Galaxy: Encouraging Data Sharing Among Sources with Schema Variants

Len Seligman, Peter Mork, Michael Morse, Arnon Rosenthal, Chris Wolf, Jeff Hoyt
The MITRE Corporation
{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 matching, 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 biomedical researcher may make experimental 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 sub-communities to add their own information, for example using an XML wildcard (e.g., *xs:any*). This is a widespread design pattern in practice,[1] 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 sub-communities' coronae in turn serve 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:

- 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: provide tools that encourage reuse of specifications, and then use schema match and mapping tools (e.g., [1]) to resolve remaining heterogeneity.

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 consists of 1) a directed acyclic graph in which nodes are schemata and arcs indicate derivation (described below), 2) metadata on data sources (e.g., how they can be queried), and 3) a 1-to-n relationship between a schema and the data sources that use the schema. 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 uses the schema to which it is linked.

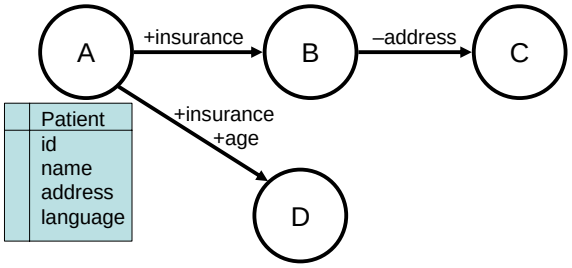

Figure 1: Sample sharing scenario in which participant A forwards a spreadsheet (or schema) to two colleagues (B and D) each of whom customize the spreadsheet. B then forwards his revised spreadsheet to C who makes further revisions.

[1] E.g., the National Information Exchange Model (www.niem.gov)

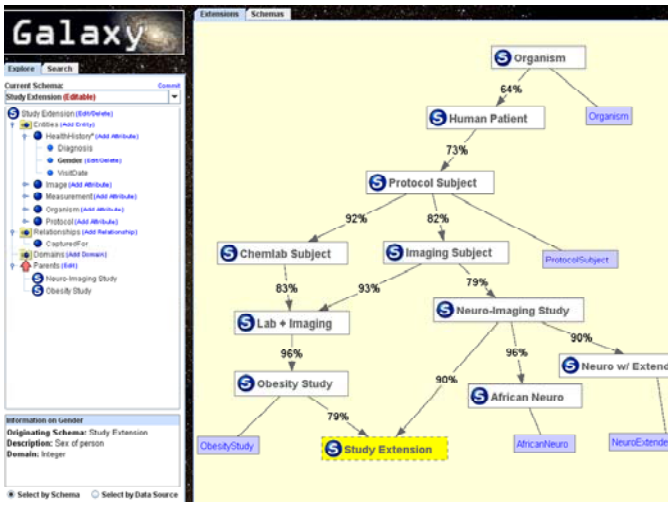

Figure 2: Galaxy User Interface, showing a navigation interface for extensions of a particular schema, with instances shown in blue. Once an appropriate schema is found, this schema can be extended or modified graphically.

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

When schema S_2 is derived from S_1, this means that S_2 imports both the constructs (*including their semantics*) and the constraints from S_1. The imported component is exactly the same in S_2 as it is in S_1. (i.e., each component could be identified using a URI.) In addition to importing components, a corona can customize the schema either by adding or removing components.

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, B.insurance could have a domain of {Yes, No} for autos, while D.insurance could be {Aetna, BlueCross, …} for health.

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

3. GALAXY QUERY MODEL

Based on the Galaxy repository model above, we now describe the Galaxy query model. We first describe how queries are posed, then describe how queries are forwarded to relevant data sources, and finally, describe the handling of overloaded domain values.

Recall that each schema in the derivation graph contains a set of schema components and each such component is uniquely defined. We define the schema universe to be the set of all schema components defined by any participant. A query in the galaxy model is posed using the constructs appearing in the schema universe. Thus, a query can (in theory) draw from a potentially large set of names. In practice (and in our demo), queries are posed using the constructs appearing in a particular schema, but, by definition, such queries are valid with respect to the universe.

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 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 database. 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 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, the Galaxy middleware correctly preserves the distinction.

4. DEMONSTRATION

The Galaxy system contains a number of features that enable ease of extension and querying for schema families. We will demonstrate these features by illustrating a typical schema extension life-cycle. Database architects use Galaxy to facilitate schema extension through four main steps: searching for potentially relevant schemata, navigating the derivation graph to identify a schema to reuse, extending that schema to accommodate application specifics, and finally querying instances of that schema for data. Details of each process follow:

Search: To identify a core set of schema elements to extend, users enter 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 user 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

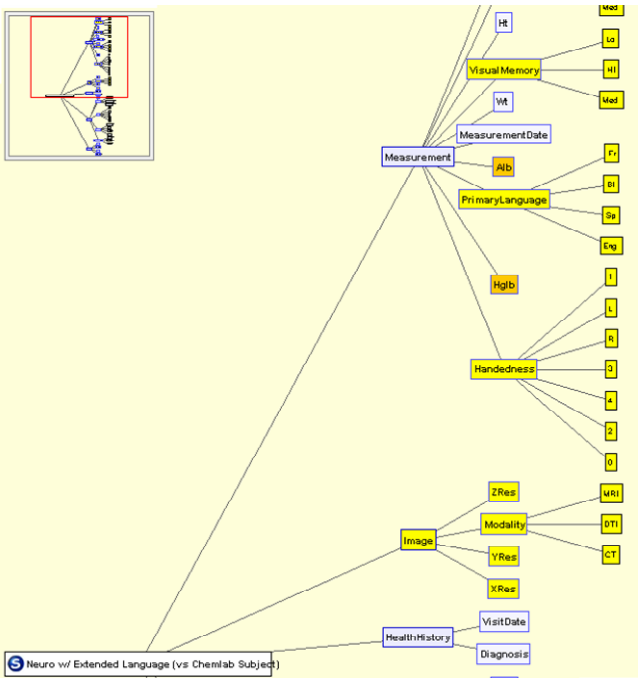

Figure 3: The differences between members of a schema family can be quickly identified through a graphical difference display. Elements common to two schemata are shown in gray while elements existing in only one or the other are in orange or yellow.

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 new schema can be created 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.

Query: Instances of a schema can be queried through middleware, the architecture for which is shown in Figure 4. This middleware allows users to graphically generate queries against a particular schema. An inference engine identifies data sources to which a query can sensibly be forwarded. These instances receive the query and forward all results back to the original database, which combines the results before presenting them to the user.

During the demonstration, we will use a family of schemata taken from a biomedical application domain whose core includes patient identifying elements. Derived schemata from this core add elements necessary for different types of patient studies. For example, one derived schema is for a neuro-imaging study and adds schema elements pertaining to image modalities to the schema core. Numerous health and medical study schemata extend from the core schema, and many of these schemata have data sources associated with them, which will be used to illustrate Galaxy's distributed query facilities.

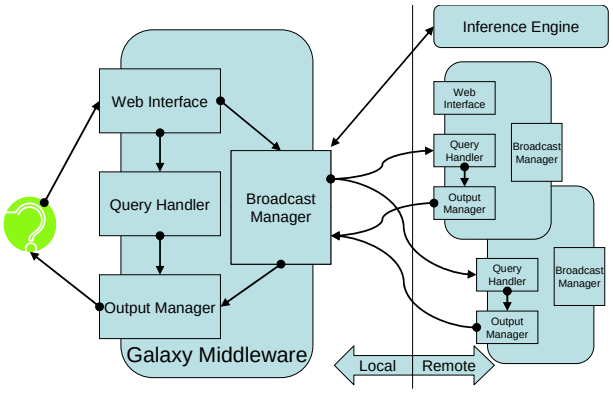

Figure 4: Galaxy middleware allows users to query their favorite local schema, but receive results from all or selected remote sources whose schemata contain the components necessary to answer the query.

5. RELATED WORK

Numerous schema matching methods exist [6]; they are applied to existing schemas, while we create schemas with known matches, via reuse. Model management [2] provides a rich, general purpose, very complex framework [4]; 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 [5] and evolution [3]: whereas in Galaxy each schema is immutable, each derived schema can be viewed as a new version of its parent.

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] Aumueller, D., Do, H., and Rahm, E. Schema and Ontology Matching with COMA++. *SIGMOD*, 2005.

[2] Bernstein, P. Applying Model Management to Classical Meta Data Problems. *CIDR*, 2003.

[3] Franconi, F., Grandi, F., and Mandreoli, F. A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases. *Computational Logic*, 2000.

[4] Melnik, S., Rahm, E, and Bernstein, P. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD,* 2003.

[5] Roddick, J. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology.*, 37(7), 1995.

[6] Rahm, E., and Bernstein, P. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*. 10(4) 2001.