

Enabling Information Integration and Workflows in a Grid Environment with Automatic Wrapper Generation

Xuan Zhang Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{zhangx, agrawal}@cse.ohio-state.edu

ABSTRACT

With a growing trend towards grid-based data repositories and data analysis services, scientific data analysis often involves accessing multiple data sources, and analyzing the data using a variety of analysis programs. One critical challenge in this, however, is that data sources often hold the same type of data in a number of different formats, and also, the formats expected and generated by various data analysis services are often distinct.

We believe that the traditional approach for dealing with this problem, which is using hand-written *wrappers*, is not an effective and scalable solution for a grid environment. This paper presents a new approach, which involves generating wrappers automatically for enabling grid-based information integration and workflows. In this approach, a layout descriptor is used for describing the data format for each data source, as well as the input and output format for each tool or service. Efficient wrappers are then generated automatically for translation between any two data formats. Our design separates wrapper generation service from the wrapper execution. The wrapper generation service analyzes the layout descriptors and generates a WRAPINFO data structure. The wrapper comprises a set of application independent modules which take the WRAPINFO data structure as the input.

We demonstrate our wrapper generation tool with two real case studies. Besides showing the effectiveness of our system, the experiments results from these two case studies show that the wrapper generation overhead is very small, automatically generated wrappers scale well to large datasets, and for the one case where this comparison was possible, the execution time of our wrapper was within 30% of that of a hand-written one.

1. INTRODUCTION

Analysis of large and/or geographically distributed scientific datasets [8] has emerged as a key component of grid computing. Many projects have been working on technologies to support *data grids*. Most popular directions have been replica services [6, 9], reliable and predictable data transfers [2, 36], and constructing workflows [1, 11]. Metadata cataloging and metadata services have also received much attention lately [12, 33].

In a data grid, data from multiple sources may have to be analyzed using a variety of analysis tools or services. This can introduce a number of challenges. In recent years, several research groups have initiated work addressing some of these challenges. For examples, Chimera is a system for supporting virtual data views and demand-driven data derivation [15, 40]. Similarly, CoDIMS-G is a system providing grid services for data and program integration [14]. A number of projects have focused on scientific workflows. The workflow management research group under Global Grid Forum's Scheduling and Resource Management area is active in this area, and has compiled a list of existing projects in this area¹.

One of the important challenges in this area is that data formats or layouts used by different data sources and expected by different data analysis tools can vary significantly. We consider one example from the bioinformatics domain, where a number of datasets and

analysis tools are available for researchers. Biologists frequently want to run BLAST search on SWISSPROT databases. However, differences in dataset layouts forbid them from doing so directly, as BLAST asks for sequences to be stored in FASTA format, and SWISSPROT data is stored in a different and much more complicated form. The common way of addressing this problem has been through hand-written *wrappers*. The function of a wrapper program is to transform the data from one source into a format that is understandable by another. Wrappers have been used in workflow systems, in systems like Chimera and CoDIMS-G that we mentioned earlier, and more traditionally, as part of database integration or mediator systems [17, 32, 18, 38].

In this paper, we present an approach for automatic wrapper generation suitable for a data grid environment. We believe that hand-written wrappers are not practical and scalable in a grid environment for the following reasons:

- To achieve interoperability between N data formats, an order of $O(N^2)$ wrappers have to be written. A single update in a data format will involve rewriting of $O(N)$ wrappers. Thus, hand-written wrappers are not scalable with respect to the number of available resources, because of the high programming and maintenance effort involved.
- In the vision of grid-based or web-based computing, it is desirable to discover data sources and available data analysis services dynamically. In such a scenario, any approach that requires a wrapper written specifically for a given data source or a given data processing service is not going to be practical.
- For execution in a grid environment, a wrapper either needs to be a grid or web-service, which usually means slower execution, or needs to be ported on a variety of hardware and software platforms, which could be very time consuming.

Our approach involves generating wrappers automatically starting from declarative annotation of data layout formats associated with each source. Our work particularly focuses on datasets available as flat-files, and analysis programs which expect text strings as input. The descriptor we use for annotation is similar in flavor to the Data Format Definition Language being developed by the DFDL Working Group in the Global Grid Forum². Such descriptions provide sufficient information for the system to understand the layout of binary or character flat-files, without relying on any domain- or format-specific information. Based on this information, our analysis module generates an XML data structure which we refer to as the WRAPINFO data structure. Using this data structure, a set of application independent modules we have implemented carry out the data transformation tasks.

Overall, our approach offers the following advantages:

- For each resource, only one layout descriptor needs to be written, irrespective of any other resources it may be integrated with. Moreover, as new data sources or tools are pub-

¹Please see <http://www.isi.edu/~deelman/wfm-rg>

²Please see <http://forge.gridforum.org/projects/dfdl-gw>

lished, or existing ones move to new formats, only their layout descriptors need to be written or rewritten.

- Resources can be discovered *on-the-fly*, and as long as they contain the layout descriptors as part of their metadata, they can be integrated with other resources automatically.
- Unnecessary transformation of data is avoided. In comparison, some approaches for integration require that all data be converted to a single format (such as XML), which can be very expensive if the datasets are large.
- By generating an XML data structure for each transformation, and then using a set of application independent modules, our approach allows efficient execution in a grid environment. Only one set of modules need to be ported on each platform, and then a variety of transformation tasks can be carried out efficiently.

We have demonstrated our approach by two case studies from the bioinformatics domain. They have shown that our system can handle complex transformation tasks effectively. In addition, our experiments results have shown that our automatically generated wrappers scale well to large datasets, the overhead of wrapper generation is very low, and the performance of automatically generated and modular wrapper is within 30% of a hand-written wrapper.

2. SYSTEM OVERVIEW

This section gives an overview of our approach and the system. The overview of our system is shown in Figure 1.

In order to generate a wrapper that is capable of transforming a dataset of a general format into another dataset of a general format, the system needs to have information about the physical data layouts. It also needs to understand the user's logical view of the data (i.e. the schema) so that it can draw the correspondence between the input and output datasets. We have designed a layout description language to achieve both of the above. The information about both the source and the target data layouts are represented using our layout description language. Tabular-structured input or output schemas can also be described using the same language, whereas semi-structured input or output schema are described using the XML DTD format. The *layout parser* parses the layout descriptors and generates internal data entry representations. The schemas are input into the *mapping generator*, which generates the mapping between the source and the target data schema. The inferred schema mapping is presented to the user in a flat file so that it can be verified or modified if needed.

The internal representation of data entries and the mapping completely defines a wrapping task and the functionality of a wrapper can be inferred from them. This inference can be carried out by either the wrapper generation system, or the wrapper itself. For a better overall system performance, we need to reduce the computations performed by the wrapper, and also allow it to execute independent of the wrapper generation system. Therefore, a wrapper generation system module, *Application Analyzer*, performs all the analysis and summarizes important application-specific information for the wrapper in a data structure, which we refer to as the WRAPINFO data structure. The details of the underlying algorithms for this purpose are presented in Section 5.

The wrappers work independently from the wrapper generation system. Our wrappers comprise three modules, the *DataReader*, the *DataWriter* and the *Synchronizer*, each of which is independent of the specific transformation task that needs to be carried out. The information specific to a wrapping task is already captured in the WRAPINFO data structure. Using this data structure as the input, these three modules can carry out a transformation task. The

DataReader and the *DataWriter*, as their names suggest, are responsible for parsing the input dataset and writing to the output files, respectively. The *Synchronizer* serves as a coordinator between these two modules, as it forwards the values constructed by the *DataReader* to the *DataWriter*, and manages the input dataset buffer.

Our design is very well suited for generating wrappers to carry out transformation tasks in a grid environment. Wrapper generation can be easily implemented as a grid service. As shown in our experimental evaluation, for large datasets, the wrapper generation time is a very small fraction of the actual wrapper execution time. The wrapper generator only requires the layout descriptors as input. In comparison, a wrapper needs to be executed at a location where the data movement costs for the input and output datasets are minimized. At the same time, the transformation time can be high, and the wrapper needs to be executed efficiently. By designing the wrapper with application independent modules and representing the WRAPINFO data structure in a machine independent XML file, we make it simpler for the wrappers to be ported for efficient execution on a variety of platforms.

3. TECHNICAL ISSUES AND CHALLENGES

This section gives an overview of the main technical challenges involved in automatic wrapper generation. We also list some restrictions on the transformation tasks that can be carried out by our wrappers.

A number of issues arise because we are dealing with flat-files. The data fields in such files can be separated by different implicit or explicit delimiters. Data fields can be of variable lengths, and certain fields can be optional, whereas certain fields may appear multiple times. Thus, the first set of challenges involve accurately describing such complex physical layouts, correctly parsing the data as per these descriptions, and creating a mapping between physical layouts and the high-level schema. Similarly, for target datasets, we need to capture the layouts, and need to be able to output data to match these layouts.

Clearly, there could be significant differences in source and target high-level schemas. We need to be able to create a mapping between source and target schemas. As described earlier, we consider this a semi-automatic process, and user involvement is possible and sometimes even necessary.

A more significant technical challenge arises because of the possibility that the source and target datasets use different types of schemas. To consider this point, we introduce an application, which we refer to as the TRANSFAC-to-Reference transformation example. This application will be used as a running example in this paper. TRANSFAC [21] is a database on eukaryotic transcription factors, their genomic binding sites, and DNA-binding profiles. Like many biological datasets, it is semi-structured with many optional data fields and a variable number of *references* per record. Suppose, we are interested in obtaining literature information for the transcription factors, as well as their *accession number*, name and species, which are all stored in the ASCII flat file TRANSFAC-factor40. These fields are used to recognize repeated entries from difference data sources. They could also be used to answer queries such as “*Find all the TRANSFAC-FACTOR entries that were published in Cell.*”

To load data into a relational Database Management System, we need to wrap the data from the original semi-structured flat file into a relational table, which we will refer to as the *reference table*. This reference table will have one reference per row.

For our wrapper, we assume that both source and target datasets are organized as a set of records, such that the order of the records in the dataset is not important. Clearly, this assumption holds true for the example we have described above. It should be noted that

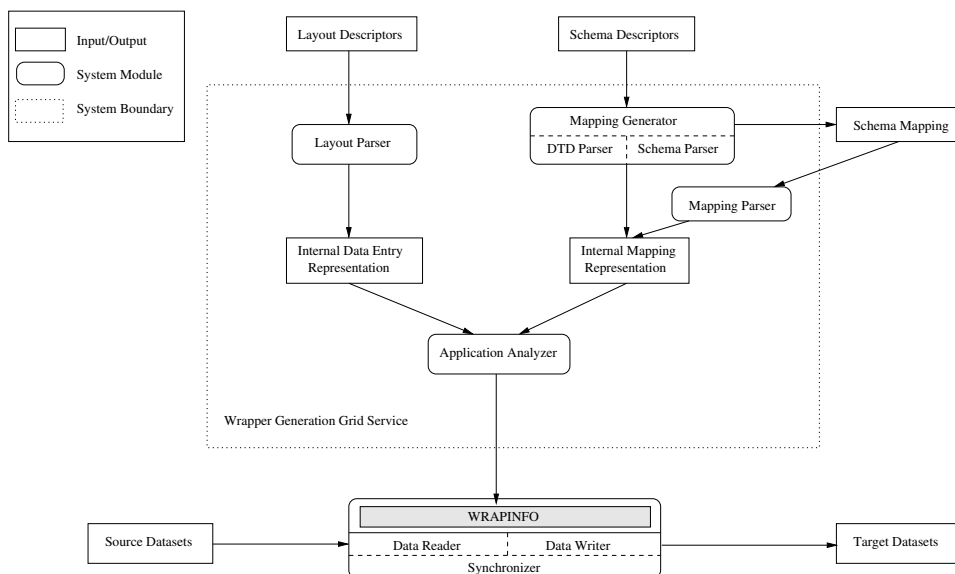


Figure 1: Overview of the System

we do not necessarily have a one-to-one mapping between the records in the source and target datasets. Further, we assume that either of the source or the target dataset uses a tabular schema.

For simplicity, we consider the case when the source dataset is semi-structured and the target dataset is tabular. For such an example, the data fields can be divided into two categories:

DEFINITION 1. A one-to-one data field potentially has multiple values in each source record, and each value will appear only in one target relational record.

DEFINITION 2. A one-to-multiple data field contains only one value in each source record, and this value repeats itself among all the target records transformed from this source record.

For example, when a TRANSFAC-FACTOR entry with three references is converted, three rows are created in the reference table. Each row has its unique reference value. Therefore, the data fields associated with them, such as the reference title, are *one-to-one*. On the other hand, the accession number will be copied to the three rows that are generated, and therefore, it is a *one-to-multiple* data field.

During the transformation process, data fields need to be treated differently based on their type. The value of a one-to-one data field could be discarded once it is written to the output, while the value of a one-to-multiple data field could be written multiple times. Based on this observation, the two types of data fields also require different types of buffers to store their values.

Finally, another important issue related to flat-files is as follows. Flat-files datasets in many domains are intended for human readability. Although from the perspective of the schema, each record comprises several data fields, the representation in a layout is at a finer level. We refer to it as the *variable level*. A *variable* is the smallest unit in processing the dataset layout, and a large data field can be broken into several variables. For example, in the TRANSFAC-FACTOR-TABLE file, when the title of a reference is too long, it is broken into several lines. We call each line one variable, and a number of such lines or variables form one complete title data field. The impact of this possibility is as follows. After extracting a variable value from the input dataset, the wrapper has to decide if it is a continuation of the current data field, or it

signals the start of another data field value. Similarly, when writing a data field value to the output dataset, the wrapper has to decide if the value needs to be split into variables, and if so, what format each variable should follow.

4. LAYOUT DESCRIPTION LANGUAGE

This section briefly describes the layout description language used in our system. This language is used to describe the flat file layouts and relational schemas. We had three goals in designing the language: it should be easy to interpret and process, it should be easy to write, and most importantly, it should have sufficient expressive power. Our language is an extension of a language used previously for supporting SQL queries on flat-file scientific simulation datasets [37]. Our work has significantly extended this language to support several features of less structured datasets.

A descriptor in our language comprises of two components. The *Dataset Schema Description* states the logical or virtual relational view of the data, The *Dataset Layout Description* describes the actual layout of the data within a file. In the dataset layout description, the key words are DATASET, DATATYPE, DATASPACE and DATA. DATATYPE is used to relate a DATASET to a schema. DATASPACE describes the actual layout associated with each file in the DATASET. DATA is used to list the location of the files.

Many features were introduced to deal with datasets which are not very structured. Several special tokens are used in our language. Constant strings, such as the two-character line-type codes used by many biological flat data files, are between double quotes (“”). A repetitive structure which appears at least once, is enclosed by < and >, and if it is optional, it is enclosed between [and]. A general data field named DUMMY is introduced to represent space-filling data that are not in the schema.

The layout descriptors used for the input and output datasets for the TRANSFAC-to-Reference transformation problem we introduced earlier are shown in Figures 2 and 3. The input TRANSFAC data is semi-structured and we use the XML DTD format to describe its schema. The output reference table can be captured using a relational schema. In the layout descriptors, for example, the data field RA, the list of authors, is across multiple lines following the constant string “\nRA” in the TRANSFAC flat file. So, it is inserted between < and > in the input layout descriptor. But, in the

reference table, it is desirable to write the entire list as a continuous string, irrespective of its length. So, the RA field appears alone in the output layout descriptor.

Component I. Semistructured Schema Description (DTD)

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT transfact (AC, ID, FA, SY, OS, REF*, ...)>
<!ELEMENT AC (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT SY (#PCDATA)>
<!ELEMENT FA (#PCDATA)>
<!ELEMENT OS (#PCDATA)>
<!ELEMENT REF (RN, RX, RA, RT, RL)>
<!ELEMENT RN (#PCDATA)>
<!ELEMENT RX (#PCDATA)>
<!ELEMENT RA (#PCDATA)>
<!ELEMENT RT (#PCDATA)>
<!ELEMENT RL (#PCDATA)>
...
```

Component II. Dataset Layout Description

```
DATASET "TRANSFACData" {
  DATATYPE {TRANSFAC}
  DATASPACE {
    <
      "AC " AC
      ["\nXX" DUMMY]
      "ID " ID
      ...
      "\nFA " FA
      ["\nXX" DUMMY]
      ["\nSY" SY]
      ["\nXX" DUMMY]
      "\nOS " OS
      ...
      <
        "\nRN " RN
        ["\nRX " RX]
        <"\nRA " RA>
        <"\nRT " RT>
        "\nRL " RL
      >
      ["\nXX" DUMMY]
      "\n/\n" | EOF
    >
  }
  DATA {data/TRANSFACfactor40.dat}
}
```

Figure 2: The Descriptor for TRANSFAC data

5. SYSTEM IMPLEMENTATION AND KEY ALGORITHMS

This section discusses the important details of wrapper generation and wrapper execution. We first describe the wrapper generation system modules, which include the parsers, the mapping generator, and the analyzer. These modules are collectively responsible for analyzing the information provided by the layout descriptors and presenting them to the wrapper as the WRAPINFO data structure. We will then detail how the wrapper uses this information to carry out the transformation process.

5.1 Wrapper Generation System

5.1.1 Mapping Generator

The goal of the mapping generator is to analyze the schema of the source and the target datasets. It generates the *schema mapping* in the form of an editable text file for the user to verify and/or modify. Currently, we use strict name matching as the criteria for schema mapping. The schemas are first parsed by the *DTD parser* or the *schema parser*. For every field in the target schema, the mapping generator searches the source schema for either a simple element

Component I. Relational Schema Description

```
[TRANSTABLE] // schema name
AC = string // data_field_name = data_field_type
FA = string
OS = string
RA = string
RT = string
RL = string
```

Component II. Dataset Layout Description

```
DATASET "TRANSTABLEData" {
  DATATYPE {TRANSTABLE}
  DATASPACE {
    <
      "" AC
      "\t" FA
      "\t" OS
      "\t" RA
      "\t" RT
      "\t" RL
      "\n"
    >
  }
  DATA {data/TransTable.dat}
}
```

Figure 3: The Descriptor for the TRANSFAC Reference Table

or an attribute with exactly the same name. If no match could be found, the default value of its data type will be used as its mapping. The mapping is presented to the users as an editable file so that the users are able to interact with the system, and define their own mappings.

```
// schema pair, [source schema]:[target schema]
[TRANSFAC]:[TRANSTABLE]
// data fields pairs, source data field : target data field
transfact.AC : AC
transfact.FA : FA
transfact.OS : OS
transfact.REF*.RA : RA
transfact.REF*.RT : RT
transfact.REF*.RL : RL
```

Figure 4: Automatic Generated Schema Mapping File for the TRANSFAC-to-Reference Example

Figure 4 shows the format of the mapping file created by the mapping generator in the TRANSFAC-to-Reference example. The mapping for all six data fields desired by the output are found. This file contains not only the pairing between source and target data fields, but also the cardinality of the pairing. The one-to-one data field pairs are indicated by the "+", "?" or "*" sign, following the element or the attribute where the multiple cardinality is introduced. The use of signs is consistent with the XML DTD standard³.

5.1.2 Layout Parser

The Layout Parser is the module that extracts information from the data layout descriptors and generates representations that can be understood by the analyzer.

Most text based databases' data, including semistructured data, is stored in an entry-wise manner, often for better human interpretability. For the same reason, data fields within each entry are broken into several pieces if they are too long. In our layout descriptors, this is captured by the use of tokens like "<>" and "[]". Further examination of the data layout shows a pattern of alternate *delimiters* and *variables*, which we will refer to as DLM-VAR

³Please see <http://www.w3.org/TR/REC-xml/>

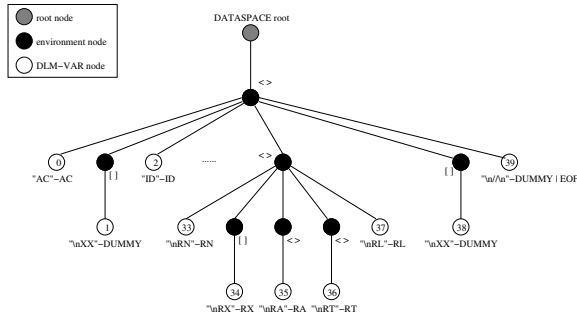


Figure 5: Logical View of TRANSFAC Data Layout as a Tree

Delimiter Look-up Table

Node Number	0	1	2	...	33	34	...
Delimiter	"AC"	"\nXX"	"ID"	...	"\nRN"	"\nRX"	...

Usefulness Look-up Table

Node Number	0	1	2	...	33	34	35	...
Useful?	Yes	No	No	...	No	No	Yes	...

Cardinality Look-up Table

Node Number	0	1	2	...	33	34	35	...
Cardinality	one_to_multiple	-	-	...	-	-	one_to_one	...

Label Look-up Table

Node Number	0	1	2	...	33	34	35	36	37	...
Label	0	-	-	...	-	-	0	1	2	...

Reachable Look-up Table

Node Number	Reachable Nodes
0	1, 2
1	1, 2
2	3, 4
.	.
.	.
.	.
33	34, 35
34	34, 35
35	35, 36
36	36, 37
37	33, 38, 39
.	.
.	.
.	.

Parameters

one_to_one_total=3, one_to_multiple_total=3, complete_in=39

Table 1: WRAPINFO data structure for the TRANSFAC-to-Reference Example

pairs. The *delimiters* are special values explicitly inserted in order to separate the values of interest, which are the variables. Exceptional cases may exist, but they could be generalized by introducing *dummy* delimiters and *dummy* variables.

Based on this observation, we have designed a tree data structure to capture the layout of the text data. As an example, the tree view of the TRANSFAC data is shown in Figure 5. In this tree, the leaves are DLM-VAR pair nodes. The last leaf is a generalized DLM-VAR pair with a dummy variable. The internal nodes in the tree, also called the *environment* nodes, indicate how the children are repeated.

5.1.3 Application Analyzer

After the layout and schema information is converted by the Layout Parser and the Mapping Generator into internal forms, they are processed by the Application Analyzer. As we stated earlier, the output of the Application Analyzer is a data structure we refer to as WRAPINFO. This data structure should capture all application specific information needed by the wrapper. An additional goal is

store this information in a form that minimizes any execution time analysis by the wrapper modules, and therefore, allows efficient execution.

To understand the information that is needed for the wrapping process, we can view the DataReader and the DataWriter modules in the wrapper as two Turing machines, each with multiple tapes. Their particular configuration is determined by the application. The number of tapes are dependent on the target data schema. Furthermore, each tape serves as a buffer for an output data field. The automata are related to the data layouts and the transition actions mimic the process of tracing the DLM-VAR nodes along the layout tree. Then, the goal of the Application Analyzer is to generate the suitable configuration information for each application and store it. The configuration of the tapes are stored as constant parameters, and the transition actions of the automata are coded as look-up tables. The WRAPINFO data structure comprises these parameters and look-up tables.

One important concept is the *reachability* of one DLM-VAR pair from another. Consider a node a in the layout tree we had described earlier. Then, $a.DLM$ and $a.VAR$ are respectively the delimiter and the variable for this node. The reachability is defined as follows:

DEFINITION 3. A node b is reachable from the node a if there can be a valid layout in which $a.DLM$ and $b.DLM$ form the boundaries for $a.VAR$.

Intuitively, the set of reachable nodes contain all the DLM-VAR pairs that could be read/written next. Thus, their delimiters form the set of all possible strings indicating the end of the current variable value.

The Application Analyzer performs the following actions to populate the WRAPINFO data structure:

1. Number all the DLM-VAR nodes in the source and target layout trees, as shown in Figure 5.
2. Create a *delimiter look-up table* for the source and the target indexed by the DLM-VAR node number.
3. Mark the DLM-VAR nodes in the source layout trees as either *useful*, if the corresponding data field is in the schema mapping, or *useless* otherwise. Create a *usefulness look-up table* indexed by the DLM-VAR node number.
4. Categorize the useful DLM-VAR nodes according to their data fields' cardinality counts and record the totals as constants, *one_to_one_total* and *one_to_multiple_total*.
5. Label one-to-one and one-to-multiple nodes separately and record the labels and the cardinality information in a *label look-up table* and a *cardinality look-up table*.
6. For each DLM-VAR pair, generate a list of DLM-VAR nodes that are *reachable* from it, as defined earlier. Record all of its reachable nodes' numbers as one row in a *reachable look-up table*.
7. Find the DLM-VAR node in the source layout tree which indicates the completion of reading one data entry. Record its number as *complete_in*.

An example of the WRAPINFO data structure is shown in Table 1. The specific values are for the TRANSFAC-to-Reference transformation problem.

5.2 Wrapper

We now describe the actual implementation and execution of the wrapper. Based on the WRAPINFO data structure, the wrapper first sets up suitable buffers for the particular transformation task. For each one-to-multiple data field, at any given time, only one value is valid. Therefore, a single elementary data type variable needs to be declared. For our discussion, assume the data type to be string. Thus, collectively, a string array of size *one_to_multiple_total* is declared to buffer all the one-to-multiple data fields. On the other hand, by the time when a complete source data entry is read, a one-to-one data field could have several valid values and they all need to be buffered before they can be written to the output dataset. To store all the one-to-one data fields' value, an array of string lists of size *one_to_one_total* is declared.

The string array and the string list array are shared by the DataReader and the DataWriter. The DataReader scans the input dataset, reconstructs the values for the data fields and writes them to this value buffer. The DataWriter then retrieves the data, writes them to the output dataset, and empties the buffer when needed. The Synchronizer, as the name suggested, synchronizes their actions.

In the rest of this section, we will explain in details the three general modules, i.e., DataReader, DataWriter and Synchronizer.

5.2.1 DataReader and DataWriter

While scanning the input dataset, the DataReader keeps track of the number of the DLM-VAR node being read. From its reachable list, the DataReader chooses the DLM-VAR node whose delimiter appears first in the input data buffer as the next DLM-VAR node. It considers the end of this variable as being just before this delimiter. If this value is useful, it will be extracted and appended to a temporary string and the searching process is repeated. The DataReader determines if the value of current data field is completely reconstructed by checking the labels of the nodes visited. When the label changes, it indicates the beginning of another data field. (Otherwise, it means that the current data field continues on the next line.) At this time, the DataReader stores the value in the temporary string according to the data field's cardinality. If it is *one_to_one*, the new value will be added to the correct list. If it is *one_to_multiple*, this value will replace the old one.

Similarly, the DataWriter keeps track of the number of the DLM-VAR node being written. When the DataWriter is ready to write these values to the target dataset, it always gets the value of a *one_to_one* data field by removing it from its list and gets the value of a *one_to_multiple* data field by copying instead. If the value is too long, it is splitted whenever possible, given that the data field is allowed to continue on multiple lines.

Both modules return the number of the next node to be read/written, which is used by the Synchronizer as signals of the progress of the reading/writing process.

5.2.2 Synchronizer

The Synchronizer controls the progress of the wrapper by switching between calling the DataReader and the DataWriter. It is also responsible for managing the input dataset buffer for the DataReader to scan. The Synchronizer repetitively calls the DataReader until the data buffer is filled with all the useful data from one source entry. This is indicated by the completion of data value extraction for the data field with node number *complete_in*. Then, the Synchronizer starts to call the DataWriter, which will write the data to the output dataset following the desired format. The DataWriter will also erase the values of the one-to-one data fields, The Synchronizer watches for the completion of the writing process by monitoring the value buffer for all the one-to-one data fields. Once they are all empty, the Synchronizer stops calling the DataWriter and starts a new cycle by calling the DataReader. In each cycle, one source

data entry is transformed, and one or multiple corresponding target data entries are written to the output.

As the size of the input dataset is unknown beforehand, the wrapper reads the source data into the input data buffer chunk by chunk instead of by the entire file. As the DataReader scans this buffer, it continuously reduces it by removing the contents that have been processed. Whenever the DataReader is unable to find any of the reachable delimiters in the buffer, it indicates that the current variable value extends beyond the current input data buffer. The Synchronizer will stop the DataReader, read in the next chunk from the source dataset, append it to the input data buffer and resume the DataReader.

6. CASE STUDIES AND EXPERIMENTAL RESULTS

This section presents two case studies showing the application of our system on real examples. We also present quantitative data, focusing on 1) the scalability of automatically generated wrappers, 2) comparing the time required for wrapper generation with that of wrapper execution, and 3) comparing automatically generated wrappers with hand-written ones, if available. Both our case studies are from bioinformatics domain. TRANSFAC-to-Reference has been used as a running example throughout the paper. SWISSPROT-to-FASTA is another well known data transformation problem in bioinformatics.

6.1 TRANSFAC-to-Reference

The challenges associated with this transformation problem have been described throughout the paper. Here, we focus on the performance data.

In TRANSFAC release 3.4 [13], the FACTOR flat file contains 2376 entries and is of size 6.0 MB. To evaluate the scalability of the generated wrapper, we duplicated this file several times to generate 12.0 MB, 24.1 MB and 48.2 MB datasets. The experimental results are summarized in Figure 6. Because the wrapper generation times are much smaller than the wrapper execution times, we have used a logarithmic scale, and all times are shown in milliseconds. The results show that the execution time stays very close to being linear to the size of the dataset. Specifically, the ratios between the execution times are 1:2.12:4.27:8.33. Figure 6 also shows the wrapper generation overhead, that is, the time it takes for the generation system to analyze the application and write the WRAPINFO data file for the wrapper. The overhead is a constant, regardless of the size of the input dataset, and relatively small compared to the actual execution time of the wrapper. In this example, because of the large amount of optional fields in the TRANSFAC-FACTOR data file, the input data layout tree contains 40 DLM-VAR pairs and the largest set of reachable nodes is of size 23. As a result, the WRAPINFO data structure is relatively complicated. For example, the reachable look-up table is a 40×23 array. Despite the large size of the look-up tables, the average wrapper generation time is only 0.051 seconds, which corresponds to 1.2% of the wrapper execution time for the 6.0 MB dataset.

6.2 SWISSPROT-to-FASTA

We also tested our wrapper with a common data transformation problem in bioinformatics, which we refer to as the SWISSPROT-to-FASTA problem. SWISSPROT database is one of the well-known collections of protein sequences, and often needs to be queried with the popular analysis program, BLAST. However, this cannot be done directly, as BLAST requires the sequences to be stored in a format referred to as the FASTA format, and SWISSPROT data is in a different and much more complicated format.

Many hand-written wrappers are developed for the SWISSPROT-to-FASTA application. For example, SEQIO [29] package has a file conversion program called *fmtseq*. This case studies shows that our automatically generated wrapper can solve this problem just as well as these hard-coded tools. The performance of our

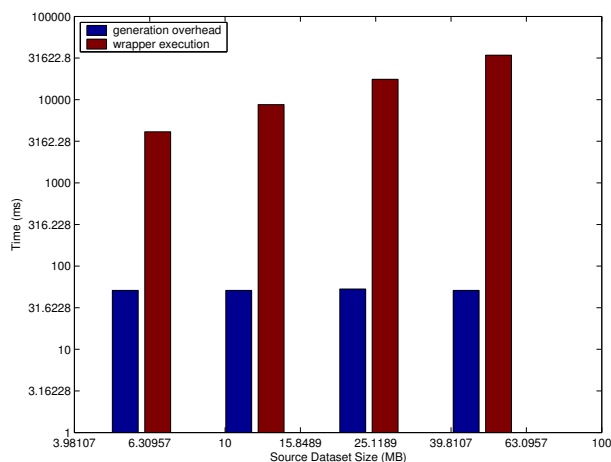


Figure 6: Results from TRANSFAC-to-Reference Problem

automatically generated wrapper and the handwritten *fmtseq* package are compared in Figure 7. For the 153,871 sequence entries in SWISSPROT Release 44.0, our wrapper takes 132.01 seconds, while *fmtseq* takes 100.44 seconds. This shows that our automatically generated wrapper is still quite efficient. Our wrapper generation system also provides users with greater flexibility, such as allowing them to customize the fields and add personalized comments. Figure 7 also shows that the generated wrapper scales well with larger dataset sizes. When we double, triple and quadruple the SWISSPROT dataset, the wrapping time increases with a ratio of 2.04, 3.12 and 4.13 times, respectively, as compared to the original dataset. The WRAPINFO data structure generated for this example is simpler than that for the previous example. The wrapper generation overhead is 0.042 second, which is 0.032% of the time to transform a single SWISSPROT dataset.

7. RELATED WORK

The issues of format conversion for flat-files in a grid environment have not received much attention. IBM's CLIO project uses database schemas to derive the transformations needed to integrate the datasets [26]. This work assumes that data is stored in databases with well-defined query interfaces, and is therefore, not applicable to flat-files. The support for *external tables* as part of Oracle's recent implementation allows tables stored in flat-files to be accessed from a database⁴. The data must be stored in the table format, or an *access driver* must be written.

The myGrid project has been developing technologies for integrating a variety of services in the web, through the use of web service composition language [39]. IBM has been developing Bioinformatic Workflow Builder Interface (BioWBI) for creating web service based workflows for biological researchers⁵. These two efforts typically require: 1) Use of XML for exchange of data between different sources, which can introduce high overheads, and 2) Java wrappers on existing applications, which can also introduce overheads. Our proposed system can overcome these limitations, though it cannot provide as much interoperability as is possible through web services.

BinX and Binary Format Description (BFD) [5] is a meta-data descriptor that gives a machine-readable view of a binary file. Our layout description language has many similarities, but our main contribution is to automatically generate the wrappers using such descriptors. Other closely related efforts in the grid community are as follows. Metadata cataloging and metadata services have

⁴See www.dbasupport.com/oracle/ora9i/External_Tables9i.shtml

⁵See <http://www.alphaworks.ibm.com/tech/wsbaw>.

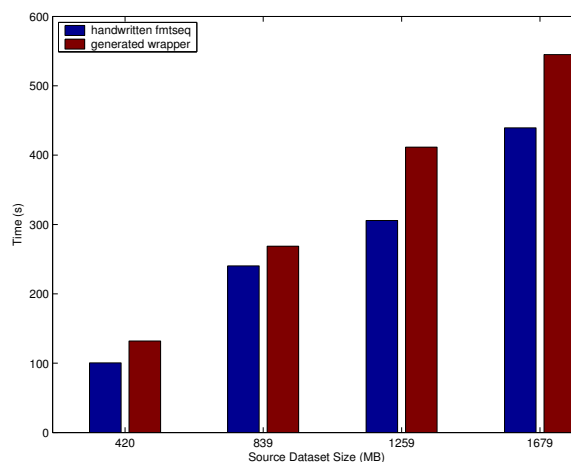


Figure 7: Results from SWISSPROT-to-FASTA Problem

received much attention lately [12, 33]. Their focus is on mechanisms for storing, discovering, and accessing metadata. Metadata catalogs have been used by Artemis project [34] for supporting uniform access to heterogeneous scientific data sources in Grid environment. Our goal is clearly complimentary, as we use metadata on the layout for generating wrappers. We hope to integrate our work with grid-based metadata services eventually. The SDSC Storage Resource Broker (SRB) provides uniform access to heterogeneous data sources. However, it does not include support for automatically generated wrappers, and most of the transparency provided is across storage devices.

A number of efforts exist on mediator-based bioinformatics integration, as reviewed in [22]. These existing mediator-based systems require hand-written wrappers. Specific examples include K2/BioKleisli [38], TAMBIS [18], Biomediator [30], and DiscoveryLink[20]. Automatic wrapper generation has also been an active research topic. Currently, most of the automatic wrapper generation research has focused on extracting information from tabular structures in HTML files [4, 28, 7, 16, 24]. ROADRUNNER [10] generates record layout structure by comparing HTML pages. Arasu *et al.* have proposed an approach [3] where no heuristics on HTML were used. However, multiple pages generated from a same template must be collected for template construction. Overall, automatic wrapper generation using layout descriptors has not been studied by any of these efforts.

Information or data integration has been widely studied for more than two decades now. One of the early approaches to information integration was using federated databases [31]. Use of mediators has been another dominant approach, and has been taken by projects like TSIMMIS [17] and InfoHarness [32], as well as many bioinformatics projects we listed above. More recently, focus has been on semantic interoperability. Use of ontologies and ontology alignment [23] and semantic mediation [19] are some of the approaches. Many algorithms for schema mapping have been proposed, and are reviewed by Rahm and Bernstein [27]. The initial prototype of our wrapper generation system uses a strict name matching algorithm to generate schema mapping. More advanced algorithms can be easily incorporated into our system because of the modular design and our future work will consider this.

Our work also has similarity with earlier work on data translation. Mamrak *et al.* have developed a system called Chameleon [25] to help automate data translation tasks. A more recent system, Configurable Chemical Middleware (C2M) is based on the same approach [35]. The key limitation of both these systems is that a common intermediate format is needed between the source and tar-

get datasets. The user needs to describe the relationship between each of the source and target datasets to such a common format. In comparison, our approach can translate between two formats whose layout descriptors have been written independently, and does not require a common intermediate format.

8. CONCLUSIONS

This paper has presented a new approach for information integration. We believe that our approach is effective and practical for a number of scenarios, including, grid-based data integration, supporting scientific workflows, and enabling data sharing in collaborative environments. The key advantage of our approach is that once a layout descriptor is written for a data source or the input/output of a service, integration and interoperability can be achieved automatically. Unlike the traditional approaches that require manually written wrappers, our approach requires only a small amount of effort when a new data source or tool needs to be added to the integration system, or when the data format of a resource changes.

Our current system requires that layout descriptors be written manually. In our ongoing work, we are focusing on making this process a semi-automatic one. On one hand, we are using a variety of data mining techniques to learn the format of a data source. At the same time, we are using techniques for error-recovery in parsing to provide feedback when a layout descriptor needs to be corrected.

9. REFERENCES

- [1] David Abramson and Jagan Kommineni. A Flexible IO Scheme for Grid Workflows. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.
- [2] W. E. Allcock, I. Foster, and R. Madduri. Reliable Data Transport: A Critical Service for the Grid. In *Proceedings of the Workshop on Building Service Based Grids*, 2004.
- [3] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348, 2003.
- [4] Naveen Ashish and Craig A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*. IEEE Computer Society, 1997.
- [5] R. Baxter, R. Carroll, D. J. Ecklund, B. Gibbins, D. Virdee, and Q. Wen. BinX - A Tool for Retrieving, Searching, and Transforming Structured Binary Files. Available at <http://www.edikit.org/binx/pub.htm>, 2003.
- [6] M. Cai, A. Chervenak, and M. Frank. A Peer-to-Peer Replica Location Service Based on A Distributed Hash Table. In *Proceedings of SC 2004*, November 2004.
- [7] Liangyou Chen, Hasan M. Jamil, and Nan Wang. Automatic wrapper generation for semi-structured biological data based on table structure identification. In *14th International Workshop on Database and Expert Systems Applications*. IEEE Computer Society, 2003.
- [8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbusy, and S. Tuecke. The Data Grid: Towards An Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 2001.
- [9] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. In *Proceedings of the Conference on High Performance Distributed Computing (HPDC)*, June 2004.
- [10] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [11] Ewa Deelman, Jim Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Albert Lazzarini, Adam Arbre, Richard Cavanaugh, and Scott Koranda. Mapping Abstract Complex Workflows onto Grid Environments. In *Journal of Grid Computing*, pages 9–23, 2003.
- [12] Ewa Deelman, G. Singh, M.P. Atkinson, A. Chervenak, N.P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su. Grid-Based Metadata Services. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM04)*, 2004.
- [13] E. Wingender et al. Transfac release 3.4: Technical information. <http://www.es.embnet.org/Services/ftp/databases/transfac/readme.txt>, 1998.
- [14] V. Fontes, B. Schulze, M. Dutra, F. Porto, and A. Barbosa. CoDIMS-G: A Data and Program Integration Service for the Grid. In *Proceedings of the Workshop on Grid Middleware (held in conjunction with Middleware 2004)*. ACM Press, October 2004.
- [15] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. In *Proceedings of the Conference on Scientific and Statistical Data Management*, July 2002.
- [16] X. Gao and L. Sterling. Autowrapper: automatic wrapper generation for multiple online services. In *Asia Pacific Web Conference '99*, 1999.
- [17] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and Accessing Heterogenous Information Sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, 1995.
- [18] C. A. Goble, R. Stevens, G. Ng, S. Bechhofer, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent access to multiple bioinformatics information sources. *IBM Systems Journal*, 40(2), 2001.
- [19] A. Gupta, B. Ludascher, and M. E. Martone. Registering Scientific Information Sources for Semantic Integration. In *Proceedings of Conference on Conceptual Modelling*, 2002.
- [20] L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope. Discoverylink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40(2), 2001.
- [21] T. Heinemeyer, E. Wingender, I. Reuter, H. Hermjakob, A. E. Kel, O. V. Kel, E. V. Ignatieva, E. A. Ananko, O. A. Podkolodnaya, F. A. Kolpakov, N. L. Podkolodny, and N. A. Kolchanov. Databases on transcriptional regulation: Transfac, trrd, and compel. *Nucleic Acids Research*, 26:364–370, 1998.
- [22] Thomas Hernandez and Subbarao Kambhampati. Integration of biological sources: Current systems and challenges ahead. *SIGMOD Record*, 33(3):51–60, 2004.
- [23] E. Hovy. Combining and Standardizing Large Scale, Practical Ontologies for Machine Translation and Other Uses. In *First International Conference on Language Resources and Evaluation*, 1998.
- [24] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.
- [25] S. A. Mamrak, M. J. Kaelbling, C. K. Nicholas, and M. Share. Chameleon: A system for solving the data-translation problem. *IEEE Transactions on Software Engineering*, 15(9), 1989.
- [26] R. Miller and M. A. Hernandez and L. M. Haas and L. Yan and C. H. Ho and R. Fagin and L. Popa. The CLIO Project: Managing Heterogeneity. *ACM SIGMOD Record*, pages 78–83, 2001.
- [27] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal – The International Journal on Very Large Data Bases*, 10(4), 2001.
- [28] Daniel Rocco and Terence Critchlow. Automatic discovery and classification of bioinformatics web sources. *Bioinformatics*, 19(15), 2003.
- [29] Seqio: A package for reading and writing sequence files. <http://bioweb.pasteur.fr/docs/seqio/seqio.html>, 1996.
- [30] Ron Shaker, Peter Mork, J Scott Brockenbrough, Loren Donelson, and Peter Tarczy-Hornoch. The biomediator system as a tool for integrating biologic databases on the web. In *Proceedings of the Workshop on Information Integration on the Web*, 2004.
- [31] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
- [32] L. Shklar, A. Sheth, V. Kashyap, and K. Shah. InfoHarness: Use of Automatically Generated Metadata for Search and Retrieval of Heterogeneous Information. In *Proceedings of CAISE*, 1995.
- [33] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. In *Proceedings of Supercomputing 2003 (SC2003)*, November 2003.
- [34] Rattapoom Tuchinda, Snehal Thakkar, Yolanda Gil, and Ewa Deelman. Artemis: Integrating scientific data on the grid. In *Proceedings of the 16th Annual Conference on Innovative Applications of Artificial Intelligence (IAI)*, pages 892–899, 2004.
- [35] P.E. van der Vet, H.E. Roosendaal, and P.A.T.M. Geurts. C2m: configurable chemical middleware. *Comparative and functional genomics*, 2:371–375, 2001.
- [36] S. Vazhkudai and J. Schopf. Using disk throughput data in predictions of end-to-end grid transfers. In *Proceedings of the Third Workshop on Grid Computing (Grid 2002)*, November 2002.
- [37] Li Weng, Gagan Agrawal, Umith Catalyurek, Tahsin Kurc, Sivaramkrishnan Narayanan, and Joel Saltz. An Approach for Automatic Data Virtualization. In *Proceedings of the Conference on High Performance Distributed Computing (HPDC)*, June 2004.
- [38] Limsoon Wong, Kleisli, a functional query system. *Journal of Functional Programming*, 10(1), 2000.
- [39] C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *Journal of Cooperative Information Science*, 2003.
- [40] Tong Zhao, Michael Wilde, Ian Foster, Jens Voeckler, Thomas Jordan, Elizabeth Quigg, and James Dobson. Grid middleware services for virtual data discovery, composition, and integration. In *Proceedings of the Workshop on Grid Middleware (held in conjunction with Middleware 2004)*. ACM Press, October 2004.