Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1996 Proceedings

Americas Conference on Information Systems (AMCIS)

8-16-1996

Sybil: A System for the Incremental Evolution of Distributed, Heterogeneous Database Layers

Roger King
University of Colorado, roger@cs.colorado.edu

Michael Novak
University of Colorado, novak@cs.colorado.edu

Sybil: A System for the Incremental Evolution of Distributed, Heterogeneous Database Layers

Roger Kingand Michael Novak
Department of Computer Science
University of Colorado
Campus Box 430 Boulder, CO 80309-0430
roger@cs.colorado.edu, novak@cs.colorado.edu
This research was funded under DARPA order B126 (Rome AF contract F30602-94-C-0253) and under DARPA/NASA contract NAG2-862.

Motivation

Many of today's large applications are persistent and use one or more commercial database management systems in their persistence layer. However, due to the inability to physically and logically interconnect these databases, they are typically accessed independently. This causes applications to be artificially partitioned and to maintain partially redundant (and usually inconsistent) databases. But, with the advent of the ``information highway'' and the increasing prevalence of high speed networks, application developers can now do true distributed programming (as opposed to using the Internet simply for email and file transmissions). This means that hardware to properly support multidatabase applications is finally in place. The key problem that is still facing the developers of large, persistent applications is: can we provide software that will allow multiple databases to be treated as a consistent interconnected entity capable of being methodically maintained and evolved? If so, then the vast majority of very large persistent applications will benefit, since logically, they would be most naturally engineered on top of highly diverse, distributed database platforms.

For such an approach to work, the persistence layer, which up to now has been treated as the ``hidden half'' of an application, must gain first class status. Thus, the persistence layer needs to be capable of rapidly evolving to match the rapid evolution cycle of modern applications. This evolution may include reconfiguring legacy database systems (e.g., altering the storage manager of an existing system to cluster complex objects), adding new database functionality (e.g., adding object-oriented capability to a relational system),maintaining and updating data semantics (such as schemas and constraints), interconnecting multiple database systems, and including legacy components with varying database needs (such as introducing an object-oriented application into a relational application). We felt it was vital to tackle this hidden half of the evolution problem for three reasons. First, since it is quite common for applications to share one or more databases, these databases are often the focal point of several applications. Thus, as the bridge between the applications, the persistence layer is the natural vehicle for ensuring consistent inter-application evolution. Second, the application semantics can often be more easily tracked via the persistence layer, due to the fact that database systems are specifically designed to provide many semantic clues via constructs such as schemas, constraints and structured queries. And third, application evolution can be done faster and cheaper if applications can be relieved of the expensive task of manually evolving their data needs in an add hoc manner.

Current evolution research has focused almost entirely on applications, and not the persistence layers under them. A main objective of the Sybil project is to support the continuous evolution of persistence layers to meet rapidly changing application needs. One technology that lends itself to attacking some of these problems is the area of heterogeneous databases. However, we feel that this technology is not totally sufficient, for the following reasons. Although several systems, such as Pegasus [SAD+95] and UniSQL/M[KGK+95] are capable of storing multi-model data, these systems all force the user to view data through one data model (generally object-oriented). To make viewing data in this manner possible, the various schemas must be translated into one data model, then integrated into a global schema. But schema transformation and integration must be done manually, and are therefore very costly, especially when dealing with large legacy databases. Also, the types of applications we are interested in need to manage

multiple models of data in an explicit fashion (not through the eyes of a uniform model),and usually only subparts of the various schemas or databases are related. Thus, we feel that data should accessible via the tools(e.g., query languages) provided by each database, not only through a common interface. We also feel that schema translation and integration is both unnecessary and undesirable. In fact there is a current trend toward the development of distributed database systems which maintain local autonomy and do not enforce complete global synchronization of schemas [SAL+96].

In examining a number of real world, large-scale applications in the insurance and telecommunications industry, we have seen multiple examples of large organizations spending significant amounts of money(often millions of dollars) trying to engineer a global, integrated solution, only to discover it is much too expensive to complete and will be much too rigid. Furthermore, many (if not most) heterogeneous database applications require only very narrow interconnections between diverse databases. For example, many banks, insurance companies, and telecommunications companies, because they are large, wide-spread organizations that have grown substantially over time, find themselves with dozens (or often hundreds) of databases containing customer information. While it might make sense to create a complete, integrated view of all these databases, only a small percentage of the data generally needs to be interrelated. Also, the users of those systems often prefer to keep their current model, schema, and application environment as intact as possible, rather than converting to a single integrated view. Furthermore, such database environments change so rapidly that small incremental changes make more sense than one large integration that becomes out of date a short time later.

The Sybil Approach

Therefore, we advocate incremental evolution of the database layer(based on integrating only those pieces of the schema that are somehow semantically interrelated). Making persistence evolution an incremental process will not simply reduce the cost of doing this evolution, it will in many instances turn intractable processes into tractable ones - rapid response situations must be managed in hours and days, not months and years.

Thus, a primary Sybil goal is providing a methodology for incrementally specifying and evolving a persistence layer (consisting of one or more databases, with one or more data models and schemas)throughout its life cycle, and throughout the life cycle of the applications running on top of it. In order to do this we need to be able to both capture the application semantics at creation time and to insure that the database system evolves consistently with the application(s) running on top of it.

Overall, we want to support two sorts of database layer evolution. First, traditional data models must be extensible with the capabilities of newer models. This will allow database users to incrementally make use of new modeling functionalities without having to make drastic, all-at-once changes in their environment. This will also allow newer sorts of database management systems to provide these capabilities directly, thus avoiding the extreme cost of extending traditional database systems with new capabilities. Second, persistent system layers must be extensible in that users must be able to add database components and remove old components as the encompassing application layer evolves.

Sybil provides these evolution capabilities by loosely coupling databases into alliances tailored for a specific application (or set of applications). This coupling is done by interrelating those portions of the databases that are somehow semantically related for the application. These interrelationships are maintained via rule-based mediators [Wie92] that interconnect component databases. Mediators are implemented by using the native constructs of the component database systems and a rule execution engine supported by Sybil.

Sybil uses three heuristics in developing lightweight alliances. First, users should be able to easily focus on only the logical interconnections that are strictly necessary in their data processing environment. Significantly, the designer is not required to create a complete, global view that logically encompasses all of the component databases. Thus, the time consuming, highly human-driven process of component schema

translation (into a common model) and schema integration (into a common schema) will be avoidable. Second, in most cases, we use the native constructs of the component databases to ``cobble together'' alliances; this will avoid the introduction of complex, new languages for the specification of alliances. Third, we leverage off of new, emerging object/relational standards as a way of defining and supporting some of our interdatabase constructs. Again, this prevents the user from having to learn substantive new languages and tools.

The Sybil Prototype

The current Sybil prototype supports the following constructs: interdatabase views, interdatabase constraints, and propagations of updates from one database system to another. These three constructs are ideal to test the alliance concept both because they support a large percentage of the semantic relationships that are necessary for the types of database connections we are interested in and because all three can be evolved fairly easily.

For each of these constructs, Sybil must overcome the fact that it is dealing with several different data models. This problem has two significant parts: 1) How do we specify views, constraints, and updates across systems with different data models? And, 2) How do we use the native tools of the various database systems to implement such functionalities?

By heterogeneous views, we mean a combined view that is inherently multi-model. For example, a relational database may contain pricing and ordering information for engine parts, while an object-oriented database may contain schematics for various engine components. An alliance developer might want to specify highly specialized sorts of interconnections, such as allowing the tuples in a relation to be automatically referenced as attributes of an object-oriented database. In general, we keep such views very narrow in scope, and semantically merge only the specific parts of the component schemas that must be interrelated. We draw on known results in query decomposition and result integration for accessing virtual views. There are two primary problems that we are currently attacking: categorizing the specialized sorts of heterogeneous views that would be of value to multi-database users, and extending single-model view specification and query specification languages to be multi-model.

The second sort of alliance construct is heterogeneous database constraints. An example might be requiring that a customer address in one database be consistent with an address in another database, or that the total number of outstanding customer orders across multiple databases not exceed a certain number. We are currently examining two possible approaches to specifying and maintaining inter-database constraints. The first involves using the native constraint languages of the component databases then bridging them with semantic data mappings. Part of a constraint would be specified on one database in one language, and the rest on another database in another language. For simple constraints (those that involve fairly simple mappings between databases) this is a reasonable approach. For more complex constraints, however, this approach rapidly becomes quite clumsy. The second approach involves specifying the constraint using an existing rule-based, multi-database constraint specification mechanism([CW93]). But, such a facility would have to be augmented to handle heterogeneity. We are currently pursuing both of these approaches, and intend to use the one that leads to a solution that seems to be both simple and reasonable broad in its functionality.

The third sort of alliance construct is update propagations across multiple database systems. As an example, if the address of a client changes in one database, we are likely to want to change it in other databases that reference the same client. We will draw upon existing DBMS support tools, including triggers and transaction management. We are likely to take a very simple approach to propagating updates, namely that of globally locking all involved database systems while the propagation is in progress. This is to avoid the very difficult problem of global, two phase, heterogeneous transaction support. However, assuming that the interconnections are lightweight, not much global locking will be required. We also note that more and more DBMS products are providing triggering and rule capabilities, and so the underlying support should be readily available.

Bibliography

[CW93] S. Ceri and J. Widom, "Managing Semantic Heterogeneity with Production Rules and Persistent Queues", VLDB Conference Proceedings, 1993.

[KGK+95] W. Kelley, S. Gala, W. Kim, T. Reyes, and B.Graham, "Schema Architecture of the UniSQL/M Multidatabase System", in Modern Database Systems, W. Kim, editor, Addison-Wesley, 1995, pages 621-648.

[SAD+95] M. Shan, R. Ahmed, J. Davis, W. Du, and W. Kent, "Pegasus: A Heterogeneous Information Management System", in Modern Database Systems, W. Kim, editor, Addison-Wesley, 1995, pages 621-648.

[SAL+96] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu, "Mariposa: A Wide Area Distributed Database System", The VLDB Journal, v5,

n1, January 1996, pages 48-63.

[Wie92] G. Wiederhold, "Mediators in the Architecture of Future Information Systems", IEEE Computer, March 1992, pages 38-49.