Morpheus 2.0: A Data Transformation Management System

Pete Dobbins[1], Tiffany Dohzen[2], Christan Grant[1], Joachim Hammer[1], Malachi Jones[1], Dev Oliver[1], Mujde Pamuk[2], Jungmin Shin[1], Mike Stonebraker[2]

[1]CISE Department, University of Florida
PO Box 116120
Gainesville, FL 32611-6120
Phone: (352) 262 - 7383, country code: 001
{pjd, cgrant, jhammer, mjones, doliver, jshin}@cise.ufl.edu

[2]CSAIL, The Stata Center
Massachusetts Institute of Technology
Cambridge, MA 02139
Phone: (603) 714 - 4451, country code: 001
{dohzen, mujde}@mit.edu, stonebraker@csail.mit.edu

ABSTRACT

The authors have previously built the Morpheus Data transformation system. Based on feedback from demo-ing the system at SIGMOD 2006 as well as to numerous CIO's and researchers at IBM and Microsoft, we have completely redesigned the system to facilitate a state-based browsing paradigm, the ability to filter transforms based on lineage and input-output properties and best-fit search for composite transforms. Also included is a novel crawler to search for transforms of interest either within an enterprise or across the web. The result is Morpheus 2.0, which is now operational, and is described in this paper.

1. INTRODUCTION

Information integration has been listed on all four self assessments of the DBMS community [1-4] as an "achilles heel" of computing. Basically, large enterprises have hundreds of operational systems, which are usually constructed by independent groups at different times, and a desire to share information between these systems. This requires integrating a large collection of independently written data base schemas, a task that most enterprises find enormously challenging. The industry thrust toward web services and the internet will increase the scope of this information integration problem from inside a single enterprise (intra-enterprise) to among enterprises (inter-enterprise). This thrust will make information integration that much more daunting.

At the same time, the need for information integration is not limited to industry. The internet is also becoming the preferred method for disseminating scientific data from a variety of disciplines and domains such as astronomy, biology, the geosciences, public health, and health care. The number of independently developed schemas and databases is very large and scientists have been struggling to keep up with this wealth of information. For example, in bioinformatics, the need to access and integrate data from the many and typically large genomics repositories which use a large number of data models, languages, and formats is hampering the discovery of genes and their functions. Given the complexity of the genomics data integration problem, systems such as GUS (Genomics Unified Schema) at the University of Pennsylvania [5], which provides an integrated warehouse for portions of GenBank, EMBL, DDBJ, Swiss-Prot, and dbEST, remain the exception.

This schema integration problem exists when the goal is to integrate information by extracting information from operational systems, transforming it in some sort of middleware ETL system and then loading it in a data warehouse. It exists equally when the goal is to share live information between operational systems through some sort of federated data base system, such as the IBM information integrator [6] or BEA's WebLogic Integration Suite [7].

There are three possible approaches to schema integration, which we explore in the following three subsections.

1.1 Schema Matching

Some researchers (such as [8-10]) have focused on the **schema matching** problem. For example, a positive result from such efforts would be to discover that **wages** in one human resources schema matched **salary** in a second schema. Although such research is well-intentioned, we believe that it only solves a small portion of the overall data integration problem. Specifically, independently constructed schemas **never** have identical data elements. For example, **wages** in the first schema might represent the salary of a French worker, which would be expressed in Euros, net-after taxes and include a lunch allowance. In contrast, **salary** might represent the compensation of a U.S. worker, which would be expressed in U.S dollars, gross-before taxes, while not including a lunch allowance. Hence, syntactically matching the numbers in the two fields, **wages** and **salaries**, will produce garbage, since they represent different semantic objects, respectively U.S compensation and French compensation.

There are several reasons why identical elements do not exist. First, it is rarely, if ever, the case that two schemas use the same representation for the same semantic construct. There are many representations for calendar dates. All of the following are reasonable representations for one of our birthdays:

- `October 11, 1943`
- `10/11/43`
- `11/10/43`
- `Oct. 11 1943`
- `11-10-43`

Obviously, one cannot merely match two attributes, even if they have the same or similar names, because they have different representations. That produces a composite column with “jumble” in it. Instead, one must define a **transform** that will map the individual data elements in one representation to that used by the second schema. Unless an organization has company-wide standards for the representation of common data elements, it will face this issue.

A second more difficult issue occurs when the two attributes do not semantically mean the same thing, even if they have a common representation. For example, “two days” is a reasonable time value that could appear in a time-oriented column in two different schemas. However, it can semantically mean any of the following:

- Two calendar days
- Two business days (excluding weekends and holidays)
- Two Federal Express days (which excludes Sundays)
- Two Wall Street trading days (which excludes weekends and certain other days)
- Two London trading days (which excludes weekends and another collection of days)

Again, a transform is required to map between values in the different schemas to produce a representation with a common meaning.

1.2 Standards

Obviously, a solution to schema integration is to enforce company-wide standards on representation and meaning. Even more obviously, this is a very useful exercise for any company. The less semantic diversity that exists, the easier the data integration problem will be. However, there are several major impediments to the success of standards.

First, this will not help legacy systems, which are already deployed in large numbers. Retrofitting these to obey after-the-fact standards is a major undertaking. Second, successful standards require the enterprise to have the foresight to standardize the right things. For example, a few years ago one of us consulted for a large multi-national bank, which had several autonomous divisions around the world. A recent request from customers was to receive a single integrated world-wide bank statement. Since the institution had not planned for this requirement, there was no single world-wide identifier for a customer. Since a given customer has different legal names in different countries with different addresses, it proved impossible to retrofit this capability, other than by expensive manual human-to-human discussions between bank personnel and customer personnel. A third impediment to standards is they are extremely challenging to implement across enterprises. Not only is there often an inability to co-operate, but also in every vertical market we can think of, there are multiple standards. Hence, there is an issue of which one to choose.

Effective standards appear to come in situations where there is a market elephant which can drive them (for example, between Dell and Walmart and their suppliers) or where there is substantial market advantage to co-operation (for example airline reservation systems). Even when an industry is behind standards, for example RosettaNet[1] in the electronics industry, they have proved elusive.

1.3 Knowledge Representation and the Semantic Web

A third approach to data integration is to construct a rich enough knowledge representation language so that schema elements, such as wages and salaries, can be described precisely, thereby allowing an automatically generated transformer. This has been the goal of knowledge representation languages, such as KIF[2] and KQML[3], and has been pursued (in our opinion with limited success) for at least the last 30 years. The latest incarnation of this avenue of effort is the semantic web [11].

One would be foolish to argue that this line of effort cannot produce results. However, we merely indicate that results have been elusive to date.

1.4 Our Past Experience

Some of the authors began a collaboration following the publication of the most recent data base self-assessment [1]. That report suggested building a testbed of data bases, so that researchers who wanted to explore ideas concerning data integration would not have to go to the effort of assembling data bases to integrate.

We took on the task of building such a testbed, and assembled a collection of 50 schemas that represented data on Computer Science instruction at different universities in several countries. These data bases contained information on courses, instructors, prerequisites, meeting times, credit and the like. Our system, the THALIA information integration testbed[4], is publicly available for use and includes a collection of integration tasks (challenge queries) that must be performed as well as an evaluation framework to score any given tool [8].

The exercise of building THALIA has colored our thinking on data integration extensively. A few of our challenge queries can be solved merely by matching attributes in different schemas, for example, the task of identifying specific course numbers in the various schemas. However, the overwhelming majority of the tasks require dealing with semantically heterogeneous data elements. These include:

- Character strings in different languages (German, English, etc.)
- Course credit (semester units, quarter units, etc.)

[1] http://portal.rosettanet.org/cms/sites/RosettaNet/

[2] logic.stanford.edu/kif/specification.html

[3] www.cs.umbc.edu/kqml/papers/kqmlspec.ps

[4] www.cise.ufl.edu/servlets/thalia.html

- Requirements (sophomore standing is not meaningful in Europe)
- Lab credit (some universities have this concept, some do not)
- Meeting time (many different representations, e.g. period 1, time interval, etc.)

In summary, the vast majority of the THALIA tests required transformations to be written that are well beyond the possible scope of knowledge representation or schema matching. Hence, we believe that the most profitable approach to data integration would be to build a framework and toolkit for transform construction as well as a repository for previously constructed transforms to facilitate reuse of previously written ones. The authors built an initial prototype of such a system, the Morpheus data transformation management system over the last 18 months. In Section 2 we briefly review the capabilities of this initial system. Then, in Section 3 we discuss a collection of major improvements, which collectively are called Mopheus 2.0. Relevant work by others is treated in Section 4, along with some suggestions for additional Morpheus improvements.

2. THE MORPHEUS DATA TRANSFORMATION SYSTEM

Our basic goals for Morpheus 1.0 were two-fold:

1. Make it easy to write transforms
2. Make it easy to find and reuse transforms written by others

To this end, we built a **transform construction tool** (TCT) as well as a **searchable repository** that holds transforms. These two components are integrated into the **Morpheus data transformation management system** described below.

Figure 1 shows an architecture diagram of the Morpheus system. A human interacts with our browser and GUI for building transforms, which are executed and stored inside a Postgres DBMS[5]. We take the point of view that every transform maps a Postgres data type into another Postgres data type. Hence, we store information on data types and transforms in Postgres tables, which can be browsed and modified in the ways described below.

2.1 GUI and Browser

The heart of our approach for reusing transforms is a sophisticated browser that allows a client to explore the transform repository. Our approach borrows heavily from *Squirrel* [12] developed two decades ago. Like Squirrel, Morpheus objects (transforms including their source and target data types) exist in a multidimensional space. A client can enter this space in multiple ways to establish a Morpheus object as his **current focus**. Then, he can browse to nearby objects along any of the supported dimensions. Changing his focus to a new object of interest allows the user to continue browsing, narrowing in on an ultimate object of interest.

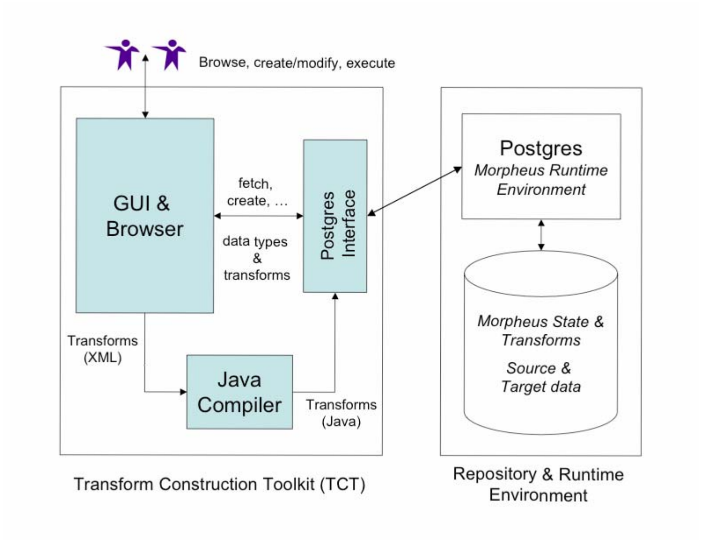

Browse, create/modify, execute
GUI & Browser
fetch, create, …
data types & transforms
Postgres Interface
Transforms (XML)
Java Compiler
Transforms (Java)
Postgres
Morpheus Runtime Environment
Morpheus State & Transforms
Source & Target data
Transform Construction Toolkit (TCT)
Repository & Runtime Environment

Figure 1. Architecture of the Morpheus System

The current Morpheus dimensions include:

- A *classification hierarchy for source and target data types*. Here, "up" (less specificity) and "down" (more specificity) are the two dimensions of browsing movement.
- A *classification hierarchy for transforms*. Again "up" and "down" are well defined.
- *Focus*. If the current focus is a transform, the client can move to the source or target data type. If the current focus is a data type, the client can browse the transforms that accept or produce the data type.
- *Textual similarity*. Similarity of textual description is another dimension in which browsing is supported. Similarity searching is supported in our prototype by the Lucene text search engine[6], which is integrated with Postgres. This engine supports exact matches as well as similarity-based matches on any of the meta-data stored for Morpheus objects, including the textual description.

Using the Morpheus browser, a client enters the repository by starting at the root of a classification hierarchy or by issuing a keyword-oriented search. The result is a list of one or more data types or transforms from which the user can select the current focus for subsequent searching.

If a user clicks on a category, the Morpheus objects, which are nearby in that category, appear on the screen. A user can click on an object to change his current focus, and the cycle repeats. Hence, the client browses by moving his current focus in any supported dimension, thereby moving to an object of interest.

Once an interesting object is identified, Morpheus allows the client to zoom into the object. Increasing amounts of detail are given, ultimately providing the visual representation for the transform or complete information on a data type.

[5] www.postgresql.org/

[6] lucene.apache.org/

2.2 TCT

In our opinion requiring a user to drop into a conventional programming language, such as Java or C++, to construct transforms, will automatically make the job of building transformations very tedious. Even simple transforms require tens to hundreds of lines of code in a conventional programming language.

Instead, we have assembled a palette of high level building blocks (workflow primitives) that can be wired together in a composite workflow to accomplish a given transformation task. We have built a compiler for this workflow language, TCT, to generate an efficient executable representation.

The workflow primitives in the current Morpheus prototype include the following:

- A *computation* primitive, which performs standard numeric transformations (e.g., metric conversions).
- A *control* primitive, which supports branching within the transform based on the value of a variable or result of an expression.
- A *lookup table* primitive, which maps an object in one field to an object in another field (e.g., letter grade to numeric grade).
- A *macro* primitive (superbox), which allows a user to group several boxes together (inside a larger box) as part of a high-level transform (macro). This is also a way to hide some of the details of a transformation.
- *Postgres user-defined functions*, so transforms can be composed out of existing ones.
- A *predicate* primitive, which maps sets (ranges) of values to related sets (e.g., University class status such as freshman to units taken, '> 0', '>35', and so on) based on a user-defined predicate.
- A *re-arranger*, which supports creation of transforms by allowing the user to visually manipulate characters and strings.
- A *wrapper* primitive for external call-outs (e.g., to Web services).

The current prototype allows a user to start either with a blank canvas or with an existing transformation from the repository. He can then edit the workspace, adding, deleting and rewiring objects of interest. When he is satisfied, he can register the transform in the Morpheus repository. In addition, there are facilities for testing transforms on individual input data elements, entered from the GUI or on tables of objects stored in Postgres.

Transforms are stored in two representations. The first is an XML data structure that corresponds to the diagram. This representation is chosen since it is easy to edit. The second representation is a Java function, which is efficient to execute. However, there is no requirement that transforms be written using the TCT. Java code can be written, compiled outside of the system, directly entered into the data base, and then incorporated into a Morpheus transform. Moreover, we expect many transforms to be available as web services.

2.3 Searchable Repository

All Morpheus information is stored in Postgres tables and our browser is thereby data base-oriented. Storing transform and data type information in a database requires a schema, and the construction of the Morpheus schema is a conventional data base design problem. The metadata we currently support includes:

- Who constructed the transform
- When it was constructed
- What data type it accepts as input
- What data type it produces as output
- A textual description of the semantics of the transform
- Position in a classification hierarchy

In addition, source and target data types are registered as Postgres data types, and transforms become user-defined Postgres functions. In this way, conversions occur inside Postgres by adding data to the data base and then running a query which invokes a transform.

If the data types of the transform have not been registered in Postgres, this step must be performed first. Next, the transform is registered as a user-defined function. In order to execute a transform a client provides one or more data elements of the source data type. With this input, a query is executed to produce the desired target result, which consists of one or more data elements of the target data type.

The reason to leverage Postgres is to simplify the run-time environment, which is all inside Postgres, as well as to use Postgres query and storage facilities, in case large amounts of data must be transformed in bulk. Over time, we expect Postgres to support horizontal partitioning, which will allow bulk transformations to be performed in parallel on multiple machines for added performance.

Our approach to data transformation has points of similarity with existing ETL tools[7] in that we provide a high-level transform tool. However, we *differ* from ETL vendors and existing transformation tools in several important ways:

1. We provide a searchable repository in which transforms are stored, along with powerful repository browsing tools allowing our users to search for existing transformations along several dimensions. Our idea is that a client with a transformation problem will browse our repository looking for a particular transform or something close, which he can modify. In the latter case, he would add his modified transform to the repository so others can benefit from his effort.
2. In addition our repository is constructed to be language neutral. Hence, transforms can be written in any programming language or be web sites, wrapped as web services.

[7] For example, Altova's *Mapforce* (http://www.altova.com/products/mapforce/data_mapping.html), Itemfield's *ContentMaster* (http://www.itemfield.com/products/overview1.aspx) or Informatica's PowerCenter (http://www.informatica.com)

3. We are leveraging the Postgres DBMS both as the repository and as a platform to execute the transformations. Using this technology, we can perform bulk transformations inside the DBMS where transaction management and powerful queries can be leveraged. This approach is in contrast to current ETL tools, which are external to databases.

3. MORPHEUS 2.0

We have initial experience with using the Morpheus system ourselves and have given several demos to persons with real data integration problems (mostly CIOs of Boston area companies). In addition, we have demoed the system at SIGMOD 2006. This experience has motivated a significant redesign, which is now operational. Salient points of Morpheus 2.0 are discussed in this section.

3.1 Crawler

One CIO of a large conglomerate reacted to the Morpheus demo by saying "this tool would be very helpful, if all of our existing transforms were in the repository. However, I don't see an easy way to justify the effort of manually finding, documenting and registering them".

To deal with this issue, we have built a crawler, which can look for transforms either inside an enterprise or across the public internet. On the public web the crawler looks for web sites with form-oriented interfaces and with trim on the web page that is indicative of a web service. The crawler outputs such web pages to a human, who assembles the required Morpheus metadata for the service. Such web services expect an instance of an input data type and respond with an instance of an output data type. As such, a web service is exactly an element of our semantic transform architecture, albeit one that uses another language for implementing the transform on a remote site. Hence, they can be wrapped by our existing Morpheus callouts for Postgres.

In addition, either within the enterprise or across the web, our crawler can look for source code files in a specific language, right now either Java or Web Service Description Language (WSDL). In WSDL mode, the crawler parses each acquired file with a .wsdl extension looking for interface information describing all publicly available functions, data type information for all message requests and message responses, address information for locating the web service and binding information about the transport protocol to be used. With this information, a connection to the web service can be established through SOAP[8] and execution of the available functions can be performed. Such execution information can help a human to evaluate the usefulness of the service and determine whether it should be registered in the Morpheus repository.

In Java mode our crawler looks for .java source code files. The crawler parses the input and output data type specifications, and looks for comments in the code that indicate the purpose of the routine. It also looks for documentation and read_me files. The file is downloaded and compilation is attempted. If successful, each method is identified as a candidate registration with Morpheus. As with the WSDL mode, the user has the ability to dynamically call each method and inspect the values returned. All of this information can be used by a human in the Morpheus registration process.

[8] http://www.w3.org/TR/soap/

When looking for form-oriented web services, the crawler parses HTML for key elements that identify the page as a form. Such elements are the form tag, entry fields (text boxes, list boxes, etc) and selection fields (submit or search buttons, selection menus, etc). Once a form has been identified, the keywords of the page (meta data, form name, button names, etc), the input data type and output data type are collected. In addition, sample data is submitted to the form, and (input, output) pairs generated. As with candidate Java and WSDL web services, this information is presented to a human to more quickly ascertain usefulness and to facilitate resulting registration.

Currently, our crawler requires a user to manually place each web service in a classification hierarchy. However, we have begun automating this process by examining the documentation field in the WSDL file and performing a keyword-oriented relevance based ranking to classification hierarchy elements. We will also investigate adding a web service to multiple categories if matches in each category reach a user-defined threshold.

At this time, the crawler is operational, but only searching and locating WSDL objects. The implementation of identifying forms and Java files will be completed soon. We have tested the current version of the crawler on the public internet for the past week. The examination of 225,000 URLs yielded only 235 WSDL objects. Of these, 220 were useful web services. As WSDL objects were found, our human cataloger required an average of 30 seconds to reject an uninteresting web service and 3 minutes to register a useful one. We are continuing to crawl for WSDL objects and anticipate that Java and form-oriented searches will provide a much greater number of useful web services.

We are about to try our enterprise crawler inside the firewall of a major corporation. Included in this trial will be our implementation of identifying useful forms and Java files. Our results on both the public and private networks will be available in the final paper.

3.2 Data as a Browsing Dimension

A frequently requested capability is to locate transforms in our repository from specification of their input-output properties. For example, a user might know that the value 27.3 can be accepted as input and maps to an output value between 62 and 64. As such, users would like to find transforms based on their input/output characteristics. In Morpheus 2.0 the (input, output) pairs are recorded in Postgres every time a transform is executed, along with the running time of the transform, which is used by the TransformScout to be discussed in Section 3.4.

To facilitate browsing by data values, we have provided a screen whereby the user can enter into a form a list of values or value ranges for either input or output. The form is transformed into a Postgres table, and a series of queries are run to find the transforms that fit the data.

Although we could run a single query to perform the required work, it is prohibitively expensive. Hence, we first eliminate the transforms that fail to match the users input or output data types. Second, we ensure that all tables are sorted in value order of the input data types. Having each value or value range in the user's specification, allows us to quickly discard transforms that do not have a recorded input value in the appropriate range. For ones which qualify, we can then match the output values quickly.

It is entirely possible that there is no recorded data that matches the user request. Hence, we also allow the user to put Morpheus in data collection mode. Here, Morpheus automatically executes potential transforms with artificial data in the range specified by the user request, and examines the output data for a match. Although slower than the default mode, this allows Morpheus to check all transforms, rather than just those that have previously recorded (input, output) pairs in the required range.

Lastly, Morpheus can also perform the automatic execution noted above in background mode. In this case (input, output) pairs based on artificial data are collected using otherwise idle cycles. Our current algorithm runs round-robin to ensure that the number of (input, output) pairs for all transforms is approximately the same. Clearly, a more sophisticated algorithm is possible.

3.3 Search by Lineage

One of the common themes that we have observed from industry interviews is a desire to incorporate lineage information into Morpheus. This reflects a desire to find transforms that have been derived from a "seed" transform by one or more modifications. To support this sort of search, Morpheus notes when a user modifies a transformation into a new one, and adds a record to a Lineage table indicating that the revised one is a direct **descendent** of the original one. Also, since a user can incorporate multiple existing transformations into a new one, the derivation (lineage) history stored in the Lineage table is a graph.

We have converted Morpheus to allow browsing and searching on this graph. A user can search the graph for all transforms that are descendents or ancestors of a given transform and within a specific graph distance. We are exploring more sophisticated metrics to compute lineage distance that are based on other information. We are especially interested in distance metrics that depend on the degree of code similarity between two related transforms. Hence, distance would be smaller for a transform that had made fewer changes to a transform than one that had modified it more extensively.

Notice that much of the literature on lineage has focused on data lineage, for example Trio [13]. In contrast, Morpheus deals with code lineage. Hence, the information stored and the queries that deal with lineage in the two cases are quite different.

3.4 Search for Composite Transform

A fourth capability requested by many of the people who have seen Morpheus was to specify an *input* data type and an *output* data type and have Morpheus find either:

1. A ranked ordering of composite transforms that map *input* to *output* through some collection of intermediate data types.
2. A ranked order of "partial" collections of transforms that map from *input* to *output* through a collection of intermediate nodes as above. However, one or more of the transforms along this path has not been defined. Hence, the user can get the transform he wants by a composition of existing transforms if he defines the missing one(s).

We have implemented a facility called TransformScout (TS), which accomplishes these two capabilities. Basically, TS searches a graph whose nodes are data types and whose edges are existing transforms. Each edge in this graph is marked with a cost function of the average running time of the associated transform. The first feature is supported by finding all connected paths from a given input data type to an output data type. This is a standard graph search problem on a graph which may have thousands of nodes, for which we are using an A* algorithm.

Both running time and memory usage can become problematic using A*. In the worst case, both are exponential in the number of nodes in the graph. We use a memory bounded variant of A* to limit complexity, heuristically discarding partial paths when necessary using the following functions. Any given partial path, X, has a priority determined by:

- F(X), the sum of cost functions for each edge in the path. This is a proxy for the cost of the path.
- H(X), a heuristic estimate of the minimal cost to reach the target data type from X. This estimate is based on the similarity between the data type at the end of path X and the target data type.

If multiple complete paths are found, we first compose stored (input, output) pairs to ascertain if the multiple paths are semantically identical. If none exist, then we execute the functions to produce (input, output) pairs that can be composed.

The result is a collection of semantically distinct composite transforms. Each composite transform may have multiple implementations (paths from input to output) which Morpheus must rank. This is accomplished by a metric which takes into account computational performance and software quality. The first portion of the metric is obtainable using F(x) above. The second is supported by allowing users to review a transform and indicate their quality rating, as well as report bugs. Searching for an incomplete path uses the same logic discussed for connected paths. There are simply more paths which must be considered.

We are in the process of evaluating our algorithms on a collection of about 200 data types concerning US government entities (e.g. Social Security Administration, Department of Education, Amtrak, Medicaid, and Internal Revenue Service). For these entities, we have wrapped about 100 functions that are publicly available on the web and registered them as transforms in Morpheus. For example, consider two data types in this collection:

- SSA (Social Security Administration) profile data type. This data type contains, among other things, the history of contributions to date.

- RBE (Retirement Break Even) age; i.e., the age at which the lifetime sum of future benefits is maximized.

RBE is calculated by statistically computing the average age of death of the individual, and computing the current value of his/her lifetime payout, and then finding the retirement year in which this number is maximized.

There are two transforms which operate on the SSA data type and produce benefits information, a Quick Benefit Calculator and Detailed Benefit Calculator[9]. There is only one transform in the system that produces Retirement Break Even Age, but it operates on an intermediate data type.

Hence, besides writing the complete transform from scratch, there are five partial paths which use some of the available transforms. TransformScout produces a ranked list of these 6 alternatives.

We plan experiments on a variety of these kinds of problems to compare the answers generated by our tool with those generated by a knowledgeable human. Results of these experiments will be available by September.

3.5 Search of the Classification Hierarchy

Using the DMOZ[10] hierarchy, a subset of the semantic web, we have built a taxonomy of general and domain-specific terms that describe the semantics of transforms and data types. Our implementation leverages the DMOZ structure since we have a large source of potentially uncategorized information which needs to be linked in a user modifiable hierarchy. For every new transform or data type that is entered into the repository, the user selects the terms in the taxonomy relevant to the new item. The terms chosen are then used as labels. The resulting category browser provides a compact search structure that complements the more rudimentary string matching using textual descriptions which Morpheus also supports. The DMOZ hierarchy also gives us the benefit of using a framework which is under active extension.

In our approach, when the search begins, the user is presented with the root level of the category list. After the user selects a category, the subcategories under this category expand into view. As additional categories are selected, the process continues and the user moves deeper into the hierarchy. When a category is selected, the associated transforms and data types are displayed within the Morpheus browsing model. The user may also extend the DMOZ taxonomy with his own categories by adding new nodes.

Our DMOZ hierarchy is the initial perspective presented to the user. However, Morpheus is extendible, so additional classification hierarchies can be added. This merely requires uploading a file containing the user-defined taxonomy. Additional taxonomies are stored in separate Postgres tables.

Moreover, the user can dynamically switch hierarchies. If the current focus is a transform or data type, then the display is redrawn showing the location of the specified object in the new hierarchy. If the current focus is a descriptive term in the hierarchy, then the search must start at the root node of the new hierarchy.

3.6 Browsing Model

Our current browsing model allows a user to “wander” the repository in multiple dimensions. However, this model does not correspond with what users have indicated they want. Users typically know various pieces of information about the transform they would like to find. For example, they might know it was written by someone in department 27, uses a lookup table, and maps 27.3 into a number between 62 and 64.

This search requirement is best supported by allowing a user to specify search criteria in multiple dimensions at once. Such multi-faceted search is very different from the multi-dimensional browsing supported in Morpheus 1.0. In addition, the user may have several multi-faceted searches open at one time. If a search yields too many candidates, he may want to interactively refine the search. Hence, *result sets* of previous queries must be remembered so that subsequent refinement is possible. Moreover, we support union and intersection on results sets, so that they can be combined together, where desirable.

This capability is foreign to SQL, which has no notion of refining results, but it is a popular information retrieval paradigm. Since every search in Morpheus turns into some Postgres query, we simply remember the query that corresponds to every result set and then AND on extra predicates to refine the result.

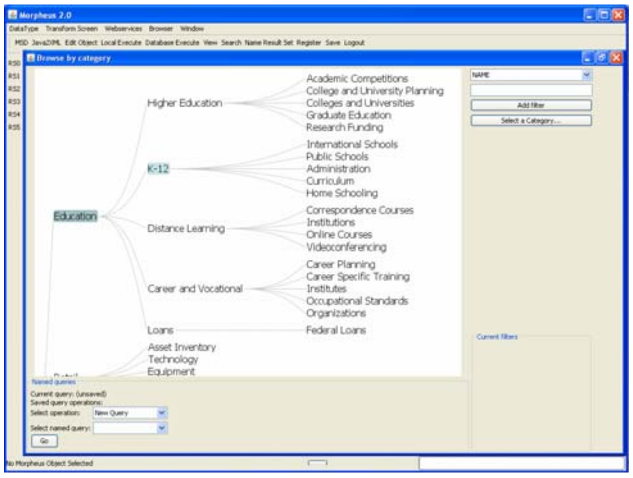
A simplistic model would deal with each new predicate with equal weight. However, it is likely that some refinement dimensions are more important than others, and a relevance concept should be introduced to model the differences in weight between the various dimensions.

3.7 Putting It All Together

We have converted the GUI in Morpheus to support the notion of result sets and multidimensional filtering. To begin his search, the user specifies a current focus, which is either empty or a result set, as noted in the lower left-hand window of the search screen in Figure 2. Then, he specifies the dimension in which he desires to add filtering in the upper right-hand window of the screen shown in Figure 2. Since different search dimensions may have radically different types of visualizations, the upper left-hand window is overlaid with a dimension-specific window. In Figure 2 we show the screen for navigating the category hierarchy.

[9] www.ssa.gov/

[10] www.dmoz.org/



NAME

Current query: (unsaved)
Saved query operations:
Select operation: New Query
Select named query:

No Morpheus Object Selected

Figure 2. The Root Search Screen

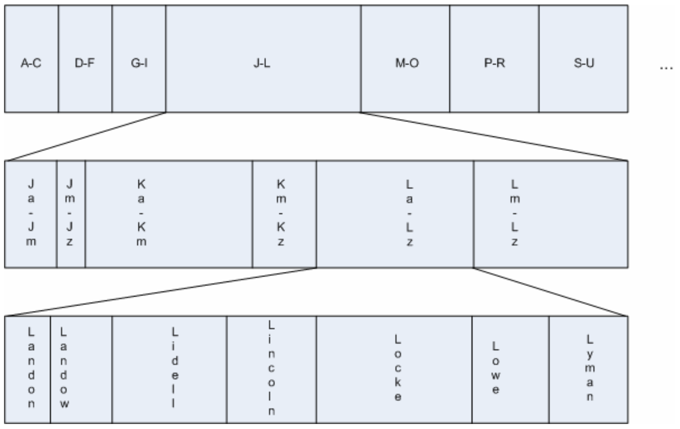


Figure 3. The Author Browse/Search Visualization

There is a "Google style" panel for keyword search of the meta data, a hierarchy browser for searching the classification hierarchy, a form-oriented text system to enter predicates on specific fields, a data panel to enter (input, output) pairs for data search, and a panel for composite search. We also plan but have not yet implemented multi-faceted GUI panels that allow filters on multiple dimensions to be specified at once. The visualization to support browsing and filtering by author name is shown in Figure 3.

In this way, a user can refine or expand any result set by altering the predicates used to construct it. When he clicks "run", the new result set is materialized, which he can either discard or register as a new result set.

4. RELEVANT PREVIOUS WORK

Information integration has been widely studied over the last quarter century from various points of view. In fact, the search engine Google has 104,000 citations on the topic of federating disparate databases. Many companies have built distributed data bases, object-oriented data bases, gateways, and/or adaptors and thereby have offerings in this area. In the 1990's, Cohera extended this support to include user-defined functions to specify mappings from local schemas to a global schema. IBM's distributed data system, Data Joiner [14], now called the Information Integrator [6], contains the same sort of capabilities, which were first explored in the TSIMMIS [15] and Garlic [16] projects.

There are numerous tools in the Extract-Transform-Load (e.g., Informatica's Power Center[11], Ascential's Data Stage[12], Visual Importer by DB Software Laboratory[13]) and data migration space (e.g., DTM Migration Kit by SQL Edit[14], SQLWays by Ispirer Systems[15]), which support the transformation of data from one representation to another. However, none of these tools are currently capable of supporting management and reuse of transformations as proposed in Morpheus.

Also, the research community has been incredibly active on this topic with efforts ranging from view integration [17], sharing architectures [18], sharing languages [19] including multi-source query processing [20], schema matching [21], data translation [22] and data cleansing [23]. Hence, in this section we can only survey a portion of this space at a very high level.

Table 1 indicates our view of the various approaches to information integration. Here, we see a standard "quad chart" with columns for text and data integration. The corresponding rows indicate whether the goal is to find matching information or to transform source information into target information.

Table 1. Information Integration Approaches

	Text	**Data**
transform	LANGUAGE TRANSLATION	OUR APPROACH
match	TAXONOMIES, ONTOLOGIES	SCHEMA MATCHING

The lower left hand field deals with text matching, for example determining that "rubber gloves" mean the same thing as "latex hand protectors". There has been considerable work on ontologies and taxonomies that address this issue [8, 24]. In fact, some argue that the semantic web [11] is largely aimed at this box.

In the lower right hand field, there has been substantial work in performing schema matching [9, 25], i.e., determining that the attribute "wages" in your schema matches the attribute "salary" in my schema. Given the range of differences that exist between the schemas, identifying the mappings has for the most part been a time-consuming and mostly manual task. Some research has been done on automating this task, for example GLUE [26], LSD [27], IMAP [28] and Corpus-Based Matching [29]. These projects have focused on developing techniques, frequently based on machine learning algorithms, for identifying semantic mappings with minimal human involvement. In contrast, Morpheus provides a powerful toolkit whereby a human can complete the task much more efficiently.

In the upper left hand field appear technologies such as language translation. In the THALIA project [30], which

[11] www.informatica.com/products/powercenter/default.htm
[12] www.ascential.com/products/datastage.html
[13] www.sqledit.com/mk/
[14] www.sqledit.com/mk/
[15] www.ispirer.com/products/

assembled a test bed of more than 50 schemas from Computer Science departments around the world dealing with courses, instructors, meeting times, prerequisites, and so on, the researchers found that text fields were inevitably in the native language spoken in the country where each university was located. Hence, to find courses on databases in the USA or Germany, one had to translate German course descriptions into English or to translate “Datenbanken” into “database”.

Lastly, the upper right hand field deals with data transformations. For example, a salary in a French data base would be expressed as an after tax quantity in Euros and would include a lunch allowance. The corresponding salary in a USA database would be gross, before taxes, and would not include a lunch allowance. A fairly complex transformation is required to one or both of these objects to make them comparable in a data warehouse or a federated information system. Historically the purpose of extract, transform, and load (ETL) systems, such as Informatica and Ascential (now owned by IBM) was to address the upper right hand field.

The purpose of the *Morpheus* project is to address the upper right hand field better than ETL systems. As such, it complements the technologies being developed to address other boxes in Table 1. Some projects attempt to build transformations automatically, for example, the context mediation approach introduced in [31]. This approach will work for fairly simple transformations, for example currency conversions, but is unlikely to succeed in more complex cases, such as the salary conversion problem noted above. Our approach, in contrast, is to provide tools to assist a human in constructing and reusing transformations.

Although information integration has been widely addressed, we believe that it is far from solved. For example, we know of no system that can score well on the THALIA benchmark and the continuous stream of war stories about the integration challenges in industry are further testament to the inadequacies of current solutions.

5. CONCLUSIONS AND FUTURE WORK

This paper has presented the main constructs in Morpheus 2.0. At a high level, there are three main contributions enhancing the original system.

First, the implementation of a crawler to assist in transform discovery, either over the web or inside an enterprise. Currently, Morpheus 2.0 performs automatic discovery and manual registration. We are working toward more automatic registration.

Second, a search paradigm, based on result sets and query refinement has been created to replace the browsing-oriented paradigm in Morpheus 1.0. This allows a user to have a focus which is a collection of records, rather than the previous scheme which supported moving around in N-dimensional space.

Lastly, new search primitives are incorporated. These include filtering on transform lineage, filtering on input-output characteristics, and the ability to find composite transforms that fully or partially solve the user’s problem.

We expect to carefully evaluate Morpheus 2.0 with real world users. Without a doubt, there will be a Morpheus 3.0 to address shortcomings discovered. We also expect to move ahead aggressively with automatic discovery and registration of transforms, since the amount of manual effort required currently is an impediment to adopting the Morpheus approach.

6. REFERENCES

[1] S. Abiteboul, R. Agrawal, P. A. Bernstein, M. J. Carey, S. Ceri, W. B. Croft, D. J. DeWitt, M. J. Franklin, H. Garcia-Molina, D. Gawlick, J. Gray, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, M. L. Kersten, M. J. Pazzani, M. Lesk, D. Maier, J. F. Naughton, H.-J. Schek, T. K. Sellis, A. Silberschatz, M. Stonebraker, R. T. Snodgrass, J. D. Ullman, G. Weikum, J. Widom, and S. B. Zdonik, "The LOWELL Database Research Self Assessment," *The Computing Research Repository (CoRR)*, vol. cs.DB/0310006, 2003.

[2] P. A. Bernstein, U. Dayal, D. J. DeWitt, D. Gawlick, J. Gray, M. Jarke, B. G. Lindsay, P. C. Lockemann, D. Maier, E. J. Neuhold, A. Reuter, L. A. Rowe, H. J. Schek, J. W. Schmidt, M. Schrefl, and M. Stonebraker, "Future Directions in DBMS Research - The Laguna Beach Participants," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 18, pp. 17-26, 1989.

[3] P. A. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman, "The Asilomar Report on Database Research," *SIGMOD Record*, vol. 27, pp. 74-80, 1998.

[4] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database Systems: Achievements and Opportunities," *Communications of the ACM*, vol. 34, pp. 110-120, 1991.

[5] S. Davidson, J. Crabtree, B. Brunk, J. Schug, V. Tannen, C. Overton, and C. Stoeckert, "K2/Kleisli and GUS: Experiments in integrated access to genomic data sources," *IBM Systems Journal*, vol. 40, pp. 512-531, 2001.

[6] IBM Corp., "Using the Federated Database Technology of IBM DB2 Information Integrator," IBM White Paper GC18-9066-00, October 2003.

[7] I. BEA Systems, "WebLogic Integration."

[8] Y. An, A. Borgida, and J. Mylopoulos, "Constructing Complex Semantic Mappings Between XML Data and Ontologies," in *International Semantic Web Conference*. Galway, Ireland: Springer, 2005.

[9] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy, "Corpus-based schema matching," in *21st International Conference on Data Engineering (ICDE)*. Tokyo, Japan, 2005.

[10] O. Topsakal and J. Hammer, "Schema Matching By Analyzing Application Source Code with Heuristics," in *IEEE International Conference on Information Reuse and Integration*. Waikoloa, Hawaii: IEEE, 2006.

[11] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, vol. 2001, 2001.

[12] R. G. Cattell, "Design and Implementation of a Relationship-Entity-Datum Data Model," Xerox PARC, Palo Alto, CA, Xerox PARC Technical Report CSL 83-4, March 1983.

[13] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A System for Data, Uncertainty, and Lineage," in *International Conference on Very Large Databases*. Seoul, Korea, 2006, pp. 1151-1154.

[14] P. G. a. E. T. Lin., "Datajoiner: A practical approach to multidatabase access," presented at Intl. IEEE Conf. on Parallel and Distributed Information Systems, Austin, TX, USA, 1994.

[15] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS," presented at AAAI Symposium on Information Gathering, Stanford, CA, 1995.

[16] V. Josifovski, P. Schwarz, L. Haas, and E. Lin, "Garlic: A New Flavor of Federated Query Processing for DB2," presented at SIGMOD 2002, Madison, WI, USA, 2002.

[17] A. P. Sheth, J. A. Larson, and E. Watkins, "TAILOR, A Tool for Updating Views," presented at International Conference on Extending Database Technology: Advances in Database Technology, 1988.

[18] A. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, vol. 22, pp. 183-236, 1990.

[19] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian, "SchemaSQL - A Language for Interoperability in Relational Multi-database Systems," presented at Twenty-Second International Conference on Very Large Databases, Mumbai, India, 1996.

[20] J. D. Ullman, "Information Integration Using Logical Views," presented at International Conference on Database Theory, 1997.

[21] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB Journal: Very Large Data Bases*, vol. 10, pp. 334-350, 2001.

[22] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman, "A Query Translation Scheme for Rapid Implementation of Wrappers," presented at Fourth International Conference on Deductive and Object-Oriented Databases, Singapore, 1995.

[23] E. Rahm and H. H. Do, "Data Cleaning: Problems and Current Approaches," *IEEE Data Engineering Bulletin*, vol. 23, 2000.

[24] J. Davies, D. Fensel, and F. v. Harmelen, *Towards the Semantic Web – Ontology-Driven Knowledge Management*: Wiley, 2002.

[25] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos, "Imap: Discovering complex mappings between database schemas," in *ACM SIGMOD International Conference on Management of Data*. Paris, France: ACM, 2004.

[26] A. Doan, "Learning to Map between Structured Representations of Data," in *CS Dept.* Seattle, WA: University of Washington, 2002.

[27] A. Doan, P. Domingos, and A. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach," presented at ACM SIGMOD Conference on Management of Data (SIGMOD'2001), Santa Barbara, USA, 2001.

[28] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos, "IMAP: Discovering Complex Mappings between Database Schemas," presented at ACM SIGMOD International Conference on Management of Data, Paris, France, 2004.

[29] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy, "Corpus-based Schema Matching," presented at 21st International Conference on Data Engineering (ICDE), Tokyo, Japan, 2005.

[30] J. Hammer, O. Topsakal, and M. Stonebraker, "THALIA: Test Harness for the Assessment of Legacy Information Integration Approaches," in *21st International Conference on Data Engineering (ICDE)*. Tokyo, Japan: IEEE, 2005, pp. 485-486.

[31] T. Gannon, S. Madnick, A. Moulton, M. Siegel, M. Sabbouh, and H. Zhu, "Semantic Information Integration in the Large: Adaptability, Extensibility, and Scalability of the Context Mediation Approach," MIT, Cambridge, MA, Research Report CISL# 2005-04, May 2005.