Galaxy: Encouraging Data Sharing Among Sources with Schema Variants

Len Seligman, Peter Mork, Michael Morse, Arnon Rosenthal, Chris Wolf, Jeff Hoyt

The MITRE Corporation
7515 Colshire Drive
McLean, VA 22102
{seligman, pmork, mdmorse, arnie, cwolf, jchoyt}@mitre.org

ABSTRACT

This demonstration presents Galaxy, a schema manager that facilitates easy and correct data sharing among autonomous but related, evolving data sources. Galaxy reduces heterogeneity by helping database developers identify, reuse, customize, and advertise related schema components. The central idea is that as schemata are customized, Galaxy maintains a derivation graph, and exploits it for data exchange, discovery, and multi-database query over the "galaxy" of related data sources. Using a set of schemata from the biomedical domain, we demonstrate how Galaxy facilitates schema and data sharing.

1. INTRODUCTION

Communities often emerge in which participants collect and manage their own data using many variants of the same schema. For example, a health care economist may make data available in a spreadsheet or Access database (Figure 1). Others then adopt and customize the schema and populate it with their own data. These customizations may in turn be further customized by others.

To improve data sharing, many communities agree on a *core schema* but also allow individual systems and subcommunities to add their own information, for example using an XML wildcard (e.g., *xs:any*) or by adding a column to a spreadsheet template received downloaded from a website. The phenomenon exists at large scale, too: one schema intended for Air Force operations was copied and adapted to create databases for Army and Navy users with somewhat different needs The National Information Exchange Model (niem.gov) originating in the Department of Justice is another example of large-scale schema reuse. This is a widespread design pattern in practice, that we call *core and corona*, where a *corona* is an extension of the core schema that adds new entity types and/or properties of interest to a narrower community. The core schema provides a base level of interoperability across many systems, while participants still have the flexibility to respond rapidly to local data needs by creating coronae as needed.

There are huge opportunities to increase the interoperability impact of such data standards by explicitly supporting the fractal nature of sharing communities—many subcommunities' coronae in turn serve as a testing ground for a future core or as some narrower community's core. Prior tools do not exploit this observation and therefore miss many opportunities to share specifications and thereby reduce data heterogeneity.

This demonstration presents Galaxy, a community schema manager that facilitates easy and correct data sharing among autonomous, but related, evolving data sources. It accomplishes this by:

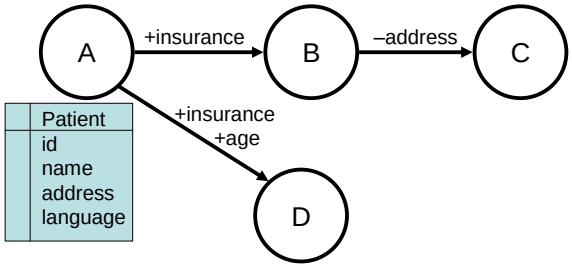

Figure 1: Sample sharing scenario in which participant A forwards a spreadsheet (or schema) to two colleagues (B and D) each of whom customize the spreadsheet. B then forwards his revised spreadsheet to C who makes further revisions.

Proceedings of the 34th VLDB Conference,
Auckland, New Zealand, 2008

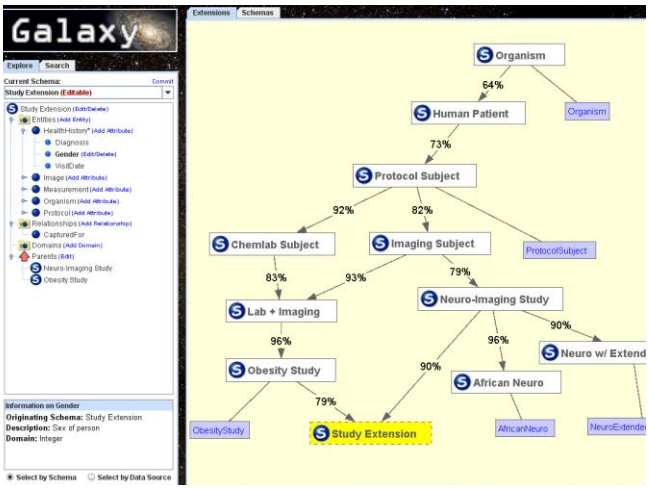

Figure 2: Galaxy User Interface, showing a navigation interface for extensions of a particular schema, with instances shown in blue. Once an appropriate schema is found, this schema can be extended or modified graphically.

- explicitly managing the derivations among families of related schemata,
- reducing heterogeneity by helping database developers identify, reuse, customize, and advertise related schema components, and
- using the knowledge of inter-schema relationships to support multi-database query and data exchange over all related data sources that can answer the query.

Of course, one could discover correspondences across coronae post facto; however even with tools, this remains expensive and error prone. We propose a complementary strategy that, in many cases, obviates the need for post facto data integration. Instead, we provide tools that encourage *and track* the reuse of schema specifications. Galaxy is not intended as a component for a schema matcher—it is a separate system to reduce the need for schema matching

In this demonstration proposal, we first describe the Galaxy models for tracking derivations and querying across schemata. We then provide details of the conference demonstration. Finally, we describe related work.

2. GALAXY REPOSITORY MODEL

Galaxy's repository tracks derivations and deviations. It stores 1) a directed acyclic graph in which nodes are schemata and arcs indicate derivation (described below), 2) a 1-to-many relationship between a schema and the data sources that instantiate the schema and 3) metadata about data sources (e.g., how they can be queried). Figure 2 illustrates a derivation graph as shown by the Galaxy user interface. A rectangle with a blue S is a schema, an arrow indicates derivation, and a blue rectangle represents a data source that instantiates the schema to which it is linked.

A schema consists of two kinds of constructs: structural constructs (attributes, relationships, and domain values in our demo) and constraints (e.g., primary and foreign keys). A *component* consists of a structural construct and its associated constraints. Structural constructs have an associated semantics that may be represented as text or more formally as a reference to an element of an ontology.

A coronal schema is obtained by *importing* and *editing*. Each imported component retains its semantics and identity (e.g., using a URI). By maintaining identity, Galaxy avoids the need for post facto integration. A component can subsequently be edited by adding or removing additional substructures (such as attributes) or by adding constraints. (When importing components rather than an entire schema, the system ensures that the result is a well-formed schema.) As shown in Figure 2, a corona can extend multiple cores, thereby establishing a derivation DAG.

To illustrate, we can view Figure 1 as a schema derivation graph. Schema *B* is derived from *A* and imports the Patient entity with attributes id, name, address and language along with their semantics and any associated constraints. *B* customizes *A* by also adding the attribute insurance. *C* imports everything in *B*, except that it removes address. Note that while both *B* and *D* add insurance, they cannot be assumed to be the same concept, since they were added independently by autonomous participants. For example, *D*.insurance could have a {Yes, No} domain for dental care, while *B*.insurance could be health {Aetna, BlueCross, …}.

The Galaxy model allows one to associate additional constraints with data sources. For example, one might have a schema PatientInfo with attributes Person.record# and Person.zip. PatientInfo may contain a constraint that Person.record# is not null. An associated data source VirginiaPatients may have an additional constraint that Person.zip must be a valid Virginia zipcode.

3. QUERY MODEL & IMPLEMENTATION

We first describe how queries are posed, and what they mean. We then describe how queries are forwarded to relevant data sources, and finally, describe the handling of overloaded domain values.

The Galaxy query model allows users to choose a schema against which to pose their queries. The instance set associated with a schema consists of all instances compatible with that schema, from any data source. Optionally, a user can manually select a subset of the data sources (in practice, the user will usually select all available data sources or restrict the query to local data only).

Thus, given a specific query, the next task is to determine which data sources could sensibly answer that query. Towards this end, we partition the schema constructs mentioned in a query into those that are optional and those that are mandatory. For example, to satisfy an equality predicate, the corresponding attribute is mandatory, but attributes

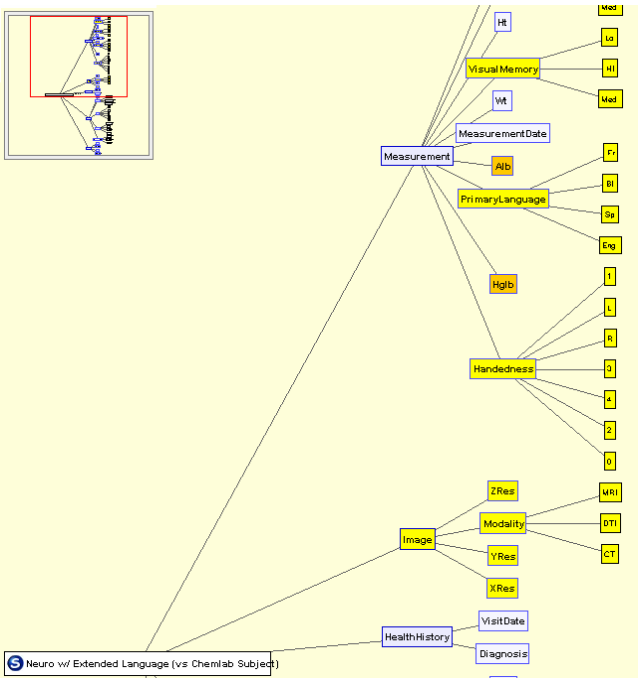

Figure 3: The differences between members of a schema family can be quickly identified through a graphical difference display. Elements common to two schemata are shown in gray while elements existing in only one or the other are in orange or yellow.

appearing only in a SELECT clause are generally optional (i.e., NULL values are acceptable).

Let *MC* be the set of mandatory constructs mentioned by query *Q*. Let *SC* be the set of schema constructs appearing in schema *S*. It is sensible to forward *Q* to instances of *S* if $MC \subseteq SC$.

Consider Figure 1 and a query for which address is a mandatory construct. It is sensible to forward that query to any instances of schemata *A*, *B*, and *D*. It is not sensible to forward the query to instances of schema *C* because address has been removed by *C*.

An added complication arises when a domain value appears in a negated context (e.g., language ≠ English). In this case, this query can sensibly be forwarded to any data source. However, some databases may contain domain values that do not make sense from the perspective of the local schema. As a result, when there are enumerated domain values, we convert negative predicates that mention specific domain values into equivalent positive predicates, such as language ∈ {Spanish, German, …}.

This complication is particularly important when the same label is introduced by multiple schemata. For example, in the domain for language, *B* defines a new domain value with the label 'BI' to mean "bilingual," while *D* introduces a different domain value (with the same label) that means "the language of Burundi." Whereas the two senses of the string 'BI' are indistinguishable from the perspective of the underlying relational databases and many data integration systems, the Galaxy middleware correctly preserves the distinction.

4. DEMONSTRATION

We will demonstrate Galaxy features that simplify the creation and querying of schema families, illustrating a typical schema extension life-cycle. We will show how database designers and end users benefit from Galaxy through four main processes: *discovering* potentially relevant schemata, *navigating* the derivation graph to identify a schema to reuse, *extending* that schema to accommodate application specifics (and documenting the result), and finally *querying* instances of that schema for data. Details of each process follow:

Discovery: To identify a core set of schema elements to extend to create a new source, a designer enters keywords pertinent to their application. Galaxy returns schemata and extensions to these schemata which match the entered text, including annotations about schema entries and attributes to assist users.

Navigation: Once one or more schema families of interest for an application have been identified, the designer can explore the schema graph to identify the particular schema that is most appropriate to serve as the core for their specific application, as shown in Figure 2. Galaxy includes mechanisms to easily identify ancestor/descendant relationships among schemata and to navigate the graph. Parents and their immediate children in the schema graph have a percentage of similarity shown. Evaluation of schema similarity is further enhanced through a graphical schema *diff*, shown in Figure 3.

Extension: A designer creates a new schema by importing the entities and attributes of one or more existing schemata in a family. This new schema is now a descendant of each of the schemata whose entities and attributes it imported. The new schema can be customized by adding or subtracting elements in a graphical environment provided by Galaxy. Galaxy understands where two schemata employ the same component.

Query: A schema, as populated by a selected set of sources, can be queried, as shown in Figure 4. The query system presents an interface from which any schema can be queried (in the future we plan to also support the automatic querying over multiple schemata). An inference engine identifies those data sources to which a query can sensibly be forwarded. These databases receive the query and forward all results back to the query system, which combines the results before presenting them to the user.

The demonstration uses a family of biomedical schemata, whose core includes patient identifying elements. We derive coronal schemata by importing the core and adding elements necessary for different types of patient studies. For example, to create a derived schema for a neuro-imaging

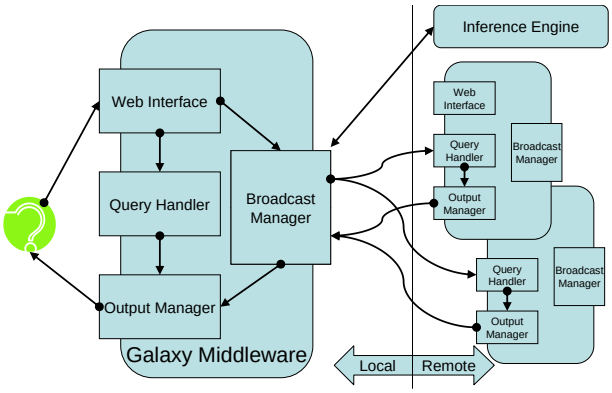

Figure 4: Galaxy middleware allows users to query their favorite local schema, but receive results from all or selected remote sources whose schemata contain the components necessary to answer the query.

study, the designer adds schema elements pertaining to image modalities. Numerous health and medical study schemata extend from the core schema, and many of these schemata have data sources associated with them. We then illustrate end users performing a variety of queries, over a variety of schemata. Galaxy's distributed query facilities bind queries to appropriate sources and combine the results.

5. RELATED WORK

Numerous schema matching methods exist [1-3]. In contrast, we reduce the need for post-facto matching; importing creates schemas with known matches. Model management [4] provides a rich, general purpose, very complex framework [5]; we show the value, for non-IT specialists, of a small set of schema extension and edit operations. Galaxy introduces a simple extension operator for which the mapping across versions is explicit.

Galaxy is related to schema versioning [6] and evolution [7]: whereas in Galaxy each schema is immutable, each derived schema can be viewed as a new version of its parent. In addition, prior schema versioning research aims to ease migration of instance-sets to more current versions. In contrast, our work versions only schemas, using them to support data sharing over autonomous but related sources. Also, in contrast to prior work, we provide explicit support for discovery, reuse, and extension of existing schema components.

Google Base [8] offers flexible sharing and query over single entity types with an extensible, un-typed set of properties. It provides very forgiving, simple structured query over data that might otherwise support nothing better than text search. In contrast, Galaxy supports the needs of communities that have somewhat greater semantic cohesiveness and need more powerful query, while still allowing participants to extend existing schemata to meet their unique requirements.

6. SUMMARY

Galaxy demonstrates reuse and customization of components in a family of related schemata. It uses knowledge of schema variants to provide easy and correct data sharing among autonomous but related, evolving data sources. We have demonstrated Galaxy to several customers; their feedback convinces us that it addresses a real and important problem.

7. REFERENCES

[1] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++," SIGMOD, Baltimore, MD, 2005.

[2] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos, "iMAP: Discovering Complex Mappings between Database Schemas," SIGMOD, Paris, France, 2004.

[3] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *VLDBJ*, vol. 10, pp. 334–350, 2001.

[4] P. A. Bernstein, "Applying Model Management to Classical Meta Data Problems," CIDR, Asilomar, CA, 2003.

[5] S. Melnik, E. Rahm, and P. A. Bernstein, "Rondo: A Programming Platform for Generic Model Management," SIGMOD, San Diego, CA, 2003.

[6] J. F. Roddick, "A Survey of Schema Versioning Issues for Database Systems," *Information and Software Technology*, vol. 37, pp. 383–393, 1995.

[7] E. Franconi, F. Grandi, and F. Mandreoli, "A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases," Computational Logic, London, UK, 2000.

[8] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu, "Web-Scale Data Integration: You can afford to Pay as You Go," CIDR, Asilomar, CA, 2007.