Flexible Constraint Management for Autonomous Distributed Databases*

Sudarshan S. Chawathe, Hector Garcia-Molina and Jennifer Widom
Computer Science Department
Stanford University
Stanford, California 94305–2140
E-mail: {chaw,hector,widom}@cs.Stanford.edu

1 Introduction

When databases inter-operate, integrity constraints arise naturally. For example, consider a flight reservation application that accesses multiple airline databases. Airline A reserves a block of X seats from airline B. If A sells many seats from this block, it tries to increase X. For correctness, the value of X recorded in A's database must be the same as that recorded in B's database; this is a simple distributed copy constraint. However, the databases in the above example are owned by independent airlines and are therefore autonomous. Typically, the database of one airline will not participate in distributed transactions with other airlines, nor will it allow other airlines to lock its data. This renders traditional constraint management techniques unusable in this scenario. Our work addresses constraint management in such autonomous and heterogeneous environments.

In an autonomous environment that does not support locking and transactional primitives, it is not possible to make "strong" guarantees of constraint satisfaction, such as a guarantee that a constraint is always true or that transactions always read consistent data. We therefore investigate and formalize weaker notions of constraint maintenance. Using our framework it will be possible, for example, to guarantee that a constraint is satisfied provided there have been no "recent" updates to pertinent data, or that a constraint holds from 8am to 5pm everyday. Such weaker notions of constraint satisfaction requires modeling time, and consequently, time is explicit in our framework.

Most previous work in database constraint management has focused on centralized (for e.g., [?]) or tightly-coupled and homogeneous distributed databases (for e.g., [1], [2], [3], [4]). The multi-database transaction approach to constraint management weakens the traditional notion of correctness of schedules [5], [6]. This approach cannot, however, handle a situation in which different databases support different interfaces. In modeling time, our work has similarities with some work in temporal databases [7] and temporal logic programming [8]. Our approach is closer to the event-based specification language in RAPIDE [9].

*Research sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Material Command, USAF, under Grant Number F33615-93-1-1339. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of Wright Laboratory or the US Government. This work was also supported by the Center for Integrated Systems at Stanford University, and by equipment grants from Digital Equipment Corporation and IBM Corporation.

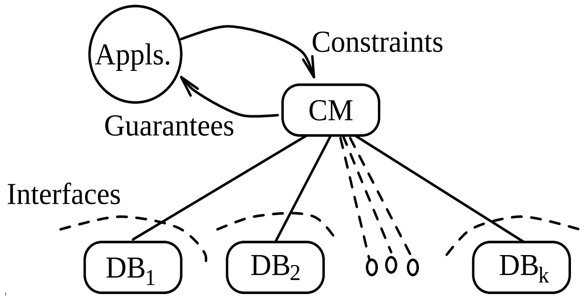

Figure 1: Constraint Management Architecture

In this paper, we give a brief overview of our formal framework for constraint management in autonomous systems and describe the constraint management toolkit we are building. The details of the underlying execution model, semantics of events, and syntax and semantics of the rule language may be found in [10].

2 Formal Framework

In this section, we present an outline of our formal framework for constraint management. Our framework assumes the simplified system architecture shown in Figure 1.[1] Each database chooses the *interface* it offers to the constraint manager (CM) for each of its data items (involved in an inter-database constraint). The interface specifies how each data item may be read, written and monitored by the CM. Applications inform the CM of constraints that need to be monitored or enforced. Based on the constraint and the interfaces available for the data items involved in the constraint, the CM decides on the constraint management *strategy* it executes. This strategy tries to monitor or enforce the constraint as well as possible using the interfaces offered by the local databases. The degree to which each constraint is monitored or enforced is formally specified by the *guarantee*. We describe interfaces, strategies and guarantees below.

2.1 Interfaces

The interface for a data item involved in a constraint describes how that data item may be read, written, and/or monitored by the constraint manager. Interfaces are specified using a notation based on *events* and *rules*. For example, consider a simple write interface for a data item X. This interface promises to write the requested value to X within, say, 5 seconds. We express this as $WR(X,b)@t \rightarrow W_g(X,b)@[t,t+5]$. Here $WR(X,b)@t$ represents a "write-request" event which requests the operation $X \leftarrow b$, and which occurs at some time t. The rule says that whenever such an event occurs, a "write"[2] event, $W_g(X,b)$ occurs at some time in the interval $[t,t+5]$. The interfaces for the data items involved in inter-database constraints are specified by the database administrator of each database, based on the level of access to the database he or she is willing to offer the CM. Currently, we rely on the users of our framework to verify that the interfaces specified do faithfully represent the actual systems.

[1]Note that we assume a centralized constraint manager for simplicity in presentation only; the constraint manager is actually distributed.

[2]$W_g()$ is a *generated* write event which occurs as the result of CM activity, to be distinguished from spontaneous write events, $W_s()$, which occur due to user/application activity in the underlying database.

2.2 Strategies

The strategy for a constraint describes the algorithm used by the constraint manager to monitor or maintain the constraint. Like interfaces, strategies are specified using a notation based on events and rules. In addition to performing operations on data items involved in a constraint, strategies may evaluate predicates over the values of data items (obtained through database read operations) and over private data maintained by the Constraint Manager.

As a simple example, consider the following strategy description, which makes a write request to Y whenever it receives a "notify" event from X:[3] $N(X, b)@t \rightarrow WR(Y, b)@[t, t + 7]$.

Once our framework has been used to specify a strategy, and to verify the correctness of a guarantee, then the rule-based strategy specification is implemented using the host language of the Constraint Manager. This is a simple translation, and may be done using a rule engine.

2.3 Guarantees

A guarantee for a constraint specifies the level of global consistency that can be ensured by the Constraint Manager when a certain strategy for that constraint is implemented. Typically a guarantee is *conditional*, e.g., a guarantee might state that if no updates have recently been performed then the constraint holds, or that if the value of a CM data item is true then the constraint holds. Guarantees are specified using predicates over values of data items and occurrences of certain events. For example, consider the following guarantee for a constraint $X = Y$: $(\text{Flag} = \text{true})@t \Rightarrow (X = Y)@@[t - \alpha, t - \beta]$. This guarantee states that if the (Boolean) data item Flag is true at some time t, then $X = Y$ (at all times) during the interval $[t - \alpha, t - \beta]$. Note that this guarantee is weaker than a guarantee that $X = Y$ always, which is very difficult to make in the heterogeneous, autonomous environments we study.

3 A Constraint Management Toolkit

We are building a toolkit that will permit constraint management across heterogeneous and autonomous information systems. For example, this toolkit will allow us to maintain a copy constraint spanning data stored in a Sybase relational database and a file system, or an inequality constraint between a *whois*-like database and an object-oriented database. We give a brief overview of this toolkit in this section.

3.1 Architecture

Figure 2 depicts the architecture of our constraint management toolkit. The Raw Information Source (RIS) is what is already present at each site (for example, a relational database, a file system, or a news feed.) The RISI is the interface offered by each RIS to its users and applications. For example, for a Sybase database, the RISI is based on a particular dialect of SQL, and includes details on how to connect to the server.

The CM-Translator is a module that implements the CM-Interface (the interface discussed in Section 2.1) using the RISI. The CM-RID is a configuration file used to specify (1) which types of CM-Interface (selected from a menu of pre-compiled interface types) are supported by the CM-Translator, and (2) how these interfaces are implemented using the underlying RISI.

[3] A notify event is an event type representing the database containing X notifying the CM of a write to X. Thus $N(X, 5)$ means that a write $X \leftarrow 5$ occurred.

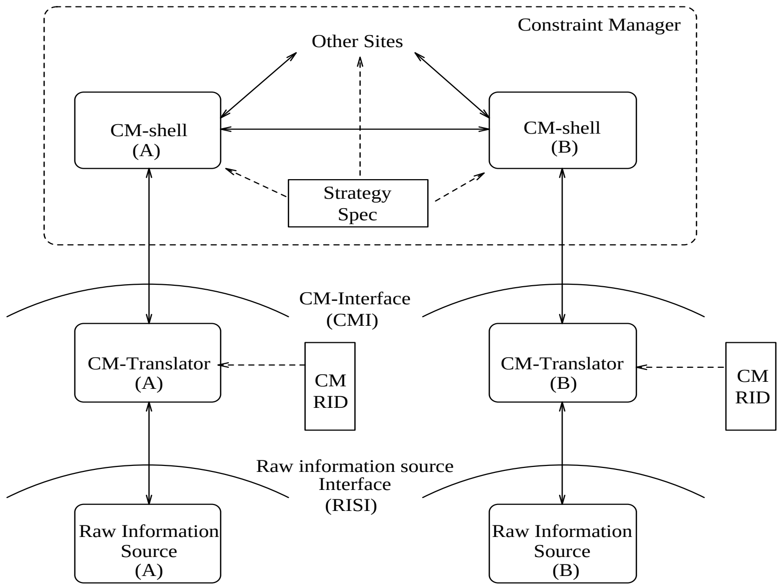

Figure 2: Constraint Management Toolkit Architecture

The CM-Shell is the module that executes the constraint management strategies described in Section 2.2. Since we specify strategies using a rule-based language, the CM-Shells are distributed rule engines that are configured by a Strategy Specification.

3.2 Application

We now describe how database administrators would use our toolkit to set up constraint management across autonomous systems. The database administrators at each site first decide on the CM-Interfaces they are willing to offer, selected from menu of pre-compiled interfaces provided by the toolkit. For example, if the underlying RIS provides triggers, then a notify interface may be offered (where the CM is notified of updates); if not, perhaps a read/write interface can be offered. The choice also depends on the actions the administrator wants to allow. For instance, even if the RIS allows database updates, the administrator may disallow a write interface that lets the CM make changes to the local data.

Each CM-RID file records the interfaces supported, as well as the specification of the RIS objects to which the interface applies. The CM-RID is also the place where site-specific translation information is stored. This includes, for example, the name of the Sybase data server that holds the data and its port number, how a read request from the CM is translated into an SQL query, how a request to set up a notify interface is translated to commands that set up a trigger, and so on.

Next, the administrator uses a Strategy Design Tool (not shown in Figure 2) to develop the CM strategy. This tool takes as input the inter-database constraints, and based on the available interfaces, suggests strategies from its available repertoire. For each suggested strategy, the design tool can give the guarantee that would be offered. The result of this process is the Strategy Specification file, that is then used by the CM-Shells at run time. (Knowledgeable administrators can write their Strategy Specifications, bypassing the Design Tool.)

At run-time, the CM-Shells execute the specified strategy based on the event-rule formalism. The CM-Translators take care of translating events to site-specific operations, and vice versa.

4 Conclusion

We address the problem of constraint management across heterogeneous and autonomous systems. We have argued that this is a problem of practical importance, and that it is not readily amenable to traditional constraint management techniques. Constraint management in autonomous environments requires weaker notions of consistency than those found in literature. We have proposed an event-based formal framework which allows us to express these weaker notions of consistency, and in which time is explicitly modeled. We have also described a constraint management toolkit that we are currently building, which demonstrates some the practical aspects of our work. In the future, we plan to work on expanding our framework to handle more complex interface and constraint types, including those with quantification over data items and probabilities associated with them.

References

[1] Eric Simon and Patrick Valduriez. Integrity control in distributed database systems. In *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, pages 621–632, 1986.

[2] Paul Grefen. Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem. In *Proceedings of the International Conference on Very Large Data Bases*, pages 581–591, Dublin, Ireland, August 1993.

[3] Stefano Ceri and Jennifer Widom. Managing semantic heterogeneity with production rules and persistent queues. In *Proceedings of the International Conference on Very Large Data Bases*, pages 108–119, Dublin, Ireland, August 1993.

[4] Marek Rusinkiewicz, Amit Sheth, and George Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, 24(12):46–51, December 1991.

[5] Ahmed Elmagarmid, editor. *Special Issue on Unconventional Transaction Management,* Data Engineering Bulletin 14(1), March 1991.

[6] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, 1(2):181, October 1992.

[7] Richard Snodgrass. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.

[8] Martin Abadi and Zohar Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8(3):277–295, 1989.

[9] David C. Luckham et al. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 1994.

[10] Sudarshan S. Chawathe, Hector Garcia-Molina, and Jennifer Widom. Constraint management in loosely coupled distributed databases. Technical report, Computer Science Department, Stanford University, 1993. Available through anonymous ftp from `db.stanford.edu:pub/chawathe/1993/cm-loosely-coupled-dbs.ps`.