to check the global uniqueness of the key attributes of the newly inserted or modified tuple in both relations. Our description focuses on data modifications in R, since modifications in S would be processed analogously. When an application inserts or updates data items ΔR in R, the corresponding trigger is executed by the database system *before* the transaction is completed. The trigger has access to ΔR via temporary tables provided by the DBMS. For each new tuple r in ΔR, the trigger first performs a check on the local data and afterwards, if necessary, on the remote data. If the local test fails, the key constraint is already violated and the remote test is omitted. If the local test succeeds, the trigger calls a UDF to check for conflicts in the remote data. The UDF is mapped to a Java function and receives r as a parameter from the trigger.

The Java function now bridges the gap between the two component systems. Using JDBC it connects to the remote database and executes an SQL query to check for the existence of r in S. The function returns $true$ or $false$ depending on the query result. The trigger receives this result and is now able to determine subsequent actions. Please keep in mind that, for the scope of this paper, we execute local and remote checks during the evaluation of the trigger condition before the transaction is completed. Thus, the transaction is blocked as long as the trigger is executed. If a corresponding tuple already exists in S, then we reject the data modifications on R. An SQL error is raised to signal the global key constraint violation.

4 Checking Global Integrity Constraints

We now concretize our concept of global integrity maintenance explaining how partial integrity constraints are expressed and implemented for different con-

straint types. Therefore we use a simplified scenario with two relational data sources. A generalization to more than two sites is discussed in section 5.

Consider a company with two research departments A and B, which both manage their own autonomous relational database DB_A and DB_B. The company wants to integrate these standalone sources into an information system and define integrity constraints to ensure global data consistency within the entire company. We assume that the sources are relational databases with enhanced activity, which host the relations shown in Table 1. Each department stores information about researchers in relation *res* (researcher number R, name N, salary S) and their corresponding projects in relation *proj* (project number P, title T, budget B). Researchers are related to projects using the *prores* relation (m:n).

Department	Database	Relations
A	DB_A	$resA(R_A, N_A, S_A)$ $projA(P_A, T_A, B_A)$ $proresA(R_A, P_A)$
B	DB_B	$resB(R_B, N_B, S_B)$ $projB(P_B, T_B, B_B)$ $proresB(R_B, P_B)$

Table 1. Example relations in the research departments A and B

According to the classification presented in [8,9], we consider four commonly used classes of global integrity constraints which can be defined on the global schema: attribute constraints, key constraints, referential integrity constraints, and aggregate constraints. In the following, we provide an example for each non-trivial class of global integrity constraints, followed by a rule definition for corresponding partial integrity constraint on the affected ACDBSs.

4.1 Attribute Constraints

The company could define a constraint saying that the budget of each project may not exceed a certain value. Since this global attribute constraint is expressed over a single attribute, it can be translated into a local attribute constraint and thus be enforced by local integrity mechanisms on DB_A and DB_B, e.g. by an additional **check** clause in both project relations. The global constraint can be enforced by a local constraint check on each ACDBS, so no EPCs are required.

4.2 Key Constraints

The company may want to ensure that each project is globally identified by a unique identifier, i.e. the values stored in P_A and P_B are globally unique. Thus, each time a project is added in one of the research departments, we have to ensure that the new project number does not already exist locally and in the project database of the other research department.

Since partial constraint checks are entirely implemented on a participating ACDBS using EPCs to access remote data, we decompose Key_G into a set of partial integrity constraints Key_{projA} and Key_{projB} for DB_A and DB_B respectively:

$$Key_G : Key_{projA} \wedge Key_{projB}$$

The partial constraints are in turn formulated as

$$Key_{projA} : local_{projA} \wedge remote_{projA,projB}$$
$$Key_{projB} : local_{projB} \wedge remote_{projA,projB}$$

A partial constraint consist of a local condition and a remote condition, which are defined for Key_{projA} as follows (Key_{projB} is defined analogously):

$$local_{projA} : \forall P, T, B, P', T', B' :$$
$$[projA(P, T, B) \wedge projA(P', T', B') \Rightarrow \neg(P = P')]$$
$$remote_{projA,projB} : \forall P, T, B, P', T', B' :$$
$$[projA(P, T, B) \wedge projB(P', T', B') \Rightarrow \neg(P = P')]$$

Suppose the tuple $projA(p, t, b)$ is inserted. According to Key_{projA} we have to check the existence of the key locally and in $projB$, stored on DB_B, with the following tests for $local_{projA}$ and $remote_{projA,projB}$:

$$localtest_{Key_{projA}} : \nexists P, T, B : [projA(P, T, B) \wedge (P = p)]$$
$$remotetest_{Key_{projA}} : \nexists P, T, B : [projB(P, T, B) \wedge (P = p)]$$

Both tests are evaluated by performing queries on the relevant relations for a tuple that has p as its project number. Therefore, we need two boolean functions: $checklocalkey : schema(projA) \rightarrow \{true, false\}$ for $localtest_{Key_{projA}}$ and $checkremotekey : schema(projB) \rightarrow \{true, false\}$ for $remotetest_{Key_{projA}}$. *checklocalkey* should always be evaluated first to avoid cost-intensive remote data access where possible. Please note that although the uniqueness of key attributes may already be enforced by an additional local key constraint, we need the local check in the partial constraint since in general it is not possible to access the result from a local constraint check from within an active component like a trigger.

The *checkremotekey* function is implemented using a remote state query, which queries database DB_B to find tuples with the values to be inserted. If the query result is not empty or the remote source is not reachable by the external program, then the function is evaluated to $false$, i.e. the corresponding transaction in database DB_A is rejected. The condition is evaluated *before* the triggering operation is committed at DB_A. The corresponding partial rule for Key_{projA} is expressed as:

define rule *PartialKeyConstraint*
on *creation of a new object in projA*
if *checklocalkey yields false* **or** *checkremotekey yields false*
do *reject transaction*

A partial constraint is herewith realized on an ACDBS with an implementation of the ECA rule using two functions *checklocalkey* and *checkremotekey* with one remote state query. Having implemented both partial constraints Key_{projA} and Key_{projB} on both systems, we are able to verify Key_G each time a modifying transaction is committed locally on DB_A and DB_B.

4.3 Referential Integrity Constraints

A widely spread constraint is the definition of referential integrity on relations to specify existence dependencies between two database objects. Referring to our scenario, the company could allow researchers of department A to cooperate on shared projects of department B. Thus, we have to ensure that a researcher in DB_A is related to an existing project in DB_B and vice versa. As already mentioned, researchers are related to projects via the *prores* relation referencing the relevant primary keys of the local project and researcher relations. Now, to reflect the global referential integrity constraint in our exemplary relational model, we allow R_B in $proresB$ to reference both, local researchers using R_B in res_B and cooperating researchers in res_A using R_A. In the scope of this paper we only consider referential integrity without cascading, although our concept of ACDBSs basically supports cascading. An outlook on cascading referential integrity can be found later in this section.

Referential Integrity Without Cascading In the following, we focus on the referential integrity concerning the researcher number R_B in $proresB$. Referential integrity for P_B in $proresB$ is handled analogously. Similar to global key constraints, a global referential constraint is first decomposed into a set of partial constraints:

$$RefInt_G : RefInt_{resA} \wedge RefInt_{proresB}$$

The existence dependency between the local parent relation $resA$ and the local dependent relation $proresA$ can be expressed as follows:

$$\begin{aligned} local_{proresA} : \forall R, P \exists R', N, S : \\ [proresA(R, P) \wedge (R = R') \Rightarrow resA(R', N, S)] \end{aligned}$$

Furthermore we formulate a remote constraint $remote_{proresB,resA}$ as

$$\begin{aligned} remote_{proresB,resA} : \forall R, P \exists R', N, S : \\ [proresB(R, P) \wedge (R = R') \Rightarrow resA(R', N, S) \vee resB(R', N, S)] \end{aligned}$$

Using these definitions, we express the partial constraints for $RefInt_G$ as

$$\begin{aligned} RefInt_{resA} &: local_{proresA} \wedge remote_{proresB,resA} \\ RefInt_{proresB} &: remote_{proresB,resA} \end{aligned}$$

A project can only be inserted into $proresB$, if a corresponding researcher exists in either $resB$ (locally) or $resA$ (remote). Contrary, a researcher in the parent relations $resA$ and $resB$ may not be deleted, as long as depending projects exist in the dependent relation $proresB$. Thus, we have to distinguish between constraint checks for insertions and deletions on the dependent and parent relations respectively.

Insertion check: Suppose the tuple $proresB(r, p)$ is inserted, whereas p refers to an existing project in $projB$. According to $RefInt_{proresB}$ we have to check the existence of r locally and remote using the following tests for $remote_{proresB,resA}$:

$$localtest_{RefInt_{proresB}} : \exists R, N, S : [resB(R, N, S) \wedge (R = r)]$$
$$remotetest_{RefInt_{proresB}} : \exists R, N, S : [resA(R, N, S) \wedge (R = r)]$$

If one of the tests yields *true*, then there exists a corresponding entry in either the local or remote parent relation and the tuple $proresB(r, p)$ can be inserted. Otherwise the insertion has to be rejected. For the implementation of these tests, we use the functions *checklocalkey* and *checkremotekey* as introduced in section 4.2. The remote test is evaluated using a remote state query on DB_A. A corresponding ECA rule for this partial constraint can be expressed as follows:

define rule *PartialReferentialConstraint*
on *creation of a new object in proresB*
if *checklocalkey yields false* **and** *checkremotekey yields false*
do *reject transaction*

Deletion check: Suppose the tuple resA(r,n,s) shall be deleted from DB_A. According to $RefInt_{resA}$ we have to ensure that there are no depending objects in the *prores* relations on DB_A and DB_B before we delete this item. Thus, we formulate the following tests for $local_{proresA}$ and $remote_{resA,proresB}$:

$$localtest_{RefInt_{resA}} : \nexists R, P : [proresA(R, P) \wedge (R = r)]$$
$$remotetest_{RefInt_{resA}} : \nexists R, P : [proresB(R, P) \wedge (R = r)]$$

The deletion check succeeds, i.e. $resA(r, n, s)$ can be deleted, if there are no dependent objects in $proresA$ and $proresB$. This partial constraint is represented by the following ECA rule:

define rule *PartialReferentialConstraint*
on *deletion of an object in resA*
if *checklocalkey yields true* **or** *checkremotekey yields true*
do *reject transaction*

Of course, we have to ensure that an entry in the parent table exists either in $resA$ or $resB$. This is realized using a key constraint on the researcher id as presented in section 4.2.

Cascading Referential Integrity Constraints With the extended functionality of Active Component Systems, we are basically able to realize cascading referential integrity on updates or deletions of tuples. Injected transactions can be executed during the evaluation of a partial integrity constraint to modify remote data stocks including even deletions, before or after the modifying operation is committed locally. If a tuple is deleted in the parent relation, we execute an injected transaction to delete all corresponding tuples in the dependent relation. Analogously, if a key value is updated in the parent relation, we cascade this update to the dependent relation by modifying the relevant entries in the remote database via injected transactions.

The corresponding partial integrity constraint for the parent relation is expressed similar to the partial rule without cascading presented in 4.3. We extend the rule to delete dependent objects from within the rule condition or action depending on the intended system behavior. Thus, we are able to delete entries in the dependent relation from within a *remote condition* or a *remote action*, *before* or *after* the local entry is deleted. Of course, since we modify data on more than one autonomous database system, we face the problem of atomic commitment in a multidatabase environment [10]. A distributed update may lead the federation into a (temporary) inconsistent state in case of a failure. We therefore need a recovery mechanism based on the concept proposed to detect inconsistencies and restore global integrity automatically after a transaction has violated global constraints. Following the line of argumentation in [11] we consider weakened notions of consistency, using guarantees for the level of consistency a system can provide. The integration of weakened consistency into our architecture is part of future work.

4.4 Aggregated Constraints

As a representative for this type of constraint let us assume that the company has a budget limit for all research projects. Thus, it must be checked whenever a project is created or updated in DB_A and DB_B that the sum of all project budgets B_A and B_B does not exceed a certain value ε. We restrict our further considerations on the standard aggregate functions *min*, *max*, *sum*, and *count*. The average function *avg* must be calculated during a partial constraint check using *sum* and *count*. Furthermore, we assume that $agg(T, w)$ is an aggregate function that calculates the aggregate of an attribute w of a relation T. The function $totalagg(agg(R_{m_1}, w_{m_1}), ..., agg(R_{m_s}, w_{m_s}))$ computes the overall aggregate of partial aggregates for $m_u \in \mathcal{C}$ and $s = |\mathcal{C}|$. Please note that *count* is a semi additive aggregate function and the overall aggregate must be calculated as the sum of partial *count* aggregates.

These preliminaries provided, we can now formulate a global aggregated constraint for our example as

$$Sum_G : Sum_{projA} \wedge Sum_{projB}$$

with the partial constraints defined as

$$Sum_{projA} : totalsum(localsum_{projA}, remotesum_{projB}) \ \leq \ \varepsilon$$

$$Sum_{projB} : totalsum(localsum_{projB}, remotesum_{projA}) \leq \varepsilon$$

Both databases have to check the total sum whenever an insertion or update occurs on B_A or B_B. Therefore, DB_A calculates its corresponding local and remote aggregate as

$$localsum_{projA} = sum(projA, B_A) \text{ and } remotesum_{projB} = sum(projB, B_B)$$

using two functions $agglocal : schema(projA) \rightarrow \mathbb{R}$ and $aggremote : schema(projB) \rightarrow \mathbb{R}$. The calculation of the remote aggregate is realized using a remote state query on DB_B. The aggregates for DB_B are calculated analogously.

Now suppose the tuple $projA(p, t, b)$ is inserted. According to Sum_{projA} we first compute $localsum_{projA}$ including the new value b and $remotesum_{projB}$ on DB_B. After we receive the result from the remote aggregation, we calculate $totalsum$ and compare the overall aggregate to ε. If the comparison yields $false$ then the insertion of $projA(p, t, b)$ is rejected. A corresponding ECA rule for this partial constraint can be expressed as:

define rule *PartialAggregatedConstraint*
on *update of* B_A *in* $proj_A$ **or** *insertion of a new object in* $proj_A$
if $totalsum(localsum_{projA}, remotesum_{projB}) > \varepsilon$
do *reject transaction*

5 Discussion

The checking mechanism presented in this paper is basically an implementation of the Local Test Transaction Protocol (LTT) presented by Grefen and Widom in [7]. The LTT exploits transaction capabilities provided by the local database system to perform a notification and wait for acknowledgment within a single transaction. Using the LTT we try to avoid remote checks by evaluating local tests first. If a local test has already failed, then we do not have to evaluate the cost intensive remote check using remote state queries. The implementation of a transaction-based protocol like LTT has to evaluate a constraint check *before* the triggering operation is committed. The external program is executed as part of the remote condition of a partial integrity rule. Our implementation certainly adopts all advantages and drawbacks of the applied LTT protocol. Thus, the implementation proposed is safe and accurate, which means that it detects all constraint violations and that, whenever an alarm is raised, there is a state in which the constraint is violated. On the other hand, since the relation is locked until the external program returns a result, the local ACDBS looses autonomy and the risk of deadlocks is relatively high, if relations in DB_R and DB_S are updated concurrently.

Due to the flexibility of our architecture, we are basically able to implement the entire set of protocols described by Grefen et al. Thus, to overcome the drawbacks of the LTT, we can modify the partial integrity constraints to implement

the Materialized Delta Set Protocol (MDS), which increases autonomy and reduces the risk of deadlocks. Therefore, we maintain an additional relation ΔR, which stores an accumulated set of updates of the original relation R. The constraint checking mechanism is then evaluated using ΔR instead, so the original relation is not locked during the check. This enables at least concurrent read access to R while updates must still be delayed until the lock is released. In our architecture ΔR is maintained using the active capabilities of the DBMS. We define a local rule on R to copy all updated items to ΔR. The partial integrity rule including the remote checks as presented above is then expressed over the Materialized Delta Set ΔR.

A generalization of the constraint checking mechanism to more than two sites is tightly corresponding to the implemented integrity checking protocols. As already described in [7], most of the constraints that involve more than one site can be decomposed into a couple of constraints, which are expressed over exactly two databases. If there is the need for multi-site constraints, the authors propose to use non-transaction-based protocols like Direct Remote Query (DRQ) or Timestamped Remote Query Protocol (TRQ), which can both be implemented using our architecture. To avoid locking of the updated relation we adjust the partial integrity rule to be evaluated *after* the modifying operation is committed.

The protocols could be optimized according to the Demarcation Protocol presented in [12]. This protocol is particularly suitable for arithmetic constraints like aggregated constraints, but can also be used for key or referential integrity constraints. The Demarcation Protocol can be seen as an extension to the LTT and thus be implemented using our architecture.

6 Related Work

In the last years, research on integrity constraints in heterogeneous environments mainly considered the simplification, evolution, or reformulation of constraints rather then mechanisms or protocols for integrity checking. A closely related concept in terms of the rule structure and constraint types is presented in [9]. The authors use private and public global constraints to define dependencies between data in different databases, similar to the partial constraints presented in this paper. One of the main differences is the use of a layered approach to support the active functionality required for event detection and rule processing. A reactive middleware based on CORBA encapsulates active and passive sources and processes rules using an external remote rule processing mechanism. Furthermore every component is assumed to have an Update Processor to execute local update requests, but the local relation cannot be locked during the evaluation of remote conditions.

The metadatabase approach [13] uses a rule-oriented programming environment to implement knowledge of information interactions among several subsystems. Each subsystem is encapsulated by a software shell, which is responsible for monitoring significant events, executing corresponding rules, and interacting

with other shells. Although conditions can be evaluated in a distributed way, the rule processing itself is still centralized.

A distributed rule mechanism for multidatabases is presented in [14] as part of the Hyperion project. A distributed ECA rule language is introduced, which is mainly used to replicate relevant data among data peers in a push-based fashion. Rules are processed by a rule management system that resides in the P2P layer on top of a peer database. The X^2TS prototype [15] integrates a notification and transaction service into CORBA using a flexible event-action model. The architecture presented resembles a publish/subscribe system, whereas publishing of events is non-blocking. Another middleware approach for distributed events in a heterogeneous environment is presented in [16]. CORBA-based, distributed, and heterogeneous systems are enhanced by Active DBMS-style active functionality. The architecture uses wrappers with event monitors to detect data modifications in the data source.

A common characteristic of the architectures just mentioned is the use of a layered approach with event monitoring to somehow notify a mediating component (e.g. a constraint manager, rule processor, or middleware component) about events occurring in the local database. If the source is not monitored, the notification mechanism is generally based on active capabilities of the underlaying database management system, but there is so far no detailed description of this interaction published.

The most distinctive characteristic of our concept is the direct usage of existing active capabilities of modern database management systems without the need for wrappers or monitoring components. Since a remote condition is evaluated during the execution of a trigger, it is irrelevant if the triggering transaction was a global or local update. We benefit from the active functionality of the DBMS in terms of transaction scheduling, locking, and atomicity, resulting in a synchronous integrity checking mechanism. Especially the ability to rollback updates depending on a remote state query makes corrective or compensative actions basically superfluous.

7 Conclusion and Future Work

We have presented an architecture for global integrity maintenance in a federated relational database using component database systems with enhanced active functionality. We introduced Active Component Database Systems which are able to communicate with other component databases to which their data is semantically related to. They are no longer just passive data providers but actively participating in global integrity maintenance. Global integrity constraints are composed of sets of partial integrity constraints for each component system that is affected by the constraint. The partial constraints are evaluated using local and remote checks which are implemented entirely on a local site. We have described the requirements and basic functionality of our architecture and provided examples for partial constraints for commonly used classes of global constraints.

In the next steps we address the problem of distributed updates using injected transactions as needed for cascading referential integrity or data replication. Furthermore, we plan to deal with the detection, resolution, and prevention of deadlocks and examine the system behavior depending on the time a rule is evaluated and an external program is called. The architecture shall be elaborated in a true loosely coupled environment to evaluate execution costs and scalability.

References

1. Heimbigner, D., McLeod, D.: A Federated Architecture for Information Management. ACM Transactions on Information Systems (TOIS) **3** (1985) 253–278
2. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys **22** (1990) 183–236
3. Paton, N.W., Díaz, O.: Active database systems. ACM Computing Surveys (CSUR) **31** (1999) 63–103
4. Loney, K., Koch, G.: Oracle8i: The Complete Reference. Osborne/McGraw-Hill (2000)
5. Popfinger, C., Conrad, S.: Tightly-coupled Wrappers with Event Detection Subsystem for Heterogeneous Information Systems. In: DEXA Workshop Proceedings, IEEE Computer Society Press (2005) to appear.
6. Chamberlin, D.: A Complete Guide to DB2 Universal Database. Morgan Kaufmann (1998)
7. Grefen, P.W.P.J., Widom, J.: Integrity Constraint Checking in Federated Databases. In: Conference on Cooperative Information Systems, IEEE Computer Society Press (1996) 38–47
8. Türker, C., Conrad, S.: Towards Maintaining Integrity of Federated Databases. In: Data Management Systems, Proc. of the 3rd Int. Workshop on Information Technology, IEEE Computer Society Press (1997) 93–100
9. Gomez, L.G.: An Active Approach to Constraint Maintenance In A Multidatabase Environment. PhD thesis, Arizona State University (2002)
10. Mullen, J.G., Elmagarmid, A.K., Kim, W., Sharif-Askary, J.: On the Impossibility of Atomic Commitment in Multidatabase Systems. In: Proc. of the 2nd Int. Conf. on System Integration, IEEE Computer Society Press (1992) 625–634
11. Chawathe, S., Garcia-Molina, H., Widom, J.: A Toolkit For Constraint Management In Heterogeneous Information Systems. In: Proc. of the Int. Conf. on Data Engineering. (1996) 56–65
12. Barbará-Millá, D., Garcia-Molina, H.: The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems. The VLDB Journal **3** (1994) 325–353
13. Hsu, C., Rattner, L.: Metadatabase Solutions for Enterprise Information Integration Problems. DATA BASE **24** (1993) 23–35
14. Kantere, V., Mylopoulos, J., Kiringa, I.: A distributed rule mechanism for multidatabase systems. In: CoopIS/DOA/ODBASE. (2003) 56–73
15. Liebig, C., Malva, M., Buchmann, A.P.: Integrating Notifications and Transactions: Concepts and X^2TS Prototype. In: EDO. (2000) 194–214
16. Koschel, A., , Kramer, R.: Configurable Event Triggered Services for CORBA-based Systems. In: EDOC. (1998) 306–318