ORCHESTRA: Facilitating Collaborative Data Sharing

Todd J. Green   Grigoris Karvounarakis   Nicholas E. Taylor

Olivier Biton   Zachary G. Ives   Val Tannen

Computer and Information Science Department
University of Pennsylvania
{tjgreen,gkarvoun,netaylor,biton,zives,val}@cis.upenn.edu

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*

General Terms

Management, Design, Experimentation

Keywords

Data exchange, data integration, data sharing, reconciliation, schema mappings

1. INTRODUCTION

One of the most elusive goals of structured data management has been sharing among large, heterogeneous populations: while data integration [4, 10] and exchange [3] are gradually being adopted by corporations or small confederations, little progress has been made in integrating broader communities. Yet the need for large-scale sharing of heterogeneous data is increasing: most of the sciences, particularly biology and astronomy, have become data-driven as they have attempted to tackle larger questions. The field of bioinformatics, in particular, has seen a plethora of different databases emerge: each is focused on a related but subtly different collection of organisms (e.g., CryptoDB, TIGR, FlyNome), genes (GenBank, GeneDB), proteins (UniProt, RCSB Protein Databank), diseases (OMIM, GeneDis), and so on. Such communities have a pressing need to interlink their heterogeneous databases in order to facilitate scientific discovery.

Schemes for data sharing at scale have generally failed in the past because database approaches tend to impose strict global constraints: a single global schema, a (perhaps virtual) globally consistent data instance, and central administration. Each of these requirements is a barrier to participation: global schema design across a community is arduous and often requires many revisions; global consistency restricts a participant from disagreeing with others (if enforced), or may result in inconsistent answers (if unenforced); central administration impedes responsiveness to evolving requirements. Even the new approach of peer data management [9, 7], which supports multiple mediated schemas and thus distributes some aspects of administration and eliminates the need for global schema design, still limits local autonomy because of strong data consistency requirements. To sidestep these limitations, data providers typically resort to custom, ad hoc tools: scientific data sharing often consists of large databases placed on FTP sites, which users download and convert into their local format using custom Perl scripts. Meanwhile the original data sources continue to be edited. In some cases the data providers publish weekly or monthly lists of updates to help others keep in sync; however, few sites, except direct replicas, actually exploit these update lists — instead, different copies of the data are simply allowed to diverge.

Our research goal is to provide a more principled and general-purpose infrastructure for data sharing with significant gains in terms of freshness, flexibility, functionality, and extensibility. Largely guided by the needs of biologists and other scientific users, but with a goal of addressing large-scale data sharing in the broader context, we define a model for a declarative, yet extremely flexible, approach to data sharing, called the *collaborative data sharing system*, or CDSS.

2. COLLABORATIVE DATA SHARING

The CDSS model dramatically reduces the barriers to sharing by allowing *loosely* coupled confederations of sites, each of which maintains a local schema and a fully autonomous, editable local data instance. Sites exchange data on an as-desired basis: the CDSS uses declarative *schema mappings* that specify a local database's relationship to other sites, as well as policies about what data the site *trusts* (based on its origins and value). The CDSS arbitrates conflicts in a custom way for each participant, based on whom and what it trusts: this allows for "selective disagreement" and is thus significantly different from prior work with sharing in mind, such as distributed data integration/exchange and groupware, in that it allows the end user complete control over the contents of the local data instance.

The different goals of the CDSS model affect even its basic unit of information transfer. A database is often the storage system for information about high-level real-world entities (such as genes, customers or quasars); a single entity may in turn be represented logically as a collection of a number of tuples in different relations. Transactional atomicity guarantees that the information about a particular real-world entity is internally consistent by ensuring that a set of updates are applied together (or else none are applied). To enforce transactional atomicity between different participants, the CDSS considers *transactions* as the basic unit of operation, and it propagates, translates, and considers

SIGMOD'07, June 11–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

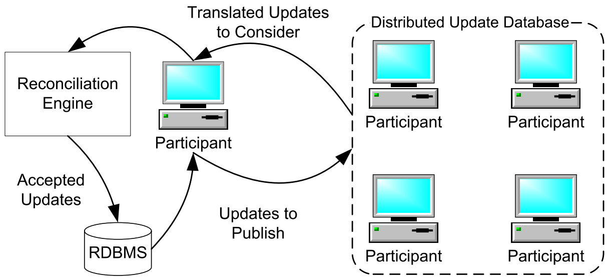


Figure 1: An overview of the architecture of a CDSS. Here the published transactions are stored in a peer-to-peer distributed database, though one can also use other methods to store the published updates.

conflicts among such units. This is in constrast to previous models of data integration and data exchange which ignore transactions. Furthermore, we observe that data dependencies between operations in different transactions (e.g., one transaction modifies a tuple inserted by another *antecedent* transaction) induce a dependency graph on the transactions themselves that must be respected when considering which transactions to accept or reject.

The CDSS consists of a network of collaborators (*participants* or *peers* at independent sites), each of which has a local database instance and may be intermittently connected. Each site spends the majority of its time operating in a *locally autonomous mode*, with users posing queries and making modifications directly over a local database instance. Upon an administrator's request, the CDSS performs an *update exchange*; this allows data to flow between participants in the system.

The two basic operations of update exchange are *publication* and *reconciliation*. When a participant publishes its new transactions, the CDSS archives them (which is needed in case participants are only intermittently connected) and makes them available to the other participants in the system. When a peer reconciles, the CDSS translates newly published transactions into that peer's schema, and then chooses a consistent subset of the translated *candidate transactions* to apply to its local instance, based on a set of user preferences. Figure 1 shows how these steps fit into the architecture of a CDSS. We describe our implementation of these steps for the Orchestra CDSS in section 3.

Each update exchange operation advances a logical clock: the overall state of data in the system has changed, and any future updates should be causally related to the previously accepted ones. The result of the publish-translate-reconcile sequence is a new data instance for the requesting peer. In effect, a public snapshot of this instance, is made visible to all other participants, while the local version can continue to be edited in a way that is only visible to the users at that site.

We observe that three aspects of the CDSS model distinguish it from past work. First, *any* participant may make updates, including deletions, and this changes how data must be mapped and propagated in a peer-to-peer environment. Second, each participant can ignore or even *override* updates it gets from elsewhere, using its own local updates. Finally, the CDSS must translate tuple-level updates while (1) keeping track of their associated transactions, for purposes of conflict detection and resolution, and (2) tracing their provenance, for purposes of trust assignment.

3. THE ORCHESTRA CDSS

The Orchestra CDSS [8, 11, 5] has been in development at the University of Pennsylvania for more than two years. It supports all of the features of the CDSS model described in section 2, and has been tested extensively on small- to medium-sized networks with update-heavy workloads. In this section we discuss the two key challenges in implementing a CDSS, translation of updates and efficient reconciliation.

Since the CDSS model relies on propagation of *updates* rather than data through the system, there must be a method to translate updates over one schema to updates over a different schema. Rules for translating updates (and determining the transactions to which they belong) can be derived from the mappings between the different schemas, though they can become somewhat complicated if the mappings involve multiple joins. The rules must also maintain enough *provenance* or *lineage* [1, 2, 12] information that (1) reconciliation can choose between transactions based on user preferences, and (2) efficient incremental recomputation of the target data instance and provenance is possible. Our work in [5, 6] has developed a new formulation of provenance to meet these needs and efficient algorithms to maintain incrementally both this provenance information and the underlying data.

The result of update translation is a set of candidate transactions that (1) may be mutually incompatible, (2) may not be applicable to the local database instance due to rejected or missing antecedent transactions, or (3) may not be trusted by the local site. The reconciliation algorithm of [11] combines candidate transactions with the antecedents transactions needed to apply them, in order to produce *applicable transaction groups*. If it finds that the candidate transaction depends on an antecedent transaction that has already been rejected, that candidate transaction must be rejected as well. Otherwise, it uses user preferences, encoded as *trust conditions* to associate numerical priorities with applicable transaction groups. These trust conditions are based on predicates over the contents and provenance of updates: in many cases, a site will assign a value judgment to a modification based on where it originated or how it was assembled. Based on these priorities, it uses a greedy algorithm to choose the highest-priority mutually consistent set of transactions to apply; if several inconsistent transactions of the same priority conflict, they must be *deferred* until a decision about them is reached by the site administrator. Transactions that modify data from previously deferred transactions must also be deferred. At any later point in time, the site administrator can manually resolve the conflict between deferred transactions by choosing which one to apply. After this is done, all deferred transactions that transitively depend on the chosen transaction are applied, and all those that depend on the rejected one are themselves rejected.

4. DEMONSTRATION

In the demonstration, we will show each step of the collaborative data sharing process using a bioinformatics schema that has been simplified to highlight key aspects of our system. Figure 2 shows the CDSS we will use. In this CDSS, four participants (the Universities of Alaska, Beijing, Crete, and Dresden) share information about reference sequences for various proteins in several organisms. Alaska and Bei-

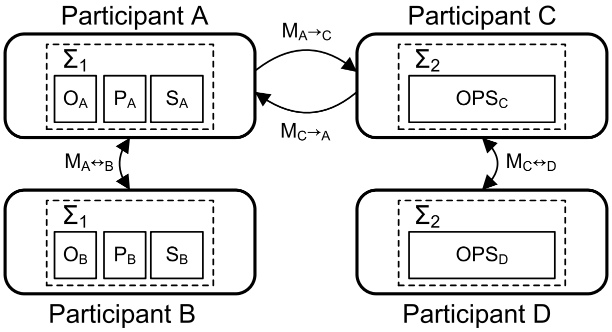


Figure 2: A CDSS for four bioinformatics sources. Participants A and B share a common schema, as do participants C and D, so mappings $M_{A\leftrightarrow B}$ and $M_{C\leftrightarrow D}$ are identity mappings. The mapping $M_{A\rightarrow C}$ translates the three tables of schema Σ_1 into the single table of Σ_2, and $M_{C\rightarrow A}$ does the inverse.

jing assign a unique ID to each organism and protein and use those to give the reference sequences, giving a schema $\Sigma_1 = \{O(\underline{org}, oid), P(\underline{prot}, pid), S(\underline{oid}, \underline{pid}, seq)\}$, while Crete and Dresden do not assign IDs, giving a second schema $\Sigma_2 = \{OPS(\underline{org}, \underline{prot}, seq)\}$. Mappings $M_{A\leftrightarrow B}$ and $M_{C\leftrightarrow D}$ are identity mappings. $M_{A\rightarrow C}$ joins the three tables of Σ_1 into the single table of Σ_2, while $M_{C\rightarrow A}$ does the inverse and splits the single table of Σ_2 into the three tables of Σ_1. Alaska, Beijing and Dresden each trust all other participants equally, but Crete trusts only Beijing and Dresden (but prefers Beijing to Dresden in the event of a conflict).

Using a Java-based GUI (shown in Figure 3), we will show the current state of each peer, the mappings between peers, and the updates (original and translated) that are applied while reconciling. The user will be able to perform updates to the local instance at each peer, to reconcile, and to resolve conflicts manually. The demonstration will show the following cases:

- Updates made by Alaska get translated into Dresden's schema and applied, and vice versa.

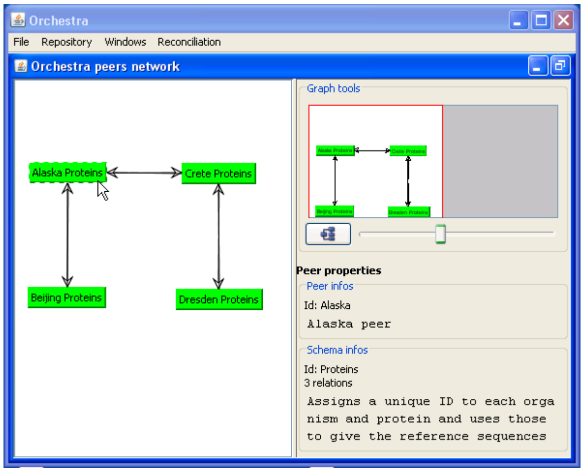
- Beijing and Dresden publish conflicting updates, and Crete therefore rejects Dresden's. Dresden then publishes more updates which depend on its earlier ones, which Crete must also reject.

- Alaska publishes an insertion of several data points in the same transaction. Beijing publishes a modification of one of them. Crete then reconciles, and ends up accepting both the transaction from Beijing and the antecedent from Alaska, even though Crete does not trust Alaska.

- Beijing and Alaska publish conflicting updates. Dresden reconciles and defers both of them, since user intervention is needed to determine which to accept. Crete reconciles and publishes a modification of Beijing's update. Dresden reconciles again and defers Crete's update. Dresden then resolves the conflict in favor of Dresden, and accepts Crete's transaction automatically.

- Beijing publishes a number of updates and then goes offline. Alaska can reconcile and still retrieve Beijing's updates from the CDSS.

These scenarios will allow the user to see the Orchestra system in action, and to understand how it solves some of the many interesting problems that can arise in the CDSS model. We will also discuss how Orchestra is being used as the core engine in the development of global-scale bioinformatics data sharing systems such as SHARQ[1] and pPOD[2].



Id: Alaska
Alaska peer

Id: Proteins
3 relations
Assigns a unique ID to each orga
nism and protein and uses those
to give the reference sequences

Figure 3: The mapping viewer of the Java-based GUI.

5. ACKNOWLEDGMENTS

This work has been funded in part by NSF grants IIS-0477972, 0513778, and 0415810, and DARPA grant HR0011-06-1-0016. The authors would like to thank Sarah Cohen-Boulakia and the members of the SHARQ project for assistance with the biological datasets, and the members of the Penn Database Group and the anonymous reviewers for their feedback and suggestions.

6. REFERENCES

[1] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, 2001.

[2] Y. Cui. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, 2001.

[3] R. Fagin, P. Kolaitis, R. J. Miller, , and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336, 2005.

[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2), March 1997.

[5] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. Submitted for publication, 2007.

[6] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[7] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, March 2003.

[8] Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *CIDR*, January 2005.

[9] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD*, June 2003.

[10] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, 1996.

[11] N. E. Taylor and Z. G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *SIGMOD*, 2006.

[12] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.

[1] http://db.cis.upenn.edu/research/SHARQ.html
[2] http://phylodata.org