Change Management in Heterogeneous Semistructured Databases (Demonstration Description)*

Sudarshan S. Chawathe   Vineet Gossain   Xiang Liu   Jennifer Widom   Serge Abiteboul

Computer Science Department, Stanford University, Stanford, California 94305

{chaw,vin,danliu,widom,abitebou}@cs.stanford.edu

1 Introduction

The vast amount of information available on the World-Wide Web has sparked great interest in the subject of storing and querying heterogeneous and semistructured data. *Heterogeneous* data is characterized by a high degree of autonomy, an absence of a common data model, an absence of standard database control facilities (such as transactions), and differing modes of access [SL90]. Heterogeneous and other Web-accessible data frequently is *semistructured*, meaning that the data may be irregular or incomplete [Abi97, AQM+96, BDHS96]. In addition to offering access to large amounts of heterogeneous and semistructured information, the Web allows this information to change at any time and in any way. These rapid and often unpredictable changes to the information create a new problem: one of detecting, representing, and querying these changes.

As an example, consider a Web site offering weather information [TWU]. Recent work on integration of semistructured data allows us to construct a *wrapper* that presents the information at this site using a uniform graph-structured data model called *OEM* [PGMW95], and allows the data to be queried through the wrapper. For example, we can ask "find cities in the Bay Area that had more than half an inch of rain overnight." However, in general people are more interested in weather trends than in past weather data. If the overnight rainfall was half an inch yesterday but was three inches the day before, then chances are this storm is tapering off. However, if there was half an inch yesterday but no rain the day before, this storm may be building. So, the query we would like to ask is "find cities in which the overnight rainfall has increased by half an inch over the past two days." This query requires access to previous states of the data, a feature that is not supported by most current semistructured and heterogeneous data management systems, nor by the Web itself. Similarly, perhaps we would like to be notified automatically every time the overnight snowfall in the Sierras has been at least six inches for three nights in a row. Such a "trigger" requires not only access to previous states of the data, but also a method to detect when the trigger should fire.

At Stanford we are addressing the problem of change management in heterogeneous semistructured databases with a system consisting of three main components, supported by two other Stanford projects. See Figure 1. The TDIFF component uses tree differencing algorithms to detect changes between snapshots of semistructured data [CRGMW96, CGM97]. The CORE component allows clients to store and query semistructured data and their changes [CAW98]. CORE uses another Stanford database project, *Lore* [MAG+97], to store the data and changes. The QSS component supports powerful and flexible subscriptions over heterogeneous data sources, notifying

*This work was supported by the Air Force Rome Laboratories under DARPA Contract F30602-95-C-011 and by an equipment grant from IBM Corporation

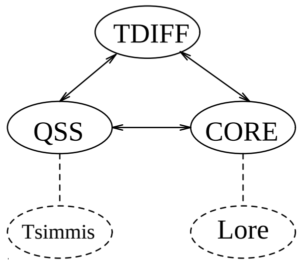

Figure 1: Components of our change management system

subscribers of changes of interest. QSS exploits the *Tsimmis* [CGMH+94] project at Stanford, which gives us the ability to wrap and integrate diverse heterogeneous sources.

Now we use our weather scenario to roughly illustrate how the components work together. The TDIFF component is used to detect relevant changes (such as changes in rainfall measure). CORE can store these changes and clients can query the history, including asking questions of the form: "has the overnight rainfall changed in the past week?" Clients wishing to be notified automatically of certain changes, such as trends in overnight snowfall, may do so by registering a subscription with the QSS component. QSS automatically polls the source data using Tsimmis, detects changes using TDIFF, and stores and queries these changes appropriately using CORE.

2 Infrastructure and Components

The supporting Tsimmis project focuses on the integration of heterogeneous data and the use of a standard data model and query language over many different types of sources. A Tsimmis wrapper presents the data from a source in a uniform graph-based data model called the *Object Exchange Model* (OEM) [PGMW95]. Queries posed over the (virtual) OEM representation of the source data are translated to native source-specific queries, and the results of these queries are translated back to an OEM representation before being returned to the wrapper client. Wrappers for a variety of data sources can be built quickly using template-based wrapper-generators [PGGMU95, HGMC+97]. In addition, Tsimmis allows clients to access (virtually) integrated data from multiple heterogeneous data sources through its *mediators*, described in [PGMU96, PAGM96]. Wrappers and mediators provide identical interfaces to clients.

Lore (for Lightweight Object Repository) is a database system designed to store and query semistructured data in OEM[MAG+97]. The Lore system offers the functionality of a traditional DBMS including OQL-like queries, multi-user support and recovery, as well as non-traditional functionality such as path expression matching and extensive type coercion. The *Lorel* query language [AQM+96] is used to query Lore data, and can be viewed as an extension to OQL.

2.1 TDIFF: Detecting Changes

The TDIFF module is used to detect changes in the heterogeneous semistructured databases being monitored by our system. In conventional databases, detecting changes to data is made easier by the availability of facilities such as transaction logs, triggers, etc. However, in heterogeneous data sources, such as Web sites, such facilities often are absent. Even in cases where these facilities are available, they may not be accessible. Therefore, in practice, we often need to detect changes by comparing two or more snapshots of the database (or a portion thereof) using differencing algorithms. Here, the semistructured nature of the data causes problems, since finding and representing changes in semistructured data is much harder than in, say, structured relational tables. We have developed algorithms to detect changes between snapshots of tree-structured data, and the TDIFF component is an implementation of these algorithms. For further information on the differencing algorithms, see [CRGMW96, CGM97]. Since the results of Tsimmis queries over wrapped sources are tree-structured, TDIFF is well-suited for detecting changes in any wrapped (or mediated) source.

2.2 CORE: Change Object Repository

While TDIFF gives us the ability to detect changes between two snapshots of a semistructured data source, we would also like the ability to store, browse, and query this temporal data. Existing temporal models do not support the irregularity and incompleteness of semistructured data, so we have developed the *Delta-OEM* (DOEM) data model. DOEM extends OEM by allowing *annotations* on the nodes and edges in the graph representation of OEM. These annotations represent the history of the node or edge, and collectively represent the complete history of the database. Nodes can have *create* and *update* annotations, storing the time of the modification and, for updates, the old and new values of the object. Edges can have *add* and *remove* annotations, storing the time of the modification. The *Chorel* (*Change Lorel*) query language extends the functionality of Lorel, allowing a user to query DOEM data, accessing both historical and current data. For more information on DOEM and Chorel, refer to [CAW98].

The CORE component is our implementation of the DOEM data model and the Chorel query language. We have implemented them as an extension to Lore. DOEM data is encoded as OEM data (by representing annotations as special OEM objects) and is stored in a Lore database. Chorel queries are then translated to Lorel queries over the OEM-encoded DOEM database. Finally, the encoded results of the Lorel query are translated back into their DOEM representation before being returned to the user. CORE offers a graphical user interface that allows users to issue Chorel queries and browse query results and their histories.

2.3 QSS: Query Subscription Service

A flexible system for detecting and reporting changes must offer a number of options and parameters. For example, one can detect changes to Web sites in a number of ways. Some Web sites offer users the option of receiving e-mail when significant changes are made. Others may need to be polled, and the polling interval may depend on the context. For example, a traffic site is best polled every few minutes, while a site with daily ski reports is best polled once every morning. For sites that change infrequently, the changes may best be detected by user request. Users may also wish to learn about changes in a number of ways. Some users may wish to be notified whenever changes of a certain kind occur. Others may wish to receive a daily or weekly report of changes of interest. Still others may wish to receive a report at their explicit request only. Our QSS component ties together these diverse options in a general-purpose subscription framework.

Each QSS *subscription* consists of three main components. The first component, *the polling query*, is the query that is executed over the information source to gather data of interest. We use Tsimmis queries as polling queries. The second component, the *filter query*, is executed over

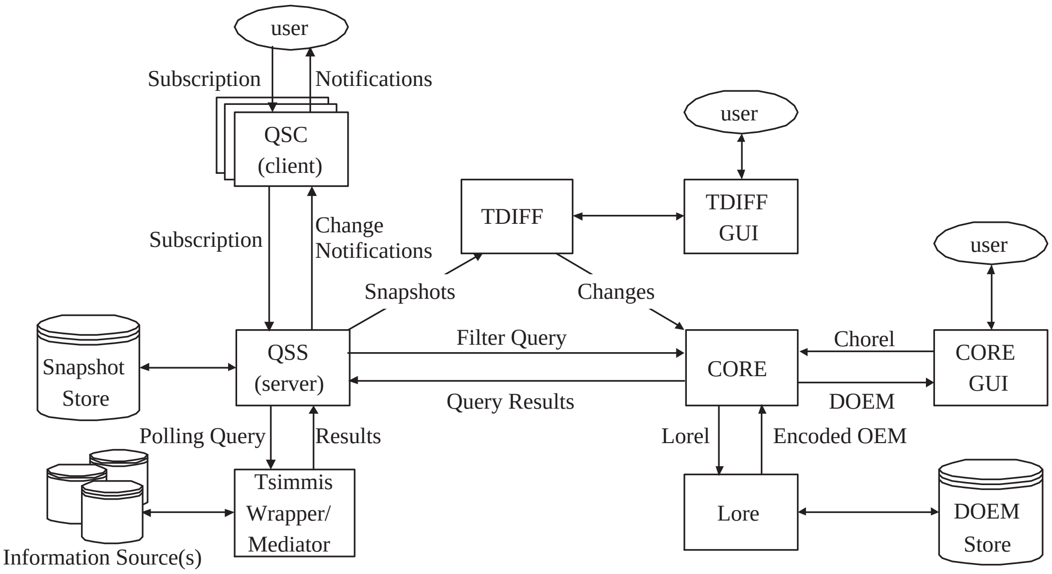

Figure 2: System Architecture

the history of the data gathered from the source and the results are returned to the subscriber. We use Chorel queries as the filter queries. The third component is a *frequency specification* that determines when and how the polling and filter queries are executed. For further details on QSS, please refer to [CAW98].

2.4 System Architecture

Figure 2 depicts the architecture of our change management system. The QSS component has a client-server architecture, with one or more client processes (*Query Subscription Clients* or QSCs) connected to the server process (QSS). Users interact with the QSC through a graphical user interface, creating subscriptions, issuing probes, and receiving results. The QSS component issues polling queries over information sources (via Tsimmis wrappers or mediators) and the result of each query is stored by QSS as the current snapshot for this query. This snapshot and the previous snapshot are sent to the TDIFF component, which identifies the changes to the data and stores them within CORE. QSS issues filter queries to the CORE database and the result of each query, if non-empty, is returned to the QSS component. QSS then sends the changes to the appropriate QSC clients, which notify their subscribers about the fresh results of their subscriptions. Users can view subscription results using the QSC user interface. In addition to the subscription service, our change management system offers users the ability to interact directly with TDIFF or with CORE, which is useful for independently exploiting their functionality.

3 Example Demonstration Walk-Through

Our change management system is fully implemented, and as of October 1997 our demonstration proceeds as follows. We will continue with our running example of a Web-based weather site that offers information such as temperature, precipitation, weather advisories, and road conditions for many cities. This site is updated frequently as weather conditions change.

The first part of our demonstration shows the detection of changes in the data source by the TDIFF component. For example, we can load two snapshots of the weather source into TDIFF and it will tell us, e.g., whether the snowfall for an area has increased or decreased and if so, what the old and new values are. Using the TDIFF graphical user interface, we can browse these changes and observe that the overnight snowfall of a city has gone from 3 inches to 0 inches, probably indicating the end of a storm. Further examples of TDIFF can be found at `http://www-db.stanford.edu/c3/`.

In the next part of our demo, we illustrate how CORE is used to browse and query the history of the weather site. In particular, we run TDIFF on a series of snapshots of the Web site and load the detected changes into a CORE database. The CORE user interface allows us to browse the complete history of the data, including updates that were made months ago. For example, we may observe that Tahoe City went through a major heat-wave last summer, with temperatures ranging from 90 to 100 degrees during August. In addition to browsing the data's history, we can also pose expressive Chorel queries over the data through the CORE user interface. For example, we can submit a query asking for "all California cities where the overnight snowfall has decreased by at least 6 inches" to determine if a storm is subsiding or not. This query would be expressed in Chorel as follows. (Like other database query languages, we expect Chorel queries to be issued primarily by client software interfacing with user-friendly GUI tools. For details of the query language, please refer to [CAW98].)

```
select C
from Weather.CA.city C
where exists S in C.snowfall<upd at T from OldVal to NewVal> :
     (OldVal - NewVal) > 6
```

We can then use the CORE interface to browse the results. The interface displays the cities and their subobjects (temperature, snowfall, etc.) in a graphical format, and we can expand the history of any edge or node in the display (e.g., the snowfall value for each city). This expansion shows all updates to this value, and we can determine if the decline in snowfall is a continuing trend.

The final part of the demonstration shows how the results of the above query can be monitored automatically using QSS. Avid skiers can create a subscription and be notified automatically anytime there has been a good snow-storm that seems to be clearing up, suggesting good skiing conditions. Using the QSS user interface, we create a subscription over the weather source. The polling query retrieves all information about California cities:

```
select C
from Weather.CA.city C
```

The filter query is the Chorel query shown earlier. We set the frequencies to poll and filter once a day. Whenever the snowfall of a city in California decreases by six inches, QSS will notify us. When we are notified of the update, we can log into the QSS interface and see the result of the filter query, which tells us which cities have had large snow-storms subside.

Our current prototype also supports a number of more sophisticated interactions not described here due to space constraints. Furthermore, we continue to work on our system, adding other useful features and supporting a greater variety of heterogeneous databases.

References

[Abi97] S. Abiteboul. Querying semistructured data. In *Proceedings of the International Conference on Database Theory*, Delphi, Greece, January 1997.

[AQM+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, November 1996.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montréal, Québec, June 1996.

[CAW98] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the International Conference on Data Engineering*, 1998. To appear. Available at `http://www-db.stanford.edu`.

[CGM97] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tuscon, Arizona, May 1997.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The Tsimmis project: Integration of heterogeneous information sources. In *Proceedings of 100th Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, October 1994.

[CRGMW96] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, Montréal, Québec, June 1996.

[HGMC+97] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Proceedings of the Workshop on Management of Semistructured Data*, pages 18–25, Tuscon, Arizona, May 1997. Available at `http://www-db.stanford.edu`.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 413–424, Bombay, India, September 1996.

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 161–186, Singapore, December 1995.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker: A mediation system based on declarative specifications. In *Proceedings of the International Conference on Data Engineering*, pages 132–141, New Orleans, February 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[SL90] A. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[TWU] The weather underground service on the world-wide web. Available at `http://www.wunderground.com/`.