CS 5974 Independent Study

SchemaMapper: A tool for visualization of schema mapping

Independent Study Report

Advisor:

Dr. Edward A. Fox

Submitted by:

Divya Rangarajan

Fall 2004
Draft 01/24/05

Abstract

The world has changed significantly in the past few years with an increasing thrust towards the use of digital information. Every kind of application domain has found reasons to use digital information sources extensively. As a result, different types of data representation models or schemas have been developed. This poses a problem when there is a need for data integration from several sources. Diverse representations must be merged in order to create a single global representation. Hence there is a need for schema mapping tools that will enable amalgamation of heterogeneous data representations. That goal is difficult to achieve today since existing schema mapping tools are domain unaware. SchemaMapper, a new tool we have developed, tries to be domain aware and hence help speed up the schema mapping process. Further, it supports visualization of the mapping process by using a hyperbolic tree representation. This has not been used before in the context of schema mapping. Although the primary motivation for SchemaMapper comes from ETANA-DL (a digital library to promote integration of information and services from diverse archaeological sites), it can potentially be used in any other similar domains in the future, or further extended for different types of schema mappings. This report describes in detail the prototype developed for exploring the feasibility of such a tool, providing architecture and implementation details. Experiments were conducted to evaluate SchemaMapper and the initial results have been very encouraging. All the schemas used during the evaluation process were real life examples taken from ETANA-DL. Analysis of the evaluation results suggests that domain awareness is extremely useful for the schema mapping process. Also, the linear tree representation of schemas which existing tools use appears to have inherent disadvantages which need to be overcome in order to make the process more effective.

Table of Contents

1. Introduction

1.1 Background

ETANA-DL is an archaeological digital library that is supported by ongoing NSF funded work involving the Digital Library Research Laboratory at Virginia Tech, headed by Dr. Edward A. Fox, in collaboration with several other universities. ETANA integrates archaeological data from several sites into a central repository and provides a web interface for archaeologists (and other interested parties) with many features related to exploration of archaeological information. The data format from different sites is represented using XML schemas and various database schemas. Data in the global repository has only an XML schema representation. Hence there is a need for a schema mapping tool that will enable mapping of local schemas to the global XML schema so that data from the local sites can be easily ported to the global repository, stored in the global data format, and made available through the ETANA web interface.

1.2 Motivation

1.2.1 ETANA-DL

Currently, the integration process of merging the local data from various archaeology sites into the global repository is being handled through programming. It is very time consuming to write a specific piece of code for every new site to be integrated into ETANA-DL. It is primarily for this purpose that an intelligent, domain-aware schema mapping tool is being developed so that the mapping process can not only be made easier but also quicker. In order to understand the problem at hand, let us consider Figure 1 that illustrates the architecture of ETANA-DL:

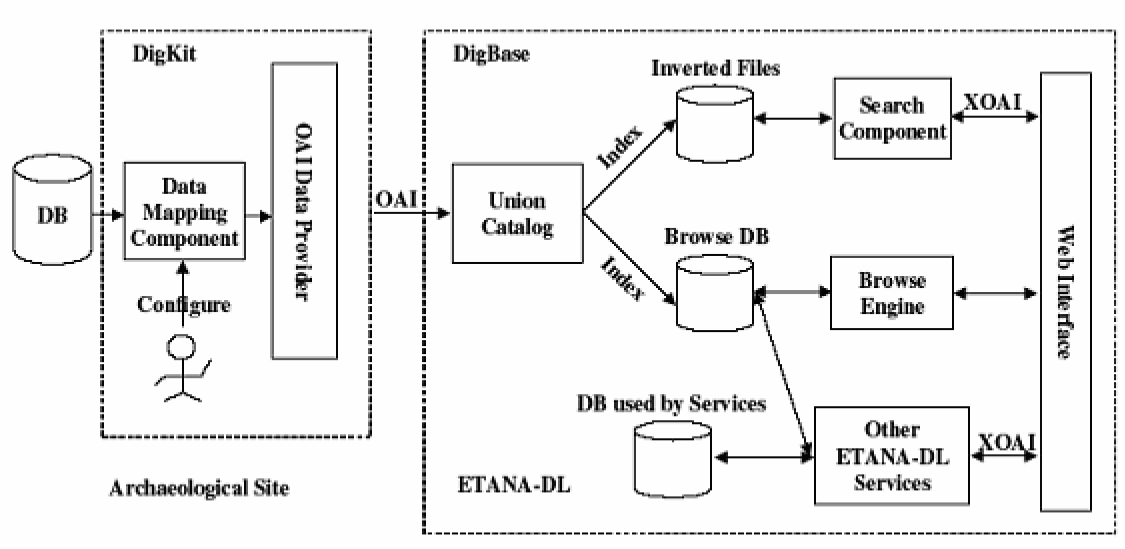


Figure 1: ETANA-DL Architecture

The “Union Catalog”, shown in the above figure, is a repository that contains all the data from various archaeological information sources. Currently, the Union Catalog contains data from six different sites, namely, Nimrin, Lahav, Umayri, Madaba, Tell-Mozan, and BD Cemetery. Data in Nimrin and Lahav are stored in relational database format, Umayri in tab-delimited text files, while Mozan and BD Cemetery use XML representations. In order to harvest data from all these heterogeneous information sources into a central Union Catalog, mapping must be performed between the schema for each individual site and the central schema. The "Data Mapping Component" shown in Figure 1 consists of components that perform this mapping, developed through programming. However, this results in different pieces of code written for each new site to be integrated into ETANA. Therefore, a visual schema mapping tool is essential in order to save time, mitigate the mapping process, and also avoid programming errors. An added advantage of a visual tool is that it does not require the person doing the mapping to be a programmer. A maintainer of such a digital library can use the tool to prepare to harvest the data from new sites.

1.2.2 Existing tools

Several commercial tools are available for the purpose of schema mapping. Such tools have certain issues, some of which have been identified below. Shown in Figure 2 is a screenshot illustrating one such tool, called MapForce™, developed and commercially marketed by Altova®.

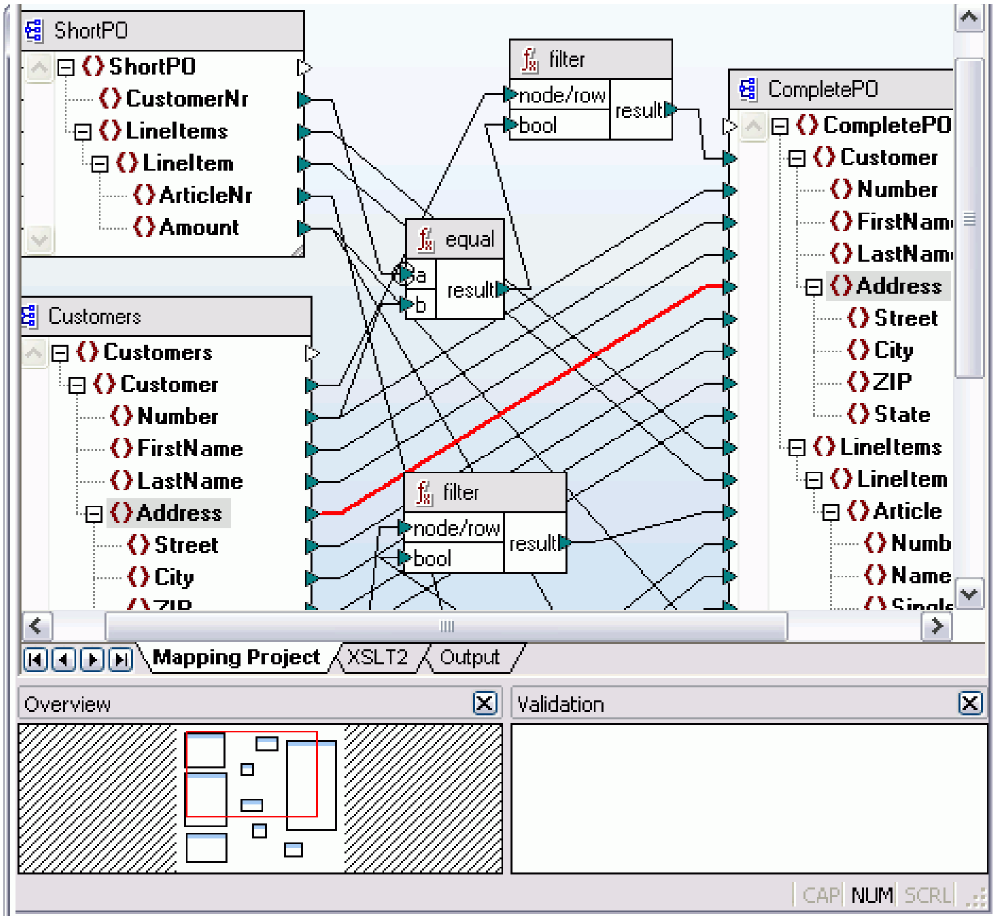


Figure 2: Screen shot of MapForce

One main issue is the use of lines to represent mappings. The lines going across from one schema to another to indicate mapped nodes make it harder to distinguish between mapped and unmapped nodes, especially when the number of nodes being mapped is very large. SchemaMapper attempts to deviate from the line-drawing approach to indicate mappings, as will be described in later sections.

Another disadvantage of MapForce is that it does not allow editing of schemas directly. One has to make use of a different schema editing tool and then reopen the schema in MapForce in order to continue mapping. SchemaMapper will allow editing of schemas from within the tool.

Also, the amount of information that can be visually represented in existing schema mapping tools is not extensive because they all use a linear tree representation to show the schema hierarchies. As a result, a lot of scrolling is required in order to find nodes in the schema, which can become very irritating to the user. SchemaMapper employs a hyperbolic tree approach for visualizing the schemas, therefore giving a better overview and showing more data on the screen. This is helpful, especially for large schema.

Yet another motivating factor for this tool is the lack of domain awareness in currently available software programs. None of the schema mapping tools available today knows anything about the domain to which the schemas belong. SchemaMapper attempts to be domain aware and makes recommendations to the user for what the possible matches are for a particular node in the local schema. This can help speed up the mapping process for the user.

The process of schema mapping is a challenging problem. Although existing tools provide several features, an attempt must be made to address the concerns above as well as others and to strive for a quality tool that is not only better at representing the process visually but is also domain aware and efficient. Although the primary motivation for SchemaMapper results directly from ETANA-DL, the lack or disadvantages of features in existing tools as described above makes a strong case for developing such a tool that can be more broadly deployed too.

2. Design

2.1 Architecture

Conceptually, the architecture of SchemaMapper is divided into four main components which are: Visualization component, Recommendation component, Mapping component, and Data generation component. Figure 3 provides an overview of how these components work together:

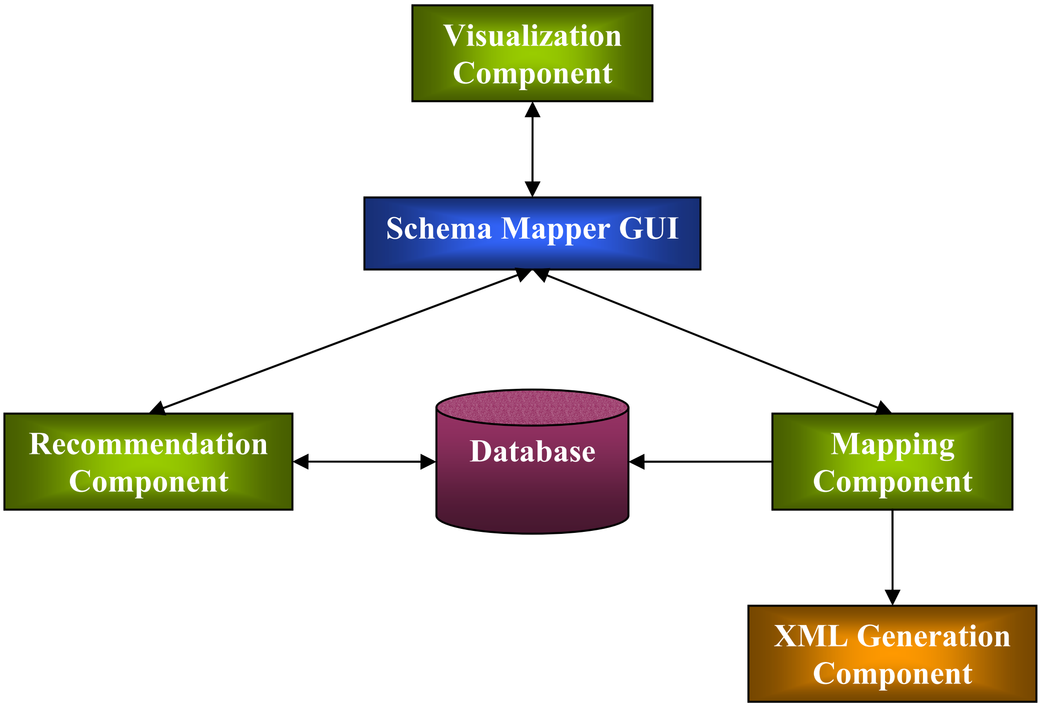


Figure 3: SchemaMapper Architecture

2.1.1 Visualization Component

This component is responsible for all the functions related to visualization of the entire schema mapping process. At present, one visualization method has been implemented for representing the schemas, which employs a hyperbolic tree representation. At any given time, two schemas can be opened for mapping, one local schema and one global schema. This terminology arises primarily from the ETANA-DL domain. A local schema denotes the schema for any new site from which data must be harvested into the Union Catalog. The global schema describes the data stored in the global ETANA database. Although the local schema can be in various formats, support for XML-XML schema mapping has currently been implemented in the tool. In the future, this can be extended to allow database-XML schema mapping as well.

Five main colors are used for the entire process of visualization. The root node is represented in the color yellow, the leaf nodes in green, and the rest of the nodes (that are non-leaf nodes) are in orange. This makes it easy to distinguish among the different types of nodes. It is important to do so because direct mappings among nodes can be established only between leaf nodes since they are the data-containing nodes. Non-leaf local nodes also can be mapped, but only if added as child nodes of global nodes. Also, one-to-one mappings and adding child node types of mappings have been explored in the current prototype. Many-to-one and other complex mappings can be easily added in the future due to the flexible componentized architecture of SchemaMapper.

A convention has been established with respect to visualization of the schemas. The local schema always opens on the left window while the global schema is represented on the right. This distinction is made by using two separate open menu items to make it easy for the user. When the user clicks on a local schema node, it is highlighted in the color red. Immediately the recommended (corresponding) nodes in the global schema also are highlighted in red on the right. A small table is maintained in the bottom panel to show current recommendations. This is useful especially in cases where several recommendations are made and some of those are hidden in the visual representation. The table provides a way to let the user know that more recommendations have been made than are visible in the viewing area.

Once the user is ready to map two nodes after selecting a node in the local schema, he can simply click on the node in the global schema that he wants to map to, and the mapping is done. At this point, the mapping is added to another table in the bottom panel called the mapping table, that keeps track of the mappings in the current session. Both of the mapped nodes change to purple at this point. Also, a color legend table is displayed at the bottom right section of the tool.

The user also is given the capability to undo mappings between nodes. Furthermore, this component provides visual capabilities to edit the global schema. The user can rename nodes in the global schema as well as delete nodes that he finds unnecessary. This feature is available only for the global schema because at no point will the local schema be edited. The local schema is only mapped to the global schema but not changed. Once done mapping, the user can save the mappings by using the menu option for the same. This action saves the mappings in the current session by generating an XSLT style sheet and automatically generates the data in the global XML format by using the local XML data file and the style sheet.

2.1.2 Recommendation Component

This component is responsible for making recommendations to the user to indicate the possible matches for a particular local schema node. This is the piece that introduces domain awareness in the tool. Several methods can be used to make recommendations based on the domain to which the schemas belong. This component can be very useful in cutting down mapping time for users.

One recommendation algorithm has been implemented in the current prototype. It involves finding name-based matches of the local schema node in the global schema for making recommendations. This method of finding possible matches considers the user selected node in the local schema, finds possible name-based matches in the global schema, and returns a list of these possibilities to the visual component. The visual component then highlights these recommended nodes in the global schema view. It also displays a list of recommendations in the "Recommendation" table located in the bottom panel. This makes it easy for the user to view all possible recommendations at one go. The table view was important to incorporate because a situation may arise in which several nodes are recommended to the user. Although all these recommended nodes are highlighted in the global view, all of them might not be visible to the user. The table hence forms the point of interaction between the visualization component and the recommendation component.

In this prototype, the given global schema is analyzed for finding appropriate recommendations. However, in the future, this will be extended to interact with a database as shown in the figure above to use saved mappings from previous sessions and make recommendations based on the history stored. This design of interactions between the visual component and recommendation component as well as the flexibility of the latter makes it easily extensible to employ other recommendation techniques that involve communication with a database, thus increasing domain awareness capabilities.

2.1.3 Mapping Component

The mapping component is responsible for doing the mappings and for keeping track of them. As soon as the user establishes a mapping between two nodes, this component saves the information in its data structures. As shown in the architecture diagram above, the visualization component is responsible for communicating with the mapping component each time a mapping is established in order to update the information. This component also interacts with the XML generation component in order to create the data in the format that conforms to the global XML format once all the mappings have been made. In the future, support will be added for this component to store the mappings to a database for use in subsequent mapping sessions.

2.1.4 XML Generation Component

Once the mappings are done, the XML generation component creates an XSLT style sheet based on current mappings stored by the mapping component and the given local and global schemas. It then takes all the local data files and generates the global data files based on the style sheet.

3. Users' Manual

Given below are specific instructions on how to operate SchemaMapper. The screenshots referred to in this guide are available in Appendix A of this document.

1. The tool is packaged in a zip file called `SchemaMapper.zip`. Unzip it into any directory of your choice. To run the program, double click on the batch file named `run.bat`.

2. SchemaMapper Graphical User Interface:

    1. *Opening a local schema*: (Screenshot 1)
       Use the "Open Local Schema" menu item to open a new local schema. It will pop up a file selector dialog. After making a selection, click "Open". The local schema will always appear on the left panel of the main window.

    2. *Opening a global schema*: (Screenshot 1)
       Similarly, use the "Open Global Schema" menu item to open a new global schema. After making the selection from the file selector dialog box, click "Open". The global schema will always load on the right panel of the main window.

    3. *Selecting nodes*: (Screenshot 2)
       To select a node in the local schema, simply left-click on it. It will highlight in red showing that it has been selected. As soon as a selection is made, the corresponding recommendations in the global schema also will be highlighted in red. The recommendations will be added to the "Recommendations" table in the bottom panel.

    4. *Mapping nodes*: (Screenshot 3)
        a. Select a local schema node you wish to map.
        b. Left-click on a global schema node you wish to map to.

       Once mapped, the color of the mapped nodes will change to purple and the mapping will be recorded in the mapping table on the bottom left corner as shown below.

    5. *To unmap nodes*: (Screenshot 4)
        a. Right click on the *local* schema node you want to unmap.
        b. Select "Remove Mapping."

    6. *Adding a local schema node as a child of a global schema node*: (Screenshot 5)
       Child nodes can be added only to non-leaf global schema nodes. Here is the procedure:
        a. Select a local (leaf or non-leaf) schema node.
        b. Select a non-leaf global schema node to map to. A dialog will appear asking whether you want to add the node as a child. Click "Yes" to confirm. The local node along with all its children will be added if it is a non-leaf node as a child node of the global node selected, and also mapped.

    7. *Renaming nodes* (allowed only in global schema): (Screenshot 6)
        a. Right click on a node and select "Rename".
        b. Enter the new name and click "OK".

    8. *Deleting a node* (allowed only in global schema): (Screenshot 6)
        a. Right click on a node.

    b. Select “Delete”.

9. *Saving mappings*: (Screenshot 7)
    a. Select the Options menu item at the top.
    b. Select “Save Mappings”.

   This action will generate an XSLT style sheet in the same directory as the main project directory. It assumes that the local XML file is located in the same directory as well. The output file is generated in the same directory.

4. Developers' Manual

4.1 Functionality Implemented

The functions currently offered by the tool for the process of schema mapping are described below:

1. *Loading schema files*:
   Currently, support for opening XML schema files (or XSD files) is available. An assumption is made that the global schema is always in XML format as is the case in ETANA-DL. Therefore, mapping between XML schemas has been implemented.

2. *Mapping nodes*:
   A set of mapping rules have been established and are enforced during the process of mapping. These are:
   - Only leaf nodes can be mapped directly to one another.
   - A *local* non-leaf node can never be mapped to a *global* leaf node.
   - A *local* non-leaf node can be added as a child to any *global* non-leaf node.
   - A *local* leaf node can be added as a child to any *global* non-leaf node.

3. *Recommendations*:
   Recommendations are made to the user in order to indicate what the possible matches might be in the global schema. Currently, these recommendations are based on analyzing the local and global schemas using a name-matching based algorithm. The user can choose to ignore recommendations if he does not like any of them and to pick a node of his own choosing to which to map the local node. Thus, the ultimate control lies in the hands of the user.

4. *Unmapping nodes*:
   Support has been added to remove mappings between nodes that have been made in the current session.

5. *Editing the global schema*:
   Editing is allowed only for global schemas as the local schema is only mapped to the global schema but never edited. Three main editing functions are provided which are:
   - Renaming nodes
   - Deleting nodes
   - Adding new nodes as children of existing global nodes

6. *Saving mappings*:
   Once the user is done mapping the schemas, the mappings are saved in an XSLT style sheet.

7. *Generating output XML data*:
   SchemaMapper generates the output XML data based on the local XML data and style sheet generated.

4.2 Implementation Details

SchemaMapper has been implemented entirely in Java. The hyperbolic tree library used for visualizing the schemas is an open source library available for free academic use from sourceforge.net. Following is a description of each of the classes used:

4.2.1 schemaNode.java

This class manages the main data structure that stores schema information. Each `schemaNode` object contains several data members for representing information about that node. This includes the name of the node, a pointer to the parent of the node, a vector of all the children of this node, the color to be used while visualizing this node, and a few other pieces of information such as whether the node is mapped, highlighted, or recommended. The entire schema is read into an instance of this structure at the very beginning when the user opens a new local schema. The global schema is read into another instance of this structure. These two object instances are used thereafter for the entire mapping process. The actual schema file is not referred to after this first step. This makes the functioning efficient as the objects are stored in memory. The structure of the `schemaNode` class is flexible enough so that it can be modified in the future to accommodate other possible pieces of information about different schemas, as and when required. Detailed information about each method in this class is available in the code comments and Javadocs.

4.2.2 schemaMapperGUI.java

This class contains all the visualization logic. It sets up the entire graphical user interface for the tool. An open source library called `hypertree` has been employed for visualizing the schemas as hyperbolic trees. This class contains the logic for loading schema files into the appropriate data structures for manipulation as well as visual use. It also acts as the driver for features such as renaming nodes, deleting nodes, making recommendations, mapping and unmapping nodes, and finally saving the mappings. The GUI logic populates the recommendation table each time new recommendations are made and clears out the old ones. It also takes care of updating the mapping and recommendation tables each time a mapping is complete. Thus, this class is responsible for making calls to the recommendation component and the mapping component. In essence, it glues together the different components and acts as the driver for the entire mapping process.

4.2.3 MappingComponent.java

This class contains the data structures for storing mappings. It stores the mappings in a `hashtable` and provides accessor and mutator methods for the data members to keep track of the mappings. This class can be extended in the future for database connectivity to store mappings across various sessions. It interacts with the XML generation components in order to create the style sheets and the global XML data files.

4.2.4 RecommendationComponent.java

This class contains the logic for making recommendations. Currently, one method of making recommendations has been implemented. This method is the name-based recommendation method. It analyzes the global schema for possible name matches for the selected local schema node and returns a list of possibilities in the form of a vector to schemaMapperGUI.java. The GUI then highlights the corresponding nodes in the global schema in red, as well as adds the recommendations to the table in the bottom panel. In the future this can be extended to accommodate other types of recommendation algorithms.

4.2.5 Xsltgen.java

Once the user has finished mapping the schemas and selects the “Save Mappings” menu item, a call to an instance of this class is made. It generates an XSLT style sheet describing the mappings in the current session, by referring to the mapping component object. It then invokes and instantiates the XSLTStreamTest.java class and generates all of the global XML files, based on the local XML files.

4.2.6 XSLTStreamTest.java

This class is instantiated by the Xsltgen.java class and used to generate the global data. It makes the assumption that all of the local XML data files are stored in the same directory as the project directory. It generates the global XML data files by making use of the XSLT style sheet generated by the Xsltgen class as well as the local XML files. The data created by this component thus conforms to the global schema. The Xsltgen.java and XSLTStreamTest.java classes together form the XML generation component, a key part in the architecture of SchemaMapper.

For Developers: Further code details and descriptions of all the class methods are located in the Java files as well as Javadocs.

4.3 Suggested Improvements

This tool is a prototype developed as a proof of concept for doing schema mapping using techniques that had not been explored by schema mapping tools before. Its architecture is flexible and componentized, therefore making it easily extensible, for additional functionality and enhancements. Following is a list of improvements recommended for the tool, based on suggestions received as well as pilot test results:

1. *Saving the global schema file*:
   SchemaMapper presently generates an XSLT style sheet when mappings are saved, which is then used to create the global XML files based on the local XML files. A necessary addition to the tool is the capability to save an edited global schema file. This can be implemented by modifying the schemaNode class. Schema information is first read into a DOM object which is analyzed and used to populate the schemaNode object. If schemaNode is modified to extend the DOM node object itself, the feature to save the global schema can be added easily.

2. *User interface improvements*:
   a. Tooltip currently displays only the name of the node on a mouse-over event. It can be extended to include other information about the node such as the node type, sample values, and so on. This can be done by modifying the hyperbolic tree library code to fetch information other than just the name, and to include it while visualizing the tooltip.
   b. The mapping table in the bottom left corner can be improved so that when a user clicks on a mapping, the hyperbolic trees reorient themselves to reveal those particular mapped nodes if they are hidden at that point in time. This can be done by capturing the mouse click event on a cell of the table, fetching the value in that cell, and sending an appropriate mouse-event to the hyperbolic tree library code to reorient the tree and bring that particular node in focus.
   c. The recommendation table also can be improved by allowing the user to click on a particular recommendation in the table and by reorienting the global hyperbolic tree to reveal that specific recommendation. This way, if a recommended node is hidden,

the user can bring it into focus. The procedure for this enhancement is the same as in the previous case.

   d. The sizes of the hyperbolic trees currently are not affected by the size of the computer monitor being used. This can be fixed by modifying the hyperbolic tree library code to use the screen size information instead of having a fixed size.
   e. When a node is renamed, some sort of indication can be incorporated in that node so that the user can look and remember which node was renamed. This can be achieved by adding a visual attribute to the renamed node. This helps the user keep track of edits to the global schema.
   f. Similarly, when a node is added as a child of a global node, an indication to the user of the same would be useful. Then, the user can keep track of what information has changed in the global schema. Currently, no indication is given, which can be disorienting to the user.
   g. Enabling double-clicking for renaming a global node can prove to be a shortcut for this purpose. This can be implemented by capturing a double-click event.

3. *Further editing capabilities for the global schema*:
   a. Sometimes, users might want to reorganize the global schema. Capability can be added to move an existing node in the global schema to another location within the schema. This can be achieved by deleting the selected node from its parent node and adding it as a child of the new node selected by the user, according to where he wants to put the node.
   b. Allowing selection of multiple nodes in both local and global schemas can be useful. This is because a user might want to add a bunch of nodes in the local schema as children of a particular node in the global schema. Currently, the user can only do this one by one. In the global schema, a scenario where this might be useful is when the user wants to delete several nodes at the same time.
   c. Adding a new node in the global schema is a useful feature. This can be achieved by allowing the user to create a new node as a child of a global node. The new node must be added to the corresponding global schemaNode object in the correct location, which can be obtained by capturing the mouse-click event.

4. *Database connectivity*:
   Another necessary feature is the ability to store mapping history across several sessions in one location. Currently, mappings are stored in the form of an XSLT style sheet. This method will generate a different XSLT style sheet for each mapping session. Future recommendation algorithms will refer to mapping history stored across many sessions. Searching through these XSLT files for this purpose is not only troublesome but also highly inefficient. A better way is to connect to a database and store all the mappings in tables so that in the future the database can be simply queried to look for mappings. This way, all the mappings will be stored in one location. This functionality can easily be achieved by having the mapping component connect to a database and by storing the mappings from each session.

5. User Evaluations

An experiment was set up for evaluating the prototype. The primary focus of the user evaluations revolved around usefulness of such a mapping tool with domain awareness, and comparison with existing mapping tools (specifically MapForce™) for measuring ease of use and usefulness of features.

5.1 Experimental Set-up

Two tools were used to perform the tasks, SchemaMapper and MapForce. Six users were employed as subjects for the experiment. One user out of these was the pilot tester for the tool. Therefore, quantitative measures were not considered for this user. Timings and number of scrolls were measured for all the other users for the first benchmark task. The schemas used were real examples taken from ETANA-DL. The local schema was called cemetery.xsd, which described pottery data from a cemetery site. The mapping target was the ETANA-DL global schema. Four out of all the users were familiar with ETANA-DL and the schemas involved in the same, while the other two were not. This provided a good sampling because familiarity with the schemas and knowledge of existing nodes could be accounted for in the tests. Another factor that helped reduce user bias is that some of the users were well versed with using MapForce while some others were not. (Please be specific as to numbers in each situation; then delete my sentence.)

Three benchmark tasks were provided to the users for performing the evaluations. The first task required the users to map six nodes in the local schema to nodes specified in the global schema using MapForce first and then SchemaMapper. The second one required them to rename a node in the global schema using both tools, in sequence.(?) The last task required using only SchemaMapper, and had the users add a node in the local schema as a child of a node in the global schema. Then, they had to rename a specified node and map four nodes. Thus the benchmark tasks ensured that all the primary functionality provided by SchemaMapper was evaluated. Table 1 summarizes the tasks assigned to the users:

Table 1: Benchmark Task Details

Task no.	***Benchmark Task***	***Instruction Details***
BM 1	Direct mapping of nodes	The users were asked to use MapForce first and open the pottery (local) schema and the ETANA (global) schema. This version of the global schema was already edited to contain nodes from the pottery schema so that direct mappings could be established without requiring the users to edit the global schema. They were asked to map directly the following nodes: ObjectType PotteryImageURL OwnerID Volume Collection Height They were then asked to repeat this task using SchemaMapper.

BM 2	Renaming a node	The users were asked to use MapForce first and then SchemaMapper, and to rename the node called VOLUME to VOL.
BM 3	Adding a local schema node as a child of a global schema node	Users were required to use only SchemaMapper for this task. They were asked to open the local pottery schema and the old ETANA schema that did not contain pottery information. They were then asked to add the local schema node named Pottery_Attributes as a child of the global schema node OBJECT. Thereafter, they were instructed to rename Pottery_Attributes to POTTERY. Finally, they were asked to map the following nodes directly from the local schema to the global schema: ObjectType PotteryImageURL OwnerID Volume

5.2 Results

5.2.1 Quantitative Results

Based on the first quantitative measure, all users took more time to complete the first task using MapForce than SchemaMapper. They found the action of using lines to join nodes for mapping in MapForce harder to use than simple clicks in SchemaMapper. Another quantitative measure was the number of times they had to scroll to find nodes in MapForce versus the number of times they had to reorient the hyperbolic trees in SchemaMapper. Every user scrolled many more times in MapForce than they reoriented the hyperbolic trees in SchemaMapper except for one user (User 2). However, this user was not able to complete the task and hence did not scroll at all in MapForce. This user declared the task as complete after mapping only 3 out of the 6 nodes required for BM1 in both tools. Overall, the mapping process took more time in MapForce than in SchemaMapper. Thus, when comparing SchemaMapper with MapForce, the quantitative results were surprisingly positive. Table 2 summarizes the quantitative results from the experiment. Quantitative data was available only for BM1.

It should be noted, however, that some of the advantage observed for SchemaMapper might be a result of learning, since MapForce was worked with first. Future experiments should adjust for possible learning effects by having an equal number of cases where MapForce was second.
(Please move text so no table or figure is split across a page.)

Table 2: Quantitative Results Raw Data

Task	*User*	*Time using MapForce (seconds)*	*Time using SchemaMapper (seconds)*	*Number of scrolls using MapForce*	*Number of reorient actions using SchemaMapper*
BM1	User 1	4.22	1.53	16	5
BM1	User 2	3.13	1.25	0	3

BM1	User 3	1.38	1.25	12	2
BM1	User 4	3.48	2.00	10	3
BM1	User 5	1.45	1.22	10	2

5.2.2 Qualitative Results

Each user was given a questionnaire at the end of each task. All users strongly preferred the ability of SchemaMapper to allow editing from within the same tool. MapForce required them to edit the schema in a different tool and then come back and reopen the schema in MapForce to continue mapping. Also, most users found the amount of scrolling involved in finding nodes in MapForce annoying and preferred to use the hyperbolic tree navigation technique offered by SchemaMapper. Providing recommendations to the user proved to be very useful and strongly preferred by them. Users were able to save time looking for nodes in the global schema to map to, by using the recommendations. That was the main time saver for mapping nodes. However, the quality of the recommendations themselves and of the mappings has yet to be evaluated.

Another surprising result was that although until now hyperbolic trees have never been used for the purpose of schema mapping, users were not confused by the hyperbolic tree representation. They all liked the fact that it can show more of the data at one time on the screen. Users also gave positive feedback about using colors to denote mappings rather than lines going across the schemas. One negative feedback some users provided was that although the full name of a node in the hyperbolic tree was provided as a tooltip, the truncated names displayed as node labels proved to be a little confusing and were the cause for some errors while mapping. Table 3 summarizes the qualitative results:

Table3: Qualitative Results Summary

SchemaMapper Feature	***Positive Feedback***	***Negative Feedback***
Recommendations	• Finding correct matches for nodes is easier and very convenient. • Mapping process is made much faster and more streamlined. • Rated as very useful feature by all users.	• Hyperbolic tree view makes it harder to view all recommendations at one time.
Use of color and mapping table to indicate mapped nodes vs. lines in MapForce	• Useful having no lines across the two screens. Keeps the screen uncluttered. • Use of a different color for mapped nodes very useful.	None
Ability to edit global schema from within SchemaMapper vs. use of a different tool (XML Spy) while using MapForce	• Very convenient to edit from within the same tool. • Saves time. • Deleting and renaming nodes easy to perform.	• Method for adding a child in global schema not so intuitive. • No way to view earlier changes made within current session.
Navigating schemas	• Easier to navigate using hyperbolic tree.	• Hyperbolic tree does

	• Requires far less scrolling than linear tree. • Hyperbolic tree representation not confusing. • Better visual overview with hyperbolic trees.	not allow collapsing of nodes that are not being considered. • Hides some node names. Truncation of names can cause errors.

6. Conclusions and Future Work

6.1 Conclusions

Judging from the results of initial evaluations described above, the development of SchemaMapper as a prototype for an intelligent schema mapping tool has proved to be a very fruitful experience. Although several schema mapping tools are available today, most of the tools are commercial and expensive to purchase. This report also has described some of their disadvantages; these SchemaMapper has tried to overcome, and will continue to improve upon. Some takeaway observations from this experience are:

- Schema mapping is a process that is significantly more effective when done visually rather than through programming. Users prefer visual tools for this purpose over writing a program to map two schemas. It saves them time and effort, and also reduces the number of errors.

- Visualization of the schema mapping process can be improved many fold. Observations and testimonials gathered during the experiments revealed that most people rarely look back at nodes they have finished mapping. Hence the use of lines to show mappings is not very useful. In fact, it is confusing to have the lines in some cases and can be avoided. The use of colors to denote mappings was effective.

- The feasibility of making recommendations to the user for the mapping process, so it will be more efficient, has been demonstrated. Also, it has been observed during experiments that this is a very useful feature and will be used extensively if provided to users.

- Large amounts of scrolling is very irritating for users. They would prefer to avoid scrolling schemas vertically if alternate mechanisms are provided.

6.2 Future Work

SchemaMapper can be extended in functionality in the future. The flexibility of its architecture makes it easy to incorporate new features. Some of these have been brainstormed and are suggested below:

1. *Additional methods for recommendation*:
   Different types of techniques may be used to make recommendations to the user by implementing different recommendation algorithms. Additional methods may be explored and added to the recommendation component. An extra benefit could be achieved by allowing the user to load a dictionary of domain specific words into the tool so that recommendations can be made based on those as well. The user also may be given the option to choose from among a pool of recommendation techniques.

2. *Additional mapping capabilities*:
   Further mapping capabilities can be added to the tool for allowing many-to-one mappings as well as more complex mappings that require concatenation or mathematical operators. The ETANA domain has not required this feature until now. However, it might be useful in the

future when additional archaeological sites are added, or when this tool is used in a different domain.

3. *Support for mapping other types of mappings*:
   In the current prototype, XML-to-XML schema mapping has been implemented. SchemaMapper can be extended in the future to support database schema mappings as well as metadata schema mappings.

4. *Other visualization options*:
   Another feature that might prove useful is the capability of using different views from within the same tool. As of now, the schemas are visualized only as hyperbolic trees. Other types of visualizations such as linear tree visualizations can be incorporated as options to the user. This feature is also useful for performing future evaluations to measure the effectiveness of schema mapping using different types of visualizations.

Acknowledgements

I would like to thank Dr. Fox for being my advisor and guide. I am sincerely grateful to him for all the support and valuable input he has provided through the design and development of this project and in courses I have taken under him. He has been a constant source of encouragement for me and this work would not have been possible without his guidance.

I would like to express my gratitude to Rao Shen for all her input and the time she has spent involving herself in this project. She has been very helpful regarding defining the project scope and in working with us to crystallize ideas into concrete designs.

I also would like to thank Ananth Raghavan for working with me on this project and I wish him all the very best in continuing this work for his thesis.

I would like to thank all the members of DLRL for their useful insights and suggestions to help improve the tool.

Finally, I thank the National Science Foundation for its support of ETANA through ITR grant IIS-0325579 to CWRU, with subcontract to Virginia Tech.

References

[1] Ravindranathan, U., Shen, R., Goncalves, M. A., Fan, W., Fox, E. A., and Flanagan, J. W. *ETANA-DL: A Digital Library For Integrated Handling Of Heterogeneous Archaeological Data*. Presented at Joint Conference on Digital Libraries (JCDL 2004), Tucson, AZ, June 7-11, 2004.

[2] Nithiwat Kampanya, Rao Shen, Seonho Kim, Chris North, and Edward A. Fox. CitiViz: A Visual User Interface to the CITIDEL System. Full paper in Proc. European Conference on Digital Libraries (ECDL) 2004, September 12-17, 2004, University of Bath, UK.

[3] Lamping, J. and Rao, R., *Laying Out and Visualizing Large Trees Using a Hyperbolic Space*. In Proceedings of the ACM Symposium on User Interface Software and Technology, 1994, 13-14.

[4] *ETANA-DL: Managing complex information applications: An archaeology digital library, 2004*. http://feathers.dlib.vt.edu.

[5] Ling Ling Yan, Renee J. Miller, Laura M. Haas, and Ronald Fagin. *Data driven understanding and refinement of schema mappings*. In SIGMOD Conference, 2001, pages 485-496.

[6] John Lamping, Ramana Rao, and Peter Pirolli. *A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies*. In Proceedings of SIGCHI conference on Human Factors in computing systems, pp. 401-408, 1995.

[7] E. Rahm, and P.A. Bernstein. *A survey of approaches to automatic schema matching*. VLDB J. 10:4 (2001), pp. 334-350. http://citeseer.ist.psu.edu/rahm01survey.html

[8] Chris North, Nathan Conklin, and Varun Saini. *Visualization Schemas for Flexible Information Visualization*. In Proceedings of the IEEE Symposium on Information Visualization (InfoVis '02), October 28 - 29, 2002.

[9] Ahlberg, C., and Wistrand, E. *IVEE: An Information Visualization and Exploration Environment*. In Proceedings IEEE Information Visualization '95, pp. 66-73, 1995.

[10] Robertson, G., Mackinlay, J., and Card, S.: *Cone Trees: animated 3D visualizations of hierarchical information*. Human factors in computing systems conference proceedings, on reaching through technology. ACM Press, New Orleans, Louisiana, United States (1991), 189-194

[11] Cockburn, A. and McKenzie, B.: *An Evaluation of Cone Trees*. In Proceedings of the 2000 British Computer Society Conference on Human Computer Interaction, University of Sunderland, September 2000.

[12] Zylab. *Visualization Module*, 2004 http://www.zylab.com/products_technology/productsheets/Visualization.pdf

[13] Altova. *Mapforce*, 2004 http://www.altova.com/products_mapforce.html

[14] Microsoft. *BizTalk Mapper*, 2004 http://www.samspublishing.com/articles/article.asp?p=26551&seqNum=5

[15] Haber, Eben M., Ioannidis, Yannis E., and Livny, Meron. *Opossum: A flexible schema visualization and Editing Tool.* Conference on Human Factors in Computing Systems. Boston, Massachusetts, 1994, pages 321-322.

[16] Johnson, B. and Shneiderman, B., Treemaps: *A Space-filling Approach to the Visualization of Hierarchical Information Structures*. In Proceedings of the 2nd International IEEE Visualization Conference (San Diego, USA, 1991), 284-291.

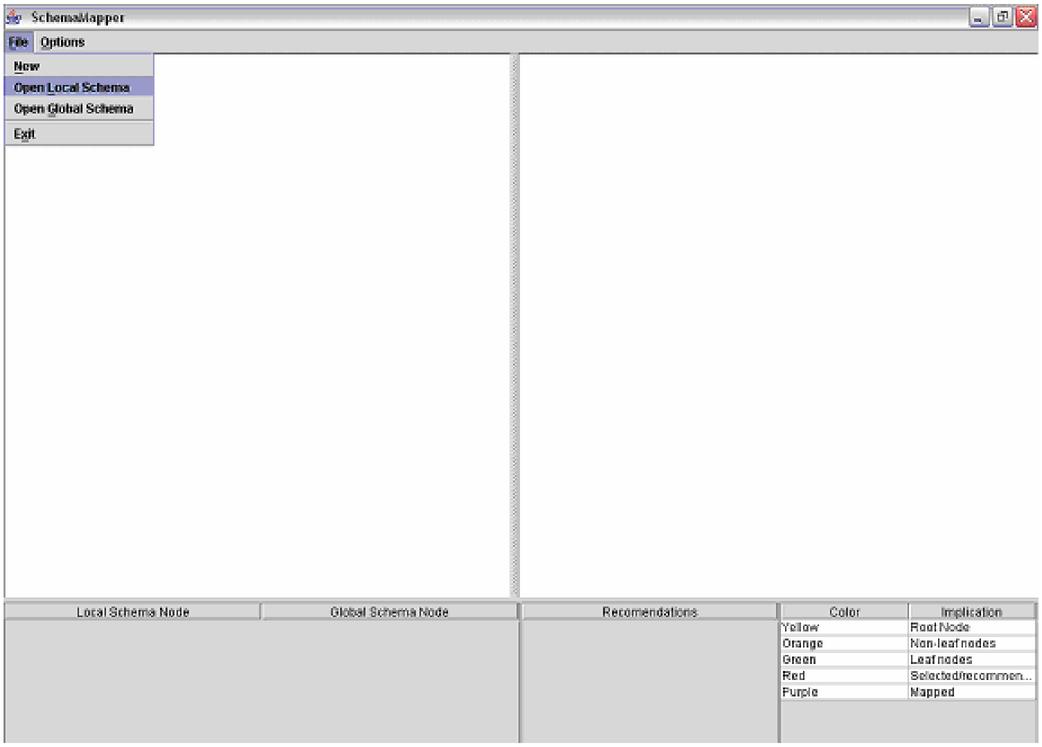
[17] Shneiderman, B., The Eyes have it: *A Task by Data Type Taxonomy*. In Proceedings of IEEE Symp. Visual Languages 96 (1996), 336-343.

[18] Mapforce vs. BizTalk Mapper, 2004. http://www.osnews.com/story.php?news_id=6809

[19] Boardman, Richard. *Bubble Trees: Visualization of Hierarchical Information Structures*. Conference on Human Factors in Computing Systems, The Hague, Netherlands (2000), pages 315-316.

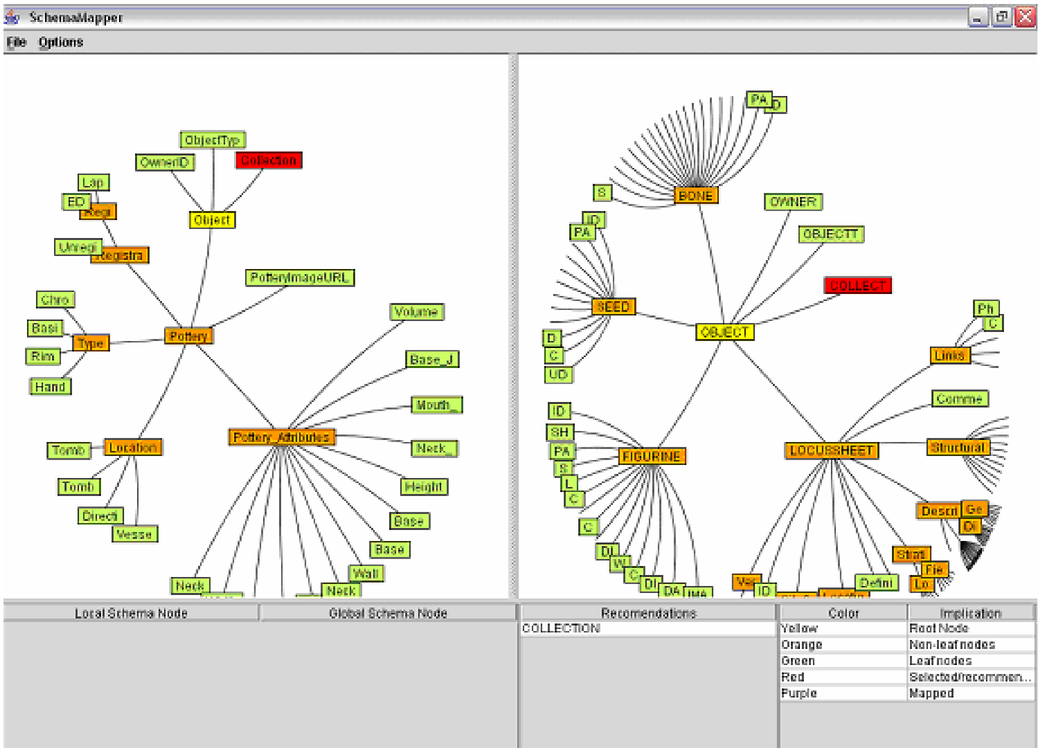
[20] [20] R. J. Miller, L. M. Haas, and M. Hernandez. *Schema Mapping as Query Discovery*. In Proc. of the Int'l Conf. on Very Large Data Bases (VLDB), pages 77-88, Cairo, Egypt, September 2000.

Appendix A: Screenshots for Users' Manual



Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 1: Opening a local/global schema

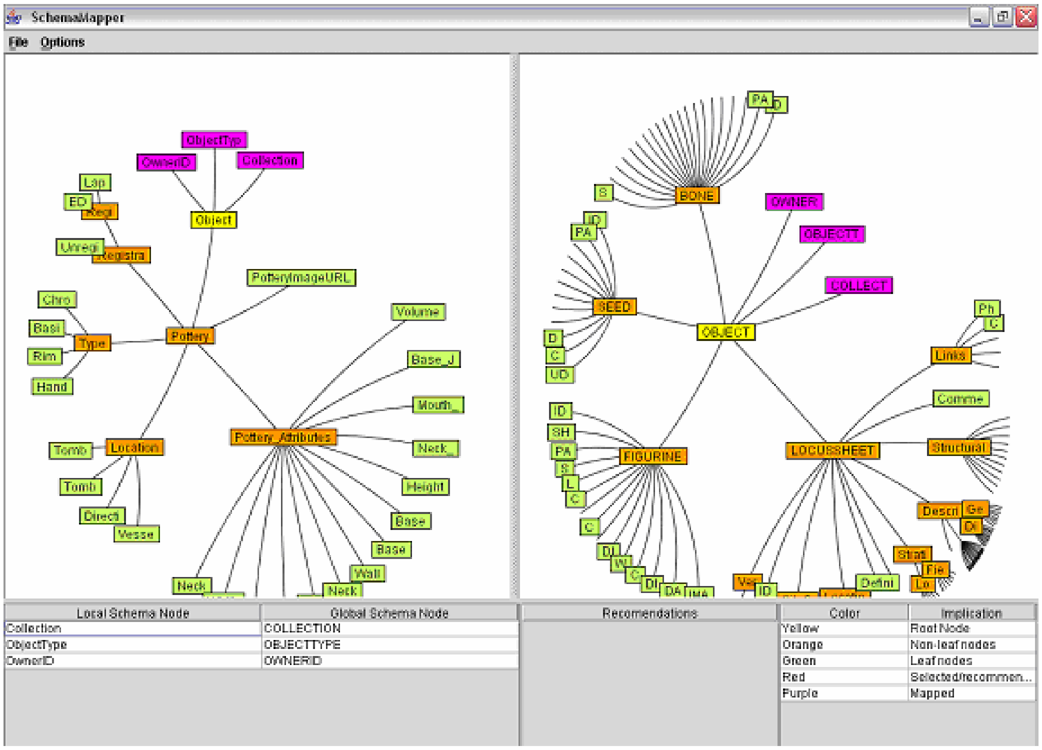


Local Schema Node	Global Schema Node

Recomendations
COLLECTION

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 2: Selecting nodes

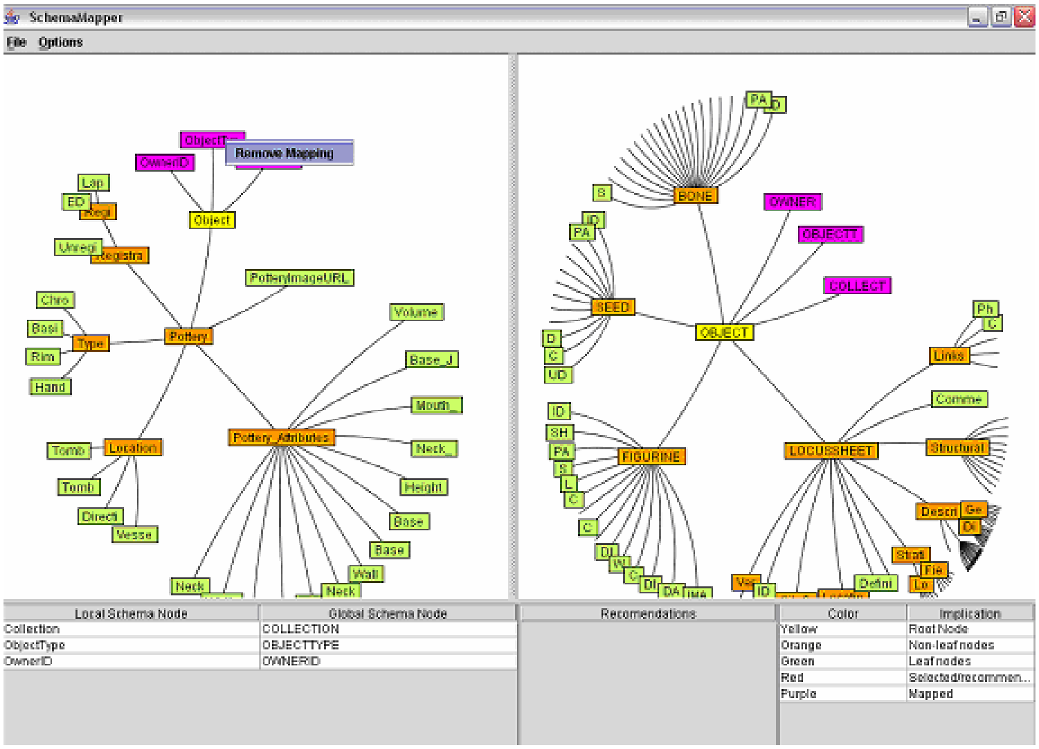


Local Schema Node	Global Schema Node
Collection	COLLECTION
ObjectType	OBJECTTYPE
OwnerID	OWNERID

Recomendations

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 3: Mapping nodes

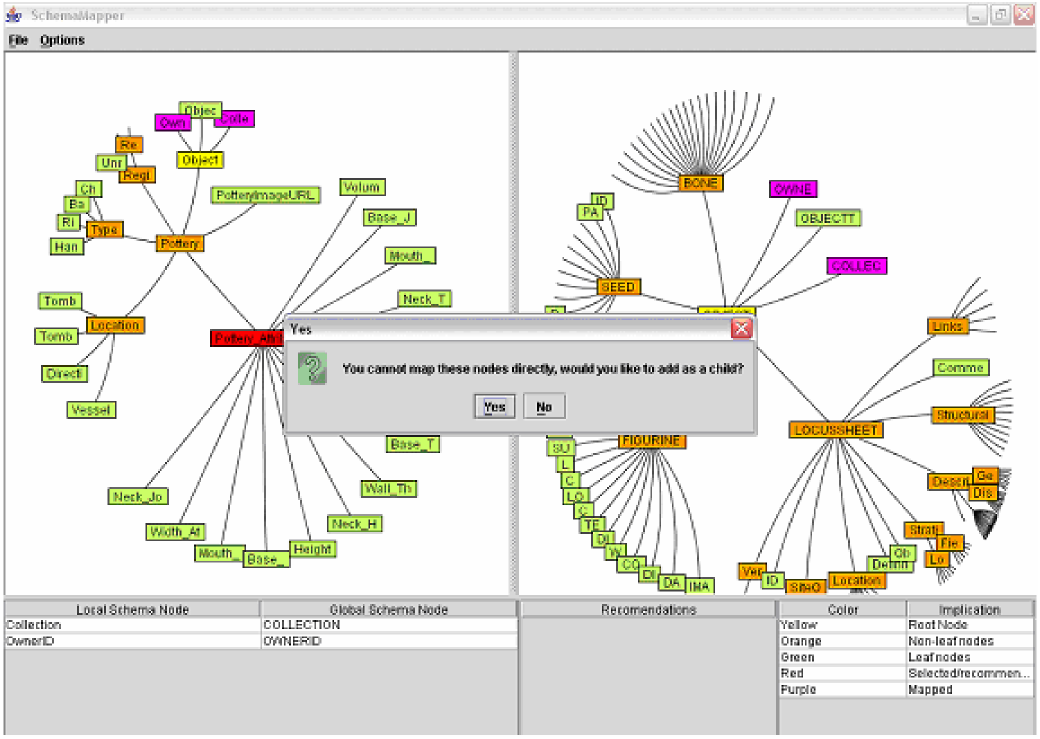


Local Schema Node	Global Schema Node
Collection	COLLECTION
ObjectType	OBJECTTYPE
OwnerID	OWNERID

Recomendations

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 4: Unmapping nodes



Yes

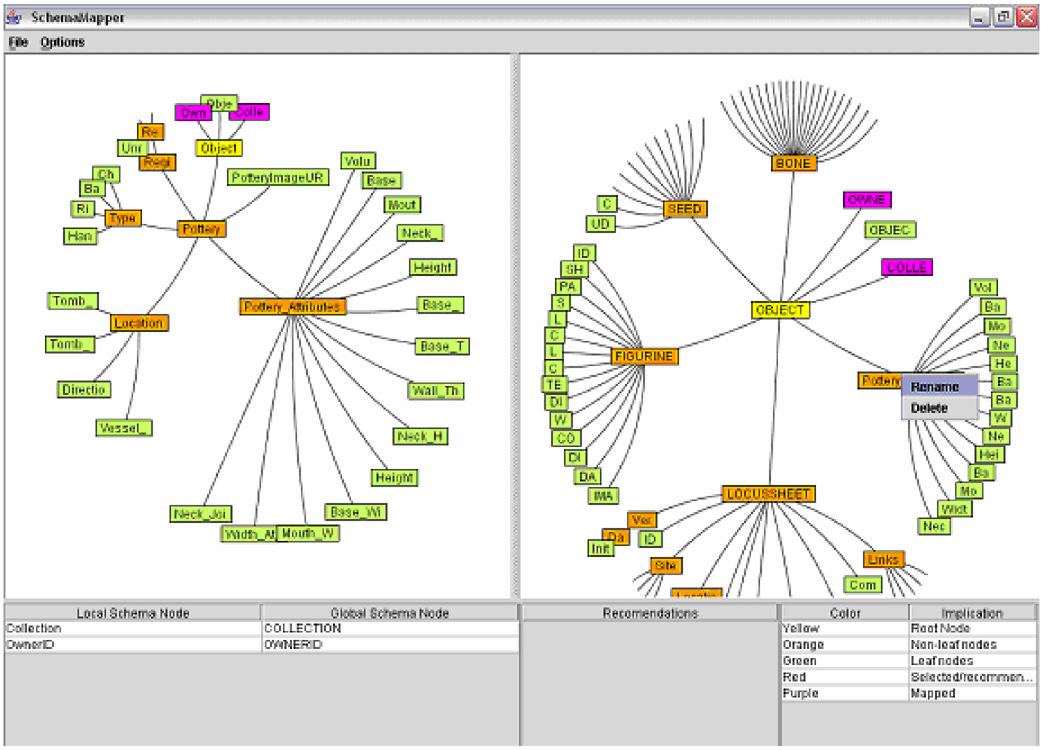
You cannot map these nodes directly, would you like to add as a child?

Local Schema Node	Global Schema Node
Collection	COLLECTION
OwnerID	OWNERID

Recomendations

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 5: Adding a local node as a child of a global node

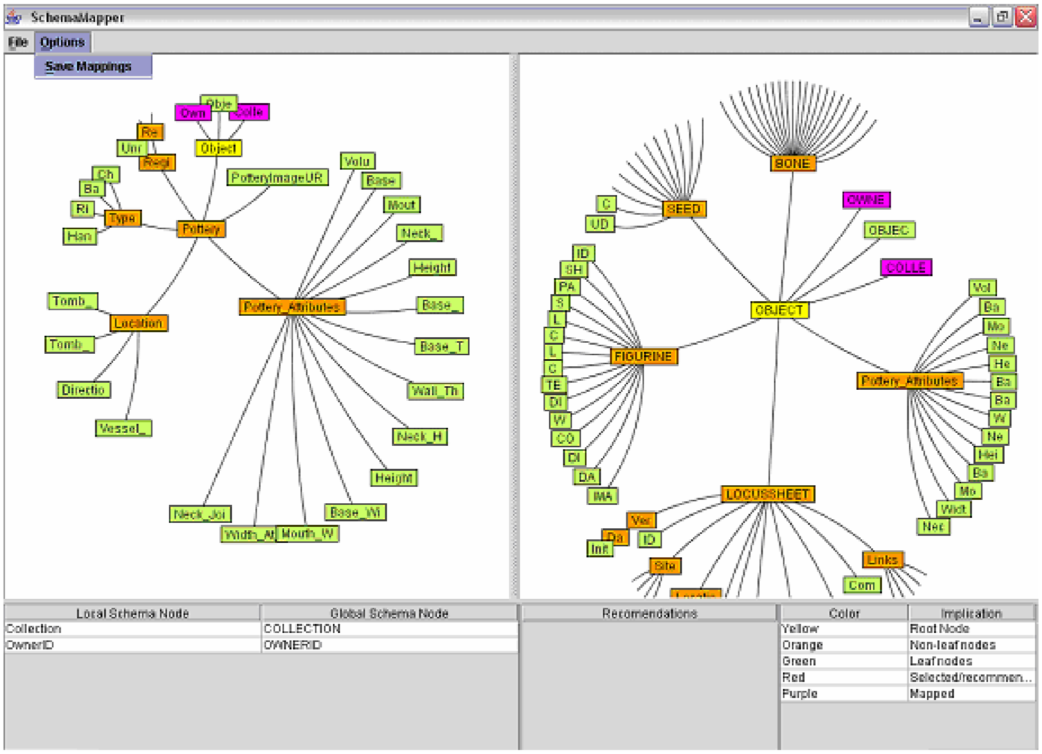


Local Schema Node	Global Schema Node
Collection	COLLECTION
OwnerID	OWNERID

Recomendations

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 6: Renaming / Deleting Nodes



Local Schema Node	Global Schema Node
Collection	COLLECTION
OwnerID	OWNERID

Recomendations

Color	Implication
Yellow	Root Node
Orange	Non-leaf nodes
Green	Leaf nodes
Red	Selected/recommen...
Purple	Mapped

Screen shot 7: Saving mappings

Appendix B: Benchmark Tasks and Survey

Task 1: (Using MapForce)
Open the Pottery local schema (cemetery.xsd) and the ETANA global schema (ETANA 1.1.xsd). Please provide local.xml as the local xml file if prompted for one.

Map the following local schema nodes to appropriate global schema nodes

- ObjectType
- PotteryImageURL
- OwnerID
- Volume
- Collection
- Height

Task 1: (Using Schema Mapper)
Open the Pottery local schema (cemetery.xsd) and the ETANA global schema (ETANA 1.1 xsd).

Map the following local schema nodes to appropriate global schema nodes

- ObjectType
- PotteryImageURL
- OwnerID
- Volume
- Collection
- Height

Question for task 1:
For such kind of tasks involving simple schema mapping do you prefer MapForce or SchemaMapper, why? (Please use the other side of the page if space is insufficient)

Task 2: (Using MapForce)
Open the Pottery local schema (cemetery.xsd) and the ETANA global schema (ETANA 1.1 xsd). Rename node name VOLUME in the global schema to VOL.

Task 2: (Using SchemaMapper)
Open the Pottery local schema (cemetery.xsd) and the ETANA global schema (ETANA 1.1 xsd). Rename node name VOLUME in the global schema to VOL.

Question for task 2:
Do you prefer being able to edit the schema from within the mapping tool or do you prefer having a separate tool for editing purposes.

Task 3: (Only for SchemaMapper)
Open the Pottery local schema (cemetery.xsd) and the old ETANA global schema (etanaGlobalOld.xsd). Add the PotteryAttributes node in the local as a child of the Object node in the Global. Rename PotteryAttributes to POTTERY.
Now map

- ObjectType

- PotteryImageURL
- OwnerID
- Volume

Question for task 3:

We presently support adding local schema nodes as a child, deleting and renaming nodes, do you see any other features that may be required for editing purposes? (Please use other side is insufficient)

General Questions for SchemaMapper:

Do you prefer the approach of SchemaMapper of generating recommendations for mapping? Why or why not?

Did you like the hyperbolic tree representation of the schemas or did you find it confusing? Why or why not?