Distributing Semantic Constraints Between Heterogeneous Databases

Stefan Grufman*†, Fredrik Samson*†, Suzanne M. Embury†,
Peter M.D. Gray† and Tore Risch*

*Department of Computer and Information Science,
Linköping University, S-581 83 Linköping, Sweden
E-Mail: {d92stegr|d92fresa}@und.ida.liu.se
torri@ida.liu.se

†Department of Computing Science, King's College,
University of Aberdeen, Aberdeen, Scotland, U.K.
E-Mail: {sme|pgray}@csd.abdn.ac.uk

To be presented the 13th International Conference on Data Engineering, ICDE'97, Birmingham, U.K., 7th - 11th April, 1997

Abstract

In recent years, research about distributing databases over networks has become increasingly important. Here we concentrate on the issues of interoperability of heterogeneous DBMSes, and enforcing integrity across a multi-database made in this fashion. This has been done through a cooperative project between Aberdeen and Linköping universities, with database modules distributed between the sites. In the process we have shown the advantage of using DBMSes based on variants of the Functional Data Model, which has made it remarkably straightforward to interoperate queries and schema definitions. Further, we have used the constraint transformation facilities of P/FDM to compile global constraints into Active Rules installed locally on one or more AMOS servers. We present the theory behind this and the conditions for it to improve performance.

1 Introduction

Work on distributed and interoperable databases is widely published [2, 20, 25] but, to our knowledge, there is no work on making two functional databases interoperable. Since the functional model is not too common it is worth rehearsing the histories of the different systems.

The functional data model database system at Aberdeen (P/FDM) [15] started life as a vehicle for research in transformation and generation of complex queries, because the functional language is much closer to a mathematical language (λ-calculus or predicate calculus) than SQL. It also evolved from Shipman's [24] functional data model into more of an object model, in the process becoming more like that used in O_2 or ODMG [3]. This was needed in order to include in the shared database some geometric computational methods for objects representing protein structures and their components, as this was its main application area. Recent research has concentrated on using FDM to express quantified multi-variable semantic constraints in a form that can easily be transformed for efficient execution on different storage modules [8, 9].

Meanwhile the AMOS system at Linköping evolved out of experience with the early IRIS data model [13], which was itself strongly influenced by Shipman's FDM. However, AMOS (Active Mediators Object System) [11] was designed as a main memory database system that could easily be cloned into multiple communicating AMOS processes running and sharing data over a distributed network. It also had a powerful active rule capability [23, 27] which we have used. AMOS furthermore has capabilities to integrate relational databases into its multi-database environment [12].

In much of the work, both AMOS and P/FDM had been making use of a powerful way of computing with functions applied to the stored data and objects, without any kind of "impedance mismatch". These computations are also much easier to transform and to store than those expressed procedurally in C++, as shown in work by both groups [17, 15, 9, 14, 27, 12]; this also applies to constraint transformation (see later). Interestingly, much of the code of AMOS is in Lisp, while P/FDM is in Prolog, mainly for its powers of symbolic computations for query transformation. However, it should be noted that both these languages use a garbage-collected store (unlike C++) and that they make it easy to generate data structures which can be treated as code.

It is important to realise that the functional data model had its roots in the first major project for interoperability between inhomogeneous distributed databases MULTIBASE [19]. Thus when it came to generating schemas and queries in AMOS to give effect to a global schema with inter-database constraints expressed in FDM, it was all so much easier because FDM was originally designed as a general model from which to generate DDL and DML statements in other languages. This also makes our results relevant to Object-Relational databases (for example OpenODB which evolved out of IRIS) [28]. This is because the functional data model plays the role of a high level object model and it is capable of being implemented on a variety of storage schemas, and thus one can use extensible relational data storage to make a system very similar to an O-R one, but with a very general schema language that makes it easy to interoperate. This has been noted in AMOS papers [11, 12].

Let us summarise features of the two systems which played a crucial role in the integration:

- Multidatabase features in AMOS: Makes it possible to have several autonomous AMOS servers running and communicating in an AMOS network.
- Active Mediator Architecture in AMOS: Allows use of active rules.
- Modules in P/FDM: Allows partitioning of a database into separate modules of the same or different module type.
- Integrity Constraints in P/FDM: P/FDM provides a constraint checking system for both structural and semantic constraints.

Our integration idea was to introduce AMOS as a new module type into P/FDM and then take as much advantage of the AMOS system as possible to improve the performance and to get the multidatabase functionality in the system.

With this integrated system, it is now possible to create databases partitioned across the two different database systems, as well as across several AMOSes on different machines anywhere on the net. Thus the new system has taken advantage of the best parts in each original system. Figure 1 shows an example of the capability of the system, having a database spread out in one hash module and two AMOS modules, one of them sited in Linköping. Note that, in this example, the embedded AMOS, linked to the P/FDM, is used as client/server interface, communicating with the other AMOSes through its inter-database interface (I/D). However, the local AMOS could be used as a storage module too. Since P/FDM maintains the global schema, we

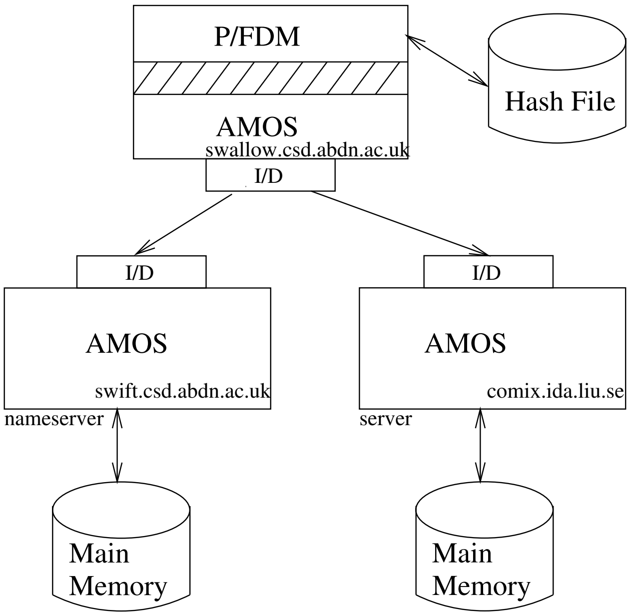

Figure 1. Example of multi-database session

were able to make use of the powerful semantic constraint language facilities in the language [1] to reference objects in different modules. This applied across the two modules, or even between two separate AMOS classes in separate AMOS modules. However, it was not very efficient, since where it could not make local checks it had to fall back on tuple-at-a-time access to the AMOS modules from P/FDM. The challenge to us, and the main results of this paper, concerns the use of active rules in the AMOS modules which are generated and planted from a Prolog program run from P/FDM. This produces a result which is theoretically equivalent, but almost always much faster than running integrity checks from P/FDM. However there are cases where it is not worth planting such rules, and this is decided by a simple heuristic. We believe that this kind of architecture is both theoretically and practically attractive, and that there are lessons in this for Object-Relational implementors.

In the next section we describe the main points of the integration of P/FDM with AMOS. Then we describe the theory behind our distribution of constraints over multiple databases, followed by a discussion of our approach to its implementation. Finally we discuss related work, summarise and discuss further research possibilities.

2 Integrating P/FDM and AMOS

When making two different systems talk to each other there are basically two different ways of communication, loose or tight coupling. Whether to look at the systems as loosely or tightly coupled depends on the level of control over the communication between the systems. It also depends on the degree of persistence of the communication link. A loose coupling is when two systems make a temporary connection, for example via a port or a pipe. Tight coupling is when the systems are more closely connected, for example running in the same process and sharing data structures.

Since Prolog is able to dynamically load foreign C code, and AMOS is provided as a C library, a tight coupling was straight-forward. The C interface functions of AMOS are declared as foreign functions in a Prolog file. When consulting this Prolog file the AMOS process is linked into the running P/FDM process. Then the tight connection is established, which gives full access to the AMOS multidatabase features used for communication with other AMOSes.

In this project, P/FDM was chosen as the 'top-level' system for several reasons.

- P/FDM had been tested with different types of storage "modules", such as hash files and Sybase storage and both in combination. Thus AMOS could easily be integrated as another storage module.
- AMOS had well working external interfaces with built in client/server and inter-database features.
- While P/FDM could send messages to AMOS servers, through its external interfaces, AMOS could not send messages to P/FDM except at P/FDM's request.

Thus there is a certain asymmetry in the combined system in that P/FDM has all the metadata and can see the global schema, whilst AMOS modules see their local schemas. Distributed constraints in such an object-oriented multidatabase setting sometimes require OIDs to be passed between the systems. This means that some of these AMOS classes will include functions apparently returning strings, which actually represent object identifiers of objects stored in P/FDM classes. In the other direction P/FDM classes may contain identifiers for AMOS objects, but this is coped with because these objects are in a separate storage module and P/FDM can deal with objects in such a module having a different representation.

There were some slight differences in the data models, mainly in the use of external identifiers (keys) for objects by P/FDM, which had to be represented by creating indexed collections in AMOS. Also there was a difference in the way subtype instances were connected to the data for their supertypes. This is explained later and was easily reconciled as noted earlier.

Initially the interface was implemented using AMOS' embedded query interface that dynamically evaluates queries. However, a substantial performance improvement was made by instead using AMOS' *fast path* interface where database functions are precompiled, optimised, and stored in the database. For example, the getentity/2 primitive of P/FDM is used to retrieve all instances of an entity. It is defined in AMOS as a derived function for each entity class. These AMOS functions are precompiled and called from P/FDM through the fast-path interface.

Another use of this system is the possibility of having several AMOSes running with their own stored data. A P/FDM multidatabase layer can be added to connect these AMOSes, and also to define relationships between the data in the existing databases. All that is needed for adding this layer is to import into P/FDM the AMOS database schemas and to define in the P/FDM schema the extra relationships.

3 A Formal Description of Constraint Distribution

The problem of distributing a constraint check over a number of databases can be stated generally as the problem of rewriting a predicate calculus expression of the constraint check into a form in which the distribution of data is respected. The rewritten predicate will be a conjunction, in which each conjunct represents the constraint check as seen from each individual database. An ideal distribution of the constraint will produce a conjunction in which each conjunct contains no direct references to data stored outside the database which it represents.

We will illustrate this idea using the following simple example illustrated by Figure 2. Suppose we have two databases, one (database A) storing information about people and their ages:

```
declare person ->> entity
declare age(person) -> integer
declare name(person) -> string
```

and another (database B) which stores information about cars and the people who drive them:

```
declare car ->> entity
declare driver(car) -> person
declare min_age(car) -> integer
```

This database also stores details of the minimum allowed age for drivers of particular cars. We also have a global constraint which requires that drivers of cars are aged between the allowed minimum age for their car, and 70:

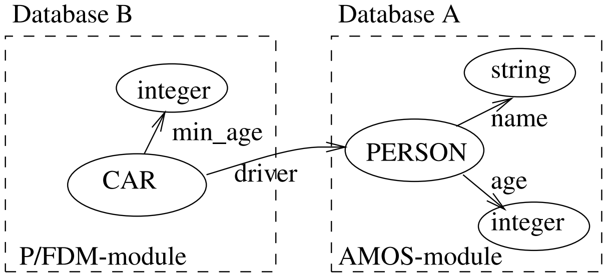

Figure 2. An example of a multidatabase schema.

constrain each c in car
to have age(driver(c)) >= min_age(c) and age(driver(c)) =< 70;

This constraint is expressed in the constraint language extension to Daplex that has been implemented for P/FDM [9]. The equivalent first order logic expression is:

$$((\forall c, p, ma, a) car(c) \wedge driver(c, p)$$
$$\wedge min\_age(c, ma) \wedge age(p, a) \Rightarrow \ a \geq ma \wedge a \leq 70)$$

Here, entity classes are represented as one-place predicates, whose argument is an instance identifier for that class, while attributes are represented by two-place predicates, which take an instance identifier as their first argument and the attribute value as their second. For the purposes of this paper, we refer to such predicates as "data retrieval predicates". The constraint is given a weak-translation [29], which means that the constraint is only required to hold for a given instance when all attributes involved in the constraint are populated for it; hence the positioning of all data retrieval predicates to the *left* of the implies operator.

How, then, can we rewrite this quantified predicate into a form which respects the distribution of the data? The solution we propose is to introduce extra *intermediate* predicates to pass information between the distributed conjuncts. Under this approach, our example constraint becomes:

$$((\forall c, p, ma) \ car(c) \wedge driver(c, p) \wedge min\_age(c, ma)$$
$$\Leftrightarrow \ car\_driving\_person(p, ma))$$
$$\wedge$$
$$((\forall p, ma, a) \ car\_driving\_person(p, ma) \wedge age(p, a)$$
$$\Rightarrow \ a \geq ma \wedge a \leq 70)$$

This form of the constraint is distributed since each conjunct contains only those data retrieval predicates which are local to a particular database, and the intermediate predicates. The first conjunct represents the view of the constraint from the point of view of the database containing information about cars. The second represents the view of the constraint from the point of view of the database containing people's ages. The intermediate predicate used here, `car_driving_person`, restricts the scope of the second conjunct so that the actual constraint check ($a \geq ma \ldots$) is only applied to people who drive cars, even though the `driving` and `min_age` attributes are not directly referred to. The reason for use of if-and-only-if ($\Leftrightarrow$) in the first conjunct is that one must not allow extra "car-driving" persons not implied by the LHS of the equivalence, since this overdoes the constraint.

It is easily shown (using the resolution inference rule) that the truth of this distributed form of the constraint entails the truth of the original, non-distributed version.

Notice that there are two fully distributed forms for this constraint, the alternative being:

$$((\forall p, a) \ person(p) \wedge age(p, a) \ \Leftrightarrow \ person\_has\_age(p, a))$$
$$\wedge$$
$$((\forall p, a, c, ma) \ person\_has\_age(p, a) \wedge car(c)$$
$$\wedge \ driver(c, p) \ \wedge min\_age(c, ma)$$
$$\Rightarrow \ a \geq ma \wedge a \leq 70)$$

In this case, the intermediate predicate acts as a record of which people have populated `age` attributes, rather than which people are drivers.

We can generalise from this example to give the distributed form of a predicate containing an arbitrary number of data retrieval predicates, stored in an arbitrary number of databases. Given an original constraint of the form:

$$(\forall x_1, \ldots, x_n) \ P_1(\ldots) \wedge \ldots \wedge P_j(\ldots)$$
$$\Rightarrow \ P_{j+1}(\ldots) \wedge \ldots \wedge P_m(\ldots)$$

where each P_i, $1 \leq i \leq m$, is some predicate over an arbitrary subset of the variables $\{x_1, \ldots, x_n\}$, and (because of the weak translation) no predicate on the RHS of the implies operator (i.e. no P_i where $j + 1 \leq i \leq m$) is a data retrieval predicate, we can derive the distributed form as:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_{d-1} \wedge C_d$$

where d is the number of databases over which the data is distributed (i.e. one conjunct for each database). Each conjunct C_i, $1 \leq i \leq d - 1$, has the form:

$$(\forall y_1, \ldots) \ Q_1(\ldots) \wedge \ldots \wedge Q_{k_i}(\ldots) \ \Leftrightarrow \ T_i(\ldots)$$

where the predicates Q_l, $1 \leq l \leq k_i$, are the subset of the predicates P_1 to P_j which are locally evaluable on database i, and T_i is an uniquely named intermediate predicate. A predicate P is locally evaluable on a database DB if:

- it is a data retrieval predicate representing an entity or attribute stored at DB
- it is a computed predicate whose input variables are all given values by other locally evaluable predicates on DB.

Grouping the locally evaluable computed predicates with the relevant data retrieval predicates in this way is important in reducing the number of solutions for the intermediate predicate, and therefore in improving constraint checking efficiency.

The final conjunct, C_d, of the distributed constraint contains the constraint check proper, and has the form:

$$(\forall z_1, \ldots)\ Q_1(\ldots) \wedge \ldots \wedge Q_{k_d}(\ldots)$$

$$\Rightarrow\ P_{j+1}(\ldots) \wedge \ldots \wedge P_m(\ldots)$$

Each predicate Q_l, $1 \leq l \leq k_d$, here is either an intermediate predicate T_i or any P_1 to P_j not used in any of the other conjuncts (i.e. left over). The consequent of the final conjunct C_d is exactly the same as the consequent of the original constraint predicate.

Under this approach, intermediate predicates appear just twice in the set of conjuncts, once on the right hand side of a $\Leftrightarrow$ expression, and once on the left hand side of the final conjunct[1]. Each intermediate predicate has a unique name, and its arguments are the set of variables whose values can be generated locally at database DB_i that are required for use in the final constraint check. More formally, if Ps is the set of all predicates appearing in the original constraint, $Ps(DB_i)$ is the subset of Ps which is locally evaluable on DB_i, and $vars(P)$ is the set of variable names appearing in the predicate P in the original constraint, then the set of argument variables for each intermediate predicate T_i is:

$$\{\ v\ \bullet\ (\exists p)\ p \in Ps(DB_i) \wedge v \in vars(p) \wedge$$

$$[(\exists q)\ q \in Ps \wedge q \notin Ps(DB_i) \wedge v \in vars(q)]\ \}$$

The variables appearing in each conjunct are all universally quantified. Again, the application of the resolution rule can be used in a straightforward way to show that the distributed conjunction syntactically entails the original non-distributed constraint.

The above discussion illustrates the construction of a set of conjuncts that represents a fully distributed version of the original constraint. As we saw in our example, several different distributed forms of the constraint can be generated, depending on which database is to contribute to the final conjunct (C_d). In general, this choice should be made so as to minimise the number of solutions that must be generated for the intermediate predicates, and thus the amount of data that must be passed between databases in order to make the constraint check (see Section 6.2).

The distribution method we have described here is so far limited to constraints involving universally quantified variables over a simple conjunction of predicates. While this includes a useful class of integrity constraints, we hope to be able to extend this method to include a wider class of constraints, involving existentially quantified variables, disjunctions and aggregates.

4 Practical Implementation of Constraint Distribution

From a practical point of view, the approach to constraint distribution described in Section 3 presents two main issues: how is the "best" distributed constraint form to be chosen, and how is the chosen form to be used to check constraints in a distributed fashion. In our current implementation, we have taken only a very basic approach to the first issue, using a simple cost model and a few simple heuristics which discourage network traffic required to pass data between databases. A full cost model would also take into account the different characteristics of each database, their respective efficiencies, and the expected update frequency for the data involved in the constraint. We hope to investigate these issues in future implementations.

Our current work, however, has concentrated on the second issue — the way in which a distributed constraint predicate can be used to check the constraint in a distributed way. Since each conjunct of the constraint represents the work required for the constraint check within a particular database, we can confine our attention to translating individual conjuncts into the language required by the corresponding database, and need no longer consider the constraint as a whole. As we have said, there are two forms of conjunct:

the intermediate conjuncts which define intermediate predicates, and

the checking conjunct which expresses the actual constraint check

Each form requires a slightly different treatment. We take an incremental approach to constraint checking, assuming that the constraint is satisfied by the initial database state. We shall use our earlier example, about the allowed ages of car-drivers, to illustrate the method.

The conjunct that is created for a particular database describes the responsibilities of that database in maintaining the integrity of the constraint. An intermediate conjunct, e.g.

[1] A more general approach can also be taken, in which no restrictions are placed on where intermediate predicates can appear (see Section 6.2).

$$((\forall c, p, ma)\ car(c) \wedge driver(c, p) \wedge min\_age(c, ma)$$

$$\Leftrightarrow car\_driving\_person(p, ma))$$

implies that its database is responsible for notifying all other databases whose conjuncts depend on its intermediate predicate of changes to its truth-value. In our example, database A is responsible for notifying database B whenever changes to the *car*, *driver* or *min_age* predicates result in a change to the *car_driving_person* predicate. Exactly how this is done depends upon the mechanisms provided by the underlying database system. In AMOS, for example, a condition action rule can be generated from the conjunct to express the required behaviour for notification of new car drivers, e.g.

```
create rule test as
    when for each car c, person p, integer ma
      where driver(c) = p and min_age(c) = ma
    do new_notify('B', car_driving_person, p, ma);
```

In P/FDM, on the other hand, which uses a more *ad hoc* triggering mechanism, a fragment of Prolog code must be attached to any update which may cause the constraint to be violated. In general, there will be several such updates for each constraint, and so several different constraint fragments must be generated, each specialised with respect to one such update. For example, the following fragment:

```
test(Car, MinimumAge) :-
    getfnval(driver, [Car], Driver),
    new_notify('B', car_driving_person(Driver, MinimumAge).
```

is specialised with respect to the update which populates the `min_age` attribute of a car. The arguments to this fragment are instantiated with the arguments to the update request itself, and the fragment checks that a driver exists for the given car before notifying database B of the new minimum age.

So much for the intermediate conjuncts; how are the checking conjuncts handled? The responsibility implied by a checking conjunct, e.g.

$$((\forall p, ma, a)\ car\_driving\_person(p, ma) \wedge age(p, a)$$

$$\Rightarrow a \geq ma \wedge a \leq 70)$$

is that of checking the given condition, and causing a global rollback if it is found to be violated. An error message should also be generated to warn the initiator of the update that it would have violated the constraint. Again, the exact translation of the conjunct depends upon the target database.

In order to evaluate the constraint check, however, it is necessary to query any intermediate predicates on its left hand side. For example, in the above checking conjunct, when the age of a person `p` is updated, it is necessary to check whether *car_driving_person* is true for `p`, and what value of minimum allowed age corresponds to them. There are three main ways in which the extension of the intermediate predicate may be made available, none of which is necessarily better than the others in all cases:

By method definition: by calling a method (or function or named query) in the responsible database which computes the argument values for which the predicate is true on demand.

For example, database A includes a method called `car_driving_person` which takes a person instance as argument, and which succeeds, returning the relevant minimum age if that person is a car driver.

This is better than using ad hoc queries since the query contained in the body of the method can be optimised once and for all when the method is defined.

By local materialisation: the responsible database explicitly maintains the set of values for which it is true. For example, database A maintains an entity class `car_driving_person` with attributes `person` and `min_age`. Database B queries this class remotely when the age of a person is updated.

By remote materialisation: the responsible database explicitly materialises the set of values for which it is true *in the database which depends on the truth of the intermediate predicate*. This database can then access this data directly, without requiring any network traffic, to determine the validity of the constraint.

For example, database A maintains an entity class `car_driving_person` remotely in database B, by sending update requests to this class whenever changes to its own data require it. Database B then accesses `car_driving_person` directly whenever the value of the `age` attribute is updated.

Each of these approaches has its advantages and disadvantages. The materialised approaches, particularly remote materialisation, result in more efficient constraint checking and reduced network traffic, but at the cost of the extra disk space required to store the materialisation (the classic tradeoff between store and recompute remotely). Remote materialisation is particularly interesting as it means that no network traffic is required to check the constraint. Clearly, an accurate cost model is vital for choosing between these options, and must include details like the estimated size of the materialised set, and the frequency of updates to it. The form of the distributed constraint also has a bearing on which method is most appropriate.

In our current implementation, we have experimented with two of these options: method definition and remote materialisation.

For the first, the constraint compiler generates AMOS functions from the distributed form of the constraint, which are called from P/FDM whenever a relevant update occurs. The AMOS functions are defined in terms of database queries that are optimised at method definition time.

For the second approach, we use the ability of AMOS to activate parameterized condition-action rules for a specific set of instances (not all active databases allow this)[23]. In this case, the compiler generates a condition-action rule containing the bare constraint check (with references to intermediate predicates removed) and the rule is activated only for those values where the intermediate predicates are true. For example, the rule generated from our running example is:

```
create rule test(person p, integer ma) as
        when age(p) < ma or age(p) > 70
        do generate_error("Rule violation ....");
```

This rule is then activated for only those person instances representing car drivers, and the process of remote materialisation becomes the process of maintaining the correct activation of the rule. Rule activation and deactivation for different parameters is controlled by the constraint manager in P/FDM.

On the AMOS side, for parameterized condition-action rules the rule management system maintains an internal table of the instances for which a parameterized rule is activated as in [22]. This internal table corresponds to the remote materialisation of the intermediate predicate.

Parameterized rules are compiled and optimised once when they are defined. Rule activation and deactivation will therefore be very fast since the system then does not need to perform any more rule optimisation. Incremental evaluation techniques have been developed for efficiently testing rule conditions of complex parameterized rules[27].

5 Related Work

Research on active databases and constraints in databases has received substantial attention in recent years [30]. However, most of the research does not consider rules and constraints in distributed or heterogeneous databases. The problem of correctly executing parallel active rules in distributed databases was discussed in [4]. The work in [5] proposes inter-database triggers as a means for maintaining equality constraints between heterogeneous databases. In [16] it is shown how a class of distributed constraints can be broken down into local update checks. [6] proposes some interface protocols for implementing constraints over heterogeneous information systems.

Our work differs from the above by using the theory of weak translation of distributed constraints [29] to provide an implementation of distributed constraints where a cost model directs on which database servers fractions of the constraints should be checked. Furthermore, our implementation is also unusual in that a separate constraint manager (in P/FDM) manages global constraints over a set of database servers in another data model (AMOS) in a multi-database environment. The constraints are heterogeneous in that they are allowed to constrain data in both P/FDM databases and AMOS databases.

6 Conclusions and Future Work

6.1 Active Rule Generation

Active rules are an efficient and obvious way to implement constraints but they suffer from the disadvantage that it is hard to get them right, especially where the collection has evolved, and where several rules work together. This is made worse by the absence of tools for administration of triggers in commercial systems [26]. Thus it is highly desirable to generate rules automatically from a declarative description of the quantified constraint, with the correctness of the translation guaranteed by theory (as in section 3). In this paper we have described an implementation of generated rules, based on the functional data model which has also been used to integrate two inhomogeneous systems.

This work is in the spirit of [1] which showed how to represent quantified semantic constraints in the CoLan language and used Prolog to transform them into triggered rules attached to class descriptors in a single ADAM object-oriented database. We have also seen the utility of Prolog for constraint transformation, but here we have generated Active Rules in the rule language of AMOS, an Active database using a similar object data model, and have distributed them across several AMOS databases.

Although this work has been tested on databases using the functional data model, there are lessons for those using Object-Relational databases with active rules. The OQL language specified by ODMG, or some variant, looks like becoming a standard for Object-Oriented databases. Currently OQL lacks facilities to specify constraints over the database, but it does allow one to formulate queries which are very similar to constraints in P/FDM, and where it is the user's intention to maintain the invariance of the query.

In P/FDM we have already implemented modules using a relational database for data storage [18] and thus it should not be difficult to code-generate active rules for an Object-Relational system from OQL instead of P/FDM. If OQL is generalised to handle multi-databases the techniques presented in this paper could be used to implement distributed constraints over OQL databases.

This work would suggest using Prolog or Lisp for this translation task, even when it is not used directly for data storage.

We have also used the constraint language of P/FDM instead of CoLan (from which it is descended) since it is more efficiently compiled for the bulk data storage used in standard P/FDM modules [8]. In the ADAM-based system we were able to selectively retract CoLan constraints, which caused the corresponding generated rules to be automatically removed from classes, (despite the complicated many-to-many relationship between constraints and classes). This is straightforward to implement in our new architecture [7, 10], and it will be important when we move more to a multidatabase system where local rules may be installed alongside rules generated by a remote host.

6.2 Constraint Distribution Model

The purpose of distributing constraints is to improve the performance of the constraint checks. We have shown how to distribute constraints in a theoretically sound equivalent fashion, using the notion of an intermediate predicate. We have discussed various ways of representing this predicate, based on where the objects and/or values, that should be constrained, are stored. The most promising way is by generating active rules enabled on subsets of instances, using special facilities of AMOS. However, this is not always best, so the chosen way of installing a constraint is then based on an approximate cost model (or heuristic) that calculates the cost for a distribution of the check, and compares that with the cost for a non-distributed constraint check. The cost model needs information about what kinds of checks the system will make and whether the module is held in an embedded, local or remote AMOS server. Currently it assumes that communication costs dominate. After comparing the costs for the alternatives, the most efficient solution is chosen.

The current cost model is rather crude but works well for our present cases. However, in general the optimisation of distributed constraints will require a more precise cost model. For example, the communication cost depends on both the number of messages sent and the amount of data, and the constraint compiler could use a more complete cost model to estimate the total cost of executing the constraint checks. This is similar to what is done in optimisation of distributed queries[21]. However, in contrast to cost models for distributed databases, a cost model for distributed constraints also has to take into account other factors such as update frequencies and update volumes.

One further way in which the efficiency of the distributed constraint check could be improved is to relax the restriction that intermediate predicates may only be used in the final constraint checking conjunct. In some circumstances, this restriction can result in a poor distribution of constraint checking effort. We are currently investigating a more general approach to constraint distribution which allows intermediate predicates to appear anywhere in the distributed constraint (for example, in the definition of another intermediate predicate). However, this approach increases the number of distributed forms that can be generated from a single constraint, and is thus even more dependent upon the availability of an accurate cost model for selecting between them.

6.3 Interoperation of AMOS and P/FDM

Since the AMOS-P/FDM system consists of two different database systems, developed with different software, it is a heterogeneous system. The problem is that the data models of the two systems differed. This meant that the data model of each system had to be extended or limited, to get a compromise to use in the integrated system. Fortunately this was straightforward as they were both originally based on the Functional Data Model. A summary of the different facilities in the AMOS and the P/FDM database systems, and how they affected the integration is given below:

AMOS:

Multidatabase architecture: Makes the distributed system independent of where the AMOSes are located on the Internet. The external interfaces and the inter-database facilities in AMOS make it easy to build a very flexible fast interface between AMOS and P/FDM.

Derived functions: The efficiently precompiled derived functions in AMOS were used to speed up the AMOS part of the interface in the integrated system.

P/FDM:

Module system: P/FDM is prepared to use different types of storage modules. This is the way of splitting up the database schema into parts, stored in either AMOS or any other P/FDM module type.

The key concept: P/FDM has different kinds of keys than AMOS, compound and foreign keys. Problems occur when these keys are declared in P/FDM as containing embedded foreign keys, since this is not currently representable in AMOS. Either equivalents of these types of keys should be implemented in AMOS, or else they should not be allowed in P/FDM when declaring an AMOS module.

6.4 Multidatabase Issues

The main concept of having P/FDM as a top layer of the integrated database system allows two alternative scenarios to be realised. In the first scenario, we have a distributed database with AMOS providing storage modules but with everything controlled and maintained from P/FDM. This has been used successfully to distribute constraints as described above.

A more complicated scenario is possible with a multidatabase session with a network of autonomous AMOSes connected to a top layer, which controls all the distribution. However, in the multidatabase, factors such as autonomy and transparency have to be considered. In AMOS-P/FDM, the local autonomy of every AMOS is high as each of the AMOS servers are their own DBMSes with direct access to the other AMOSes and their data. From the top layer, it is also possible to extend the local database schemas with object relations from one module to another, independent of the type and the location of the module. Since there is only one way of communication, from P/FDM to AMOS, it is only the top layer, P/FDM, that has access to the complete distributed database schema. The module system in P/FDM makes the multidatabase transparent from P/FDM. However, from the local AMOS view there may be extra attributes with strange values representing foreign objects stored in P/FDM, and there may also be extra active rules placed which interfere with local autonomy. When violations of global constraints by local updates occur the system should explain in some way that the local update caused a global constraint violation. This requires further research.

In the multidatabase scenario, all users of local AMOSes are allowed to make updates in their local databases. Since no communication from AMOS to P/FDM exists, the P/FDM top layer has no way of knowing when an local AMOS user is updating its part of the multidatabase. P/FDM has therefore no chance to check if the update has caused any inconsistency in the multidatabase. The only way of maintaining the database consistency is then to force the distributing mechanism always to distribute the constraints by remote materialisation, in order to enforce local checking via the intermediate predicates (which effectively represent the interactions with other databases).

As a result of this, we get an AMOS-P/FDM system where all the AMOS servers in the multidatabase have their own constraints, even when the constraints are installed from the top layer in the multidatabase system. A problem in this multidatabase, and in other multidatabases, is concurrency between users in the system. Updates have to be atomic to not allow one user's update affecting another user's update. Also, when an update is violating a constraint, so that the update has to be undone, the system must have the chance to complete the undoing of the update before another update starts changing the data in the database.

One known solution to this concurrency problem is to use two-phase commit where an update will first lock all databases involved in the update, then execute the update and finally unlock the databases. To increase the concurrency a better method could be to let the distributed rules trigger distributed repair actions for the violated distributed constraints. There are also special problems with interference among distributed active rules [4].

The constraint language of P/FDM is very general [9] including existential quantifiers, disjunctions and aggregates. This paper only considers distribution of conjunctive constraints and so more theoretical work is needed, extending the work of [5, 16].

Acknowledgements

Stefan Grufman and Fredrik Samson were enabled to study in Aberdeen under the EU Erasmus scheme. We are also grateful for EU funding for Human Capital and Mobility under the ACTNET (Active Database Net of Excellence) consortium, of which we are members. This has provided travel funds for Prof. Risch and Prof. Gray to visit each other. Suzanne Embury's work is supported by a grant GR/J44162 from EPSRC in UK. We are also grateful to Graham Kemp, Martin Sköld and Magnus Werner for discussions and help in implementation.

References

[1] N.Bassiliades, P.M.D.Gray: CoLan: A Functional constraint language and its implementation, *Data & Knowledge Engineering*, Vol. 14, No. 3, 1994.

[2] M.W.Bright, A.R.Hurson, S.H.Pakzad: A Taxonomy and Current Issues in Multidatabase Systems, *IEEE Computer*, Vol. 25, No. 5, 1992.

[3] R.G.G.Cattell (editor): *The Object Database Standard: ODMG-93, Release 1.2*, Morgan Kaufmann, 1995.

[4] S.Ceri, J.Widom: Production Rules in Parallel and Distributed Database Environments, *Proc. of the 18th Conference on Very Large Databases, VLDB'92*, Vancouver, Canada, pp 339-351, 1992.

[5] S.Ceri, J.Widom: Managing Semantic Heterogeneity with Production Rules and Persistent Queues, in *Proc. of the 19th Conference on Very Large Databases, VLDB'93*, Dublin, Ireland, pp 108-119, 1993.

[6] S.S.Chawathe, H.Garcia-Molina, J.Widom: A Toolkit for Constraint Management in Heterogeneous Information Systems, *Proc. 12th International Conference on Data Engineering, ICDE'96*, New Orleans, Louisiana, pp 56-65, 1996.

[7] S.M.Embury, P.M.D.Gray, N.Bassiliades: Constraint Maintenance using Generated Methods in the P/FDM Object-Oriented Database, *Rules in Database Systems:*

Proc. 1st International Workshop -RIDS'93,N W Paton & M H Williams (Eds), Springer-Verlag, London, pp 364-381, 1994.

[8] S.M.Embury, P.M.D.Gray: Planning Complex Updates to Satisfy Constraint Rules Using a Constraint Logic Search Engine, *Proc. of the 2nd International Workshop on Rules in Database Systems - RIDS'95*, T. Sellis (ed.), Springer-Verlag LNCS 985, pp 230-244, 1995.

[9] S.M.Embury, P.M.D.Gray: Compiling a Declarative High-Level Language for Semantic Integrity Constraints, in *Proc. of the 6th IFIP TC-2 Working Conference on Data Semantics (DS-6)*, R. Meersman and L. Mark (eds.), Chapman & Hall, 1995.

[10] S.M.Embury, P.M.D.Gray: A Modular Compiler Architecture for a Data Manipulation Language, in *Proc. of the 14th British National Conference on Databases (BNCOD-14)*, R. Morrison and J. Kennedy (eds.), Springer-Verlag, Edinburgh, U.K., pp 170-188, 1996.

[11] G.Fahl, T.Risch, M.Sköld: AMOS - An Architecture for Active Mediators. *Proc. International Workshop on Next Generation Information Technologies and Systems, NGITS'93*, Haifa, Israel, pp 47-53, 1993.

[12] G.Fahl, T.Risch: Query processing over object views of relational data, to be published in *VLDB Journal*.

[13] D.Fishman et al.: Overview of the IRIS DBMS, in W.Kim, F.H.Lochovsky (eds.): *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Addison-Wesley, 1989.

[14] S.Flodin, T.Risch: Processing Queries with Invertible Late Bound Functions, *Proc. of the 21st Conference on Very Large Databases, VLDB'95*, Zürich, Switzerland, pp 335-344, 1995.

[15] P.M.D.Gray, K.G.Kulkarni, N.W.Paton: *Object Oriented Databases - a Semantic Data Model Approach*, Prentice-Hall, 1992.

[16] A.Gupta, J.Widom: Local Verification of Global Integrity Constraints in Distributed Databases, *Proc. of the 1993 ACM SIGMOD Conference*, Washington, DC, pp 40-58, 1993.

[17] Z.Jiao, P.M.D.Gray: Optimisation of methods in a navigational query language, *Proc. 2nd International Conference on Deductive and Object-Oriented Databases, DOOD'91*, Springer-Verlag, pp 22-42, 1991.

[18] G.J.L.Kemp, J.J.Iriarte, P.M.D.Gray:Efficient Access to FDM Objects Stored in a Relational Database, *Directions in Databases: Proc. of the Twelfth British National Conference on Databases - BNCOD12*, D.S.Bowers (ed.), Springer-Verlag, New York, pp 170-186, 1994.

[19] T.Landers, R.Rosenberg: An Overview of Multibase, in H-J Schneider (ed): *Distributed databases*, North-Holland, 1982.

[20] W.Litwin, L.Mark, N.Roussopoulos: Interoperability of Multiple Autonomous Databases, *ACM Computing Surveys*, Vol. 22, No. 3, 1990.

[21] M.T.Özsu, P.Valduriez: *Principles of Distributed Database Systems*, Prentice-Hall, 1991.

[22] T.Risch: Monitoring Database Objects, *Proc. of the 15th Conference on Very Large Databases, VLDB'89*, Amsterdam, the Netherlands, pp 445-453, 1989.

[23] T.Risch, M.Sköld: Active Rules based on Object-Oriented Queries, *IEEE Data Engineering Bulletin*, Vol. 15, No. 1-4, Dec. 1992.

[24] D.W.Shipman: The Functional Data Model and the Data Language DAPLEX, *ACM Transactions on Database Systems*, Vol. 6, No. 1, 1981.

[25] A.P.Sheth, P.A.Larson: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, *ACM Computing Surveys*, Vol. 22, No. 3, 1990.

[26] E.Simon, A.Kotz-Dittrich: Promises and Realities of Active Database Systems, *Proc. of the 21st Conference on Very Large Databases, VLDB'95*, Zürich, Switzerland, pp 642-653, 1995.

[27] M.Sköld, T.Risch: Using Partial Differencing for Efficient Monitoring of Deferred Complex Rule Conditions, *Proc. 12th International Conference on Data Engineering, ICDE'96*, New Orleans, pp 392-401, 1996.

[28] M.Stonebraker: *Object-Relational DBMSs - the next great wave*, Morgan Kaufmann, 1996.

[29] S.D.Urban: ALICE - an Assertion Language for Integrity Constraint Expression, in *Proc. International Conference on Computer Software and Applications(Orlando)*, 1989.

[30] J.Widom, S.Ceri: *Active Database Systems - Triggers and Rules For Advanced Database Processing*, Morgan Kaufmann, 1996