A Rule Driven Bi-Directional Translation System for Remapping Queries and Result Sets Between a Mediated Schema and Heterogeneous Data Sources

R. Shaker[1], P. Mork, M.S.[2], M. Barclay[1], P. Tarczy-Hornoch, M.D.[1,3]

[1]Pediatrics, [2]Computer Science & Engineering and [3]Biomedical & Health Informatics
University of Washington, Seattle, WA

As the number of online biomedical data sources increases, so too do the number of ways to access such data. The research described herein focuses on creating a data access system that provides bi-directional translation and mapping of data between heterogeneous databases and a mediated schema. Semantic mapping rules stored in a knowledge base are used by our generalized software to convert XML query results obtained from each data source to a common schema representing a single ontology. We apply this approach to the domain of online genetic databases, demonstrating the system's scalability and integratability.

INTRODUCTION

There is currently no universally adopted standard for representing, storing and accessing the growing repository of public biomedical information. Though sequence, structure and function databases are often readily accessible via the Internet[1], the investment of time and expertise required to locate, aggregate and search these data sources is increasing. A common language for querying the contents of heterogeneous biomedical databases is greatly needed.

BACKGROUND

By developing a simplified ontology to describe the subset of data we wish to examine, and by creating a mediated schema based on that ontology that can act as our guide for posing queries against the realm of interest, we will demonstrate that distinct and separate data sources can be accessed using a single homogeneous view. Previous work by our group has centered on using the mediated schema and Tukwila engine to formulate query plans for accessing a heterogeneous set of online genetic databases[2].

The goal of this paper is to present a back-end data access system that complements that work and provides a single point of entry for answering queries against a sizable body of distinct but related data. It is our goal to provide a homogeneous view and allow for the querying of heterogeneous data sources, while filtering out the irrelevant data with which it may be interwoven. The solutions proposed here are not specific to the medical or genetic database community, but can be generalized to any set of online and queryable information for which a common ontology can be constructed.

Our system offers benefits over existing genetic data integration techniques. For example, local warehousing of data sources is not a requirement. Unlike solutions such as BioKleisli[3], our system provides a simple language for querying and combining data sources. Instead of relying on complex queries and the construction of virtual views, our data access system models only the shared entities from each source. By doing so, we present to the end user a simplified schema encompassing only the ontology in which they're interested. The mediated schema, rather than a view, dictates what data is made accessible to the query.

In contrast to systems such as PharmGKB[4], we use our mediated schema to perform queries across distributed databases while their approach pulls data into a central repository. Our model also provides a generalized interface system to diverse (not just relational) sources and can access sources even if underlying tables are not directly accessible.

REQUIREMENTS & APPROACH

Before beginning development of our data access system, we established a basic set of requirements. The system must be integratable, maintainable, extendable, scalable and efficient. Ease of integration makes the system accessible to other front-end query tools. To facilitate maintainability, we rely on external configuration parameters thus decreasing the required amount of skilled programmer support. Extensibility is important for the addition and support of additional data sources. Scalability and efficiency make this more than a demonstration project, and facilitate its eventual use and reliance in a production environment.

Integratability: To simplify integration of this system with other applications, we adopted a simple and generalized API. By limiting the input query parameters to a single URL, interfacing with the back-end engine is relatively simple. By returning result sets in the form of a valid XML[5] document, the process of describing and parsing expected output is straightforward. As a whole, the simplified API and development tools chosen for this system facilitate both language and platform independence.

Maintainability: To make the system more maintainable, we constructed generalized and modular solutions wherever possible. We use a two tier back-end: 1) data acquisition and 2) data translation. The data acquisition tier pulls data from remote sources and transforms it into a common, XML-based syntax while preserving original semantics. The data translation tier ("metawrapper") performs the semantic transformation. By creating this division in processing, we find that it is possible to construct a single and re-usable application to perform all data translation tasks.

The metawrapper performs a semantic transformation of each data source from its heterogeneous schema to the mediated schema. This conversion to a common schema allows query engines such as Tukwila to perform complex tasks such as joining result sets across multiple heterogeneous sources. Transformation between data source space and mediated schema space is driven by a set of semantic mapping rules. These rules are stored externally to the metawrapper application in a Protégé[6] knowledge base. This storage of mapping rules means that no code changes to the metawrapper are required when an ontology and its mediated schema change, or when the output from a remote data source changes.

By separating the data acquisition ("wrapper") component from the rest of the system, we facilitate adding, removing and modifying the applications that provide physical access to the individual data sources. This set of applications performs a simple, syntactic translation of source data into a common XML intermediate. This intermediate format is what is fed back to the translation layer. When the data source output formats change, wrappers must be updated to accept the new input formats. See Figure 1 for a depiction of the interactions between the query formulator, the metawrapper and the wrappers.

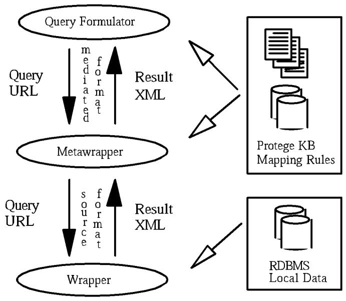


Figure 1: Translation Process

Extensibility: The two-tier model also facilitates extensibility by allowing wrappers to be written and unit tested before integration with the rest of the system. The use of a simplified API between wrappers and metawrapper enables programmers to use any language suitable for wrapper construction. In essence, any Web-based application that can be called via a URL and returns valid XML output can be used as a wrapper.

Scalability: All components are essentially standalone, with only the metawrapper requiring access to externally stored mapping rules. Multiple instances of each component may be deployed to multiple servers. Use of a Web interface provides the potential for future load balancing.

Efficiency: Because the data access engine is intended for use by real-time query engines, response time is of major importance. We address the engine's speed by internally and externally parallelizing as much of the process as possible. Internally, each wrapper is designed to return intermediate query results as soon as they are available. Externally, the metawrapper is designed to begin processing wrapper output before receiving the entire data stream.

In addition to the points already discussed, we weighed the relative advantages of local versus remote data storage. Our data access system is designed with the explicit goal of integrating remote data sources into a single view; however, there sometimes exist overriding reasons for co-locating the data source and access system together.

DESIGN & IMPLEMENTATION

Development Tools: The metawrapper and wrapper components of our system are implemented using Java. Some supporting utilities, such as those used to download and maintain local copies of the aforementioned data sources, are written using a variety of shell scripting languages. Additional class libraries worth noting include ORO Software's PerlTools and the SAX[7] XML parser. Metawrapper and wrapper servlets are accessed through an Apache Web server. Both tiers are hosted by the Jakarta Tomcat servlet engine under Redhat Linux running on Intel x86 hardware.

Protégé has been used to model our ontology and mediated schema[2]. All mapping rule sets are stored in Protégé's knowledge base and accessed on the local machine via provided class libraries. Locally stored data sources are housed by both Oracle and Postgresql. Connectivity between the wrappers and databases is accomplished using JDBC.

Forward and Reverse Mapping Rules: The core of the translation system is a set of forward and reverse mapping rules that drives the semantic translation process. Reverse mapping rules (RMR) convert data source result sets to mediated schema result sets. RMR are also used by the query formulator to determine what information is returned from a given data source and how to parse the XML produced by the metawrapper. Forward mapping rules (FMR) are used to convert mediated schema queries to queries against a particular data source. FMR are also used by the query formulator to determine what parameters may be used to query a particular data source for each entity type. FRM provide information to the query plan formulator that's necessary for cross-source joins.

Reverse Mapping Rules: There are three types of reverse mapping rules: 1) trigger rules, 2) replication rules, and 3) linkage rules. During the translation process, rule types are applied in the order listed.

Trigger rules direct the creation of mediated schema entities. Trigger rules specify an XML path and corresponding entity type. Each time an XML node is traversed, its pathname is evaluated. If the traversed pathname in the data source XML matches that of a trigger rule, then a mediated schema entity of the corresponding type is created. For example, rule $A(pheno):=omim/disease calls for the creation of an entity of type ``pheno" anytime a node with pathname ``/omim/disease" is traversed.

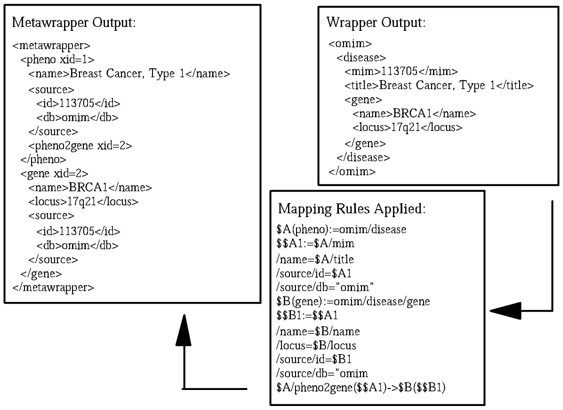
Replication rules direct the grouping of data and population of newly created entities. Replication rules specify both a source and destination XML pathname. Data is copied from a source pathname in the wrapper XML output to a destination pathname in the metawrapper XML output. Replication rules may also be used to define temporary variable storage.

Linkage rules are applied last and are used to establish interrelationships (or "edges") between mediated schema entities. Each entity created by the metawrapper is assigned a unique identifier which is stored in the form of an XML root node attribute called ``XID". Linkage rules direct the addition of references from one entity to another based on certain constraints. For example, the OMIM rule $A/pheno2gene($$A1)->$B($$B1) causes the creation of a link from each entity of type $A to each entity of type $B where $$A1 = $$B1. Note that $$A1 and $$B1 are temporary variables defined during replication.

Figure 2 illustrates a simple set of mapping rules. This example shows a subset of the rules for OMIM that were developed by our group. Note the normalization of data between wrapper and metawrapper output, as <gene> records are extracted from their parent <disease> records and are used to create separate entities.

Forward Mapping Rules: There is only one type of forward rule, and it is used to re-write the query URL sent to the metawrapper. Forward mapping rules describe the type and minimum number of input parameters necessary to query a data source for a particular entity type. For example, the GO rule Gene:{1 of {name}, 2 of {src/id, src/db}} states that either a single {name} or {id, db} pair is required in order to post a query to the GO wrapper when searching for matching entities of type "Gene".

All mapping string information is stored within the Protégé knowledge base. Access to the rules is achieved via Protégé's Java API. One set of reverse mapping rules and one set of forward mapping rules exists for each entity in our mediated schema.



Metawrapper Output:

```
<metawrapper>
  <pheno xid=1>
    <name>Breast Cancer, Type 1</name>
    <source>
      <id>113705</id>
      <db>omim</db>
    </source>
    <pheno2gene xid=2>
  </pheno>
  <gene xid=2>
    <name>BRCA1</name>
    <locus>17q21</locus>
    <source>
      <id>113705</id>
      <db>omim</db>
    </source>
  </gene>
</metawrapper>
```

Wrapper Output:

```
<omim>
  <disease>
    <mim>113705</mim>
    <title>Breast Cancer, Type 1</title>
    <gene>
      <name>BRCA1</name>
      <locus>17q21</locus>
    </gene>
  </disease>
</omim>
```

Mapping Rules Applied:

```
$A(pheno):=omim/disease
$$A1:=$A/mim
/name=$A/title
/source/id=$A1
/source/db="omim"
$B(gene):=omim/disease/gene
$$B1:=$$A1
/name=$B/name
/locus=$B/locus
/source/id=$B1
/source/db="omim
$A/pheno2gene($$A1)->$B($$B1)
```

Figure 2: Application of Mapping Rules

Wrappers: Wrapper construction is source specific and each may differ considerably in design. One API requirement for a wrapper is that it produce a valid XML document which can readily be mapped from source to mediated schema using some set of reserve mapping rules. The other requirement is that each wrapper accept a URL containing supported query parameters. In the event that a data source is unreachable, the wrapper returns an error message and terminates gracefully.

Wrappers can be designed to return more information than is supported by the mediated schema (MS). Information not referenced within RMR is simply ignored and discarded by the metawrapper. Since data sources are sometimes referenced by multiple ontologies, engineering a single wrapper to return the information required by all of those ontologies facilitates wrapper re-use across multiple MS.

Metawrapper: The metawrapper is esponsible for semantic conversion of inbound queries and outbound result sets. Similar to how a human translator provides intermediary communication between two foreign speakers, the metawrapper brokers questions and answers between the query formulator and a specific data source wrapper. The design of the metwrapper's internal parser is generalized enough to allow re-use by all currnet and future wrappers. Externally loaded mapping rules are used to reconfigure the parser at runtime. This allows one version of the metawrapper to provide translation of any supported data source. This approach bears resemblance to parser generator tools such as YACC (Yet Another Compiler Compiler)[8].

The metawrapper accepts as input a single URL. The URL contains query parameters phrased in terms of the mediated schema. The metawrapper examines the URL and decides to which data source to retarget the query. It then applies the appropriate FMR and translates the query to a format compatible with the wrapper's API. The URL is then passed to the wrapper which responds by generating an XML document containing query results. The resulting XML document is parsed and processed by the metawrapper. RMR are applied to the XML document to convert it from source to mediated schema format. The converted query results are then returned to a client such as Tukwila.

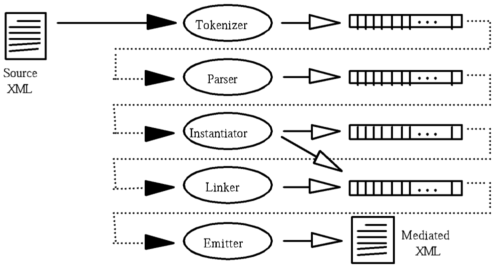


Figure 3: Threaded Processing

The metawrapper's generalized design supports reuse by any number of ontologies, essentially any mediated schema and data source combination who's relationship can be represented by our RMR syntax. It is architected using a "pseudo-compiler" approach[8]. By that we mean that the result set translation process is broken into five distinct phases (see Fig. 3): 1) tokenization, 2) parsing, 3) instantiation, 4) linkage and 5) emission. Each phase of processing within the metawrapper is carried out by a different Java thread. Threads communicate their status and results to one another by way of thread-safe work queues.

Data to be translated is passed from one thread to another, more or less in sequence. The Tokenizer is responsible for parsing XML input from the wrappers. The parser thread is where we see the first application of RMR. Each time the parser encounters an OPEN token, the token's absolute XML pathname is compared to the right hand side of each trigger rule, for example $A(pheno):=omim/disease. For each matching trigger rule, a request is enqueued to the instantiation thread to create a new entity (in our example, of type "pheno"). The instantiation thread is responsible for populating the new entities and applying all RMR replication rules.

Once an entity has been populated, its construction is almost complete. If the new entity has no potential relationship to any other entity types defined by the mediated schema (i.e. the entity is not involved in any RMR linkage rules), it is passed to the emission thread and output to the client. If linkage is required, the linkage thread takes care of creating pointers from one entity to another.

CURRENT STATUS

The metawrapper and wrappers are currently deployed as Java servlets. Wrappers have been written to integrate seven different data sources with our mediated schema. Supported databases include LocusLink, MMDB, OMIM, Entrez, BIND[9], GeneTests[10] and GO[11]. Of this list, two data sources are housed locally and the rest are accessed over the Internet.

GO and LocusLink provide publicly available distributions of their data sets. The GO data set is stored in an Oracle 8.1.6 database, while LocusLink resides in a PostgreSQL 7.0.3 database. Both databases are accessed using JDBC 1.1 compliant drivers. Population and update of local data stores can be conducted whenever new data sets become available. Downloads are performed using FTP and currently take place once per week. Depending on the data source, data sets are available in several different forms including XML, ASN.1 and tab-delimited or other proprietary formats. Additional tools and custom software have been written to load this data into our local databases.

The remaining data sources are accessed via HTTP. Most Web sites that house biomedical databases provide a CGI interface to their query engine, though little or no documentation about its use. Reverse engineering of existing HTML forms is often required in order to gain access to this data.

All servlets, both metawrapper and wrappers, reside on a single machine and are accessed via the same Web server and servlet engine. Access to mapping rules is provided via a local copy of the Protégé knowledgebase files. Data mapping rules are read once when the metawrapper is instantiated, which occurs during startup of the servlet engine.

Testing is currently under way to evaluate the overall efficiency of our system and to collect performance data for later presentation.

DISCUSSION & CONCLUSION

Successes**:** By generalizing the translation component and separating it from the data acquisition layer, complexity of the wrappers was decreased. The amount of time needed to create or modify a wrapper is now minimal[12]. The two tier design of our system promotes parallel development, with programmers able to work on acquisition and translation components concurrently and with little coordination.

Overlapping execution of wrapper and metawrapper functions allows for modularity without sacrificing performance. The time from beginning to end of wrapper output can be several seconds. This time is not wasted as the metawrapper begins simultaneously processing wrapper results.

Simple and minimal API requirements make parameter parsing and generation of output straightforward. The widespread support for Web servers makes our choice of an interface very portable. Wrappers can range in complexity from a simple CGI written in any language to a servlet and beyond. JDBC makes Java a good choice for wrapper development because of its support by a large number of database manufacturers.

XML proved a good choice for representation of both intermediate and final result sets. XML libraries are available for most popular programming languages and both parsing and generation of XML documents is relatively easy.

***Current Challenges*:** One of the problems we encountered was data source instability. Wrappers can break when changes are made to a data source, thus care must be taken to account for this eventuality. On at least one occasion, the OMIM wrapper ceased to function. Upon closer inspection, we discovered that the "screen-scraping" technique employed to interface with OMIM's Web site was no longer correctly parsing HTML pages. This points to the need for a closer relationship between our system's developers and the data source providers.

One alternative to remote access is downloading and accessing data sources locally. In some cases this is not possible because downloadable data sets are not provided: In others it is required. Sites such as GO do not provide an interface that exposes all of the search options needed to facilitate searching on the mediated schema. Local data sources are more reliable, but may often be out of date.

The most difficult challenge in developing this system was creating the RMR syntax and designing the general translation portions of the metawrapper that apply those rules. Fortunately, development of the translation layer is a one time expenditure.

***Future Challenges*:** We anticipate the need for a more robust mapping rule syntax. Also, more time will be required to manage the system as the number of wrappers and data sources increase. This will not be a service-free subsystem, but one that requires attention. Development of tools to monitor and update local databases will likely be necessary. The potential for load balancing will become a necessity, and predictably of major importance to the system's scalability and performance. Something as simple as round-robin DNS for metawrapper and wrapper access could be employed.

In terms of the API, metawrapper queries are currently restricted to a single URL. It is foreseeable that our mediated schema may wish to support larger query strings such as the nucleotide sequences required for BLAST searching. Use of a single URL may become cumbersome. A more flexible solution, such as using the HTTP POST method to pass more complex queries to the metawrapper and wrappers, has already been considered.

***Future Development*:** The modular approach to the metawrapper's design facilitates the possibility of writing additional tokenizer classes to accommodate non-XML producing wrappers. It also facilitates the creation of alternate emitter classes that would produce output in some form other than XML.

An integral intent in our design is to be able to re-use wrappers for multiple ontologies without modifying the wrapper application. Further work in this area should attempt to exploit this possibility.

As of yet, there has been no talk of a security model for accessing the metawrapper and wrappers. Both client authentication and data encryption are areas that may deserve investigation.

ACKNOWLEDGEMENTS

The authors would like to thank Jiang-Jiang Cheng for her programming support. Joint funding was provided by NHGRI and NLM (1R01HG02288).

REFERENCES

1: Macauley J, Wang H, Goodman N. A Model System for Studying the Integration of Molecular Biology Databases; Bioinformatics, 14(7):575-82.

2: Mork P, Halevy A, Tarczy-Hornoch P. A Model for Data Integration Systems of Biomedical Data Applied to Online Genetic Databases. Proceedings of the AMIA Annual Symposium; 2001 Nov 3-7; Washington D.C., USA.

3: Davidson S, Buneman P, Crabtree J, Tannen V, Overton C, Wong L. BioKleisli: Integrating Biomedical Data and Analysis Packages. In: Letovsky S, editor. Bioinformatics: Databases and Systems. Boston: Kluwer Academic Publishers; 1999. p. 201-212.

4: Rubin D, Hewett M, Oliver D, Klein T, Altman R. Automating Data Acquisition Into Ontologies from Pharmacogenetics Relational Data Sources Using Declarative Object Definitions and XML. Pac Symp Biocomput. 2002;:88-99.

5: World Wide Web Consortium (W3C). Extensible markup language (XML) 1.0 (second edition). W3C; 2000. Available from: URL: http://www.w3.org/TR/2000/REC-xml-20001006

6: http://protege.stanford.edu/

7: http://www.saxproject.org/

8: Fischer C, LeBlanc R. Crafting a Compiler. Menlo Park (CA): Benjamin/Cummings Publishers; 1988.

9: http://www.ncbi.nlm.nih.gov/

10: http://www.genetests.org/

11: http://www.godatabase.org/

12: Kossmann D. The State of the Art in Distributed Query Processing. ACM Computing Surveys; 2000 Sep. p. 28-29.