Using Metadata for Automatic Wrapper Generation (Extended Abstract)

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

1. INTRODUCTION
In scientific data-intensive computing, 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 example, Chimera is a system for supporting virtual data views and demand-driven data derivation [3, 11]. Similarly, CoDIMS-G is a system providing grid services for data and program integration [2]. 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 area[1].

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 [4, 8, 5, 10].

In this paper, we present an approach for automatic wrapper generation suitable for a scientific data-intensive environment, which is based on a novel usage of metadata. We believe that hand-written wrappers are not practical and scalable in emerging data-intensive environments 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 metadata, specifically, a 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. 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 published, 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 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.

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 [6] 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 infor-
mation for the transcription factors, as well as their accession number, name and species, which are all stored in the ASCII flat file TRANSFACfactor40. 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 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 [9]. 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.

5. **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.

5.1 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 [1], 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 4. 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 4 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 consists of 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 $40 \times 23$ 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.

5.2 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 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 automatically generated wrapper and the handwritten fmtseq package are compared in Figure 5. 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 5 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.

6. 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 techniques for error-recovery in parsing to provide feedback when a layout descriptor needs to be corrected.

7. REFERENCES


Component I. Semistructured Schema Description (DTD)

```xml
<?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

"ID " ID

"nFA " FA

"nSY " SY

"nOS " OS

"nRN " RN

"nRX " RX

"nRA " RA

"nRT " RT

"nRL " RL

"n//n" EOF

Figure 2: The Descriptor for TRANSFAC data

---

Component I. Relational Schema Description

```sql
[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 " AC  
  "ID " ID  
  "nFA " FA  
  "nSY " SY  
  "nOS " OS  
  "nRN " RN  
  "nRX " RX  
  "nRA " RA  
  "nRT " RT  
  "nRL " RL  
  "n//n" EOF  
>
```

} DATA {data/TRANSFACfactor40.dat}

}

Figure 3: The Descriptor for the TRANSFAC Reference Table

---


