MORPHING SPARSELY POPULATED DATA

Susan Davidson , Carmem Hara, Anthony Kosky and Chris Overton

University of Pennsylvania, Department of Genetics and Department of Computer and Information Science

INTRODUCTION

The Human Genome Project, and the field of molecular biology, involves a proliferation of different data sources, including both archival (GenBank, GDB) and local notebook databases, and also various software systems and tools (BLAST, FASTA). These data sources frequently use incompatible structures to represent the same or overlapping data, and further may be implemented in a variety of data-models and database management systems (DBMSs), including object-oriented systems (ACEDB, OPM, Object Store), flat-relational databases (SYBASE), and data exchange formats stored as structured text files (ASN.1). It is frequently necessary to transform data between these incompatible data sources and data-models. For example data stored in an archival database or generated by and stored in a local database at one HGP site may have an impact on the experiments being carried out at another site, and therefore needs to be stored in the local laboratory notebook database at the second site. Further useful tools, such as data browsing or analysis tools, may be implemented for a particular DBMS or database schema, and it is desirable to move data from another database into this system so as to apply these tools. Also constantly changing experimental and analysis techniques result in rapidly evolving database schemas, and it is necessary to transform data in order to reflect these schema evolutions.

In this abstract, we describe a prototype system called *Morphase* (an enzyme for data morphing) for facilitating such transformations, and discuss problems that arise when dealing with sparsely populated data such as that found in ACE and ASN.1.

MORPHASE

To ease the specification and implementation of data transformations between various data sources, we have been developing a prototype system called Morphase. Transformations between a source and target database are specified in Morphase using a high-level declarative language *WOL (Well-principled Object Language)*, which is based on Horn clause logic [1]. Transformation specifications are then translated into an underlying database programming language, *CPL (Collection Programming Language [2, 3])*, for implementation. The data-types supported by WOL include arbitrarily nested records, sets, variants, and more recently lists, thus capturing the types common to most data formats.

The CPL language interfaces to a wide variety of data sources, and has been designed to be easily extensible with drivers for new data sources as they arise. In particular, drivers to connect CPL to ASN.1, ACEDB and SYBASE have been developed; other drivers, for example to OPM, are currently being developed. The drivers are used to query as well as update data servers which are instances of their type, e.g. the Sybase driver is used for our local Sybase database, Chr22DB. To perform a query,

the driver forwards it to the data server after its translation in the server language; the resulting data is then translated into CPL format and sent back to CPL. For updates, the driver transforms CPL format into server insert code and perform updates at the data server.

Constraints on data at both the data sources and target databases are important at several stages in Morphase. After the transformation is specified, they are used to complete the specification and produce normal form rules. Constraints that are most important in this process are keys and inclusion dependencies, such as those that arise through inheritance hierarchies. Inclusion dependencies are also used to structure the insert code in the driver and avoid aborts. Since schemas typically have many constraints and they are tedious to write in WOL, we are also developing tools that are used by Morphase to automatically extract constraints from meta-data stored in various data sources. This dramatically simplifies the task of writing the transformation programs since it allows the user to focus on the logical content of the transformation.

Initial work on WOL was reported in MIMBD ’94, and a description of using WOL to transform data from a nested relational format into flat-relational format can be found in [4]. In the past year, the Morphase tool-set has been developed around the WOL prototype, and a number of important extensions for dealing with format data and object-oriented data have been made including: 1. Support for object-identities, which are referenced and created using surrogate keys; and 2. Support for lists, which frequently occur in structured text files (such as ASN.1) and tree-structured databases (such as ACEDB).

EXPERIENCES WITH TRANSFORMATIONS

We have been testing our formalism by applying it to a variety of different transformation problems involving ACEDB and ASN.1. Some of these transformations have already been hand-coded without our tools, and the comparison between the approaches has given us several insights. The first observation is that data in both ACE and ASN.1 yield a large number of variant types when translated into our data model. This is due to the fact that these formats are sparsely populated, that is they have "been developed for scientific research projects where many data are very incomplete" [5]. While a relational representation of the same data would use null values, in these formats the tagged field is simply missing. The second observation is that transformations involving variants that are hand coded typically experience an explosion in the size of programs that is exponential in the number of variants in the databases involved. WOL, on the other hand, allows clauses to express partial information about records, thus avoiding the explosion when specifying the transformation. However, to implement the transformation the specification is translated into a normal form, which again produces an exponential number of clauses in the number of variants in the databases involved. In order to avoid this exponential blow up, we found it necessary to introduce intermediate representations of data and implement transformations in two stages rather than one, which was our initial goal.

AN EXAMPLE

To illustrate this problem, consider the following translation from a simple ACE database to a nested relational structure. Since examples from molecular biology tend to be highly complex, and require a lot of explanation, we will instead use a simple ’T-shirts’ example adapted from the ACEDB design guide, [6]. Our ACEDB database consists of two classes representing t-shirts and events, either being borrowed or damaged, in the history of a t-shirt.

```
?TShirt Characteristics  Background     Colour UNIQUE  Black
                                                       Purple
                                                       Pink
                         Design UNIQUE  Pattern UNIQUE Floral
                                                       Abstract
                                        Plain
                         Graphics  Front     UNIQUE Text
                                   Back      UNIQUE Text
                         Size      UNIQUE Int
        History          UNIQUE ?Event  XREF TShirt REPEAT


?Event    TShirt    UNIQUE    ?TShirt   XREF History
          Type UNIQUE    Damaged   Damage_date    UNIQUE Int
                                   Description    UNIQUE Text
                         Borrowed  Borrowed_date  UNIQUE Int
                                   Person         UNIQUE Text
```

The combination of UNIQUE and REPEAT operators in the history attribute mean that history is represented as a list of events.

We might wish to exchange data between this database and a nested relational database with the schema:

```
TShirt(Id: int,
       Color: string,
       Design: string,
       Borrowed-history: [(Date: int, Person: string)],
       Damage-history: [(Date: int, Description: string)] )
```

Here "(....)" denotes records and "[...]" denotes lists. In order to specify a transformation from the ACE representation to the nested relational representation, we write the following WOL clauses:

```
(Id = MkTS(Y.Key), Color = "black") in Tgt.TShirt <==
     Y in Src.?TShirt,
        Y.Characteristics.Background.Colour = choice(Black, ())

(Id = MkTS(Y.Key), Color = "pink") in Tgt.TShirt <==
     Y in Src.?TShirt,
        Y.Characteristics.Background.Colour = choice(Pink, ())

(Id = MkTS(Y.Key), Design = "floral pattern") in Tgt.TShirt <==
     Y in Src.?TShirt,
        Y.Characteristics.Background.Design = choice(Pattern, choice(Floral,())
```

..... similar clauses for other colours and and patterns .....

```
(Id= MkTS(Y.key), (Date= D, Person= P) in Borrowed-history) in Tgt.TShirt <==
          Y in Src.?TShirt, X in Y.History,
          X.Type = choice(Borrowed, (Borrowed_date = D, Person = P))


(Id=MkTS(Y.key),(Date =D, Description= T) in Damage-history) in Tgt.TShirt <==
          Y in Src.?TShirt, X in Y.History,
          X.Type = choice(Damaged, (Damage_date = D, Description = T))
```

The first clause says that for every object Y in the source class ?TShirt, such that the Background.Colour attribute of Y is set to Black, there is a corresponding object in the target database table TShirt with its

Id attribute set to MkTS(Y.Key) and its Color attribute set to the string "black". The Key attribute of ACE object Y refers to the "hidden" field that is required for every class in ACE. The term MkTS(Y.Key) is an example of the use of a Skolem function, and creates an object-identifier for each value it receives. If the function MkTS is applied to the same value twice then it returns the same object identifier, but if it is applied to a new value then it creates a new object-identifier corresponding to that value.

Note that the history attribute of the source ?TShirt class is a list which is being split into two lists, Damage-history and Borrowed-history, in the target database. An atom of the form "X in Y.history" assigns a precedence to an occurrence of X representing its position in the list Y.History, so that the relative order of events in the history list is preserved in the Damage-history and Borrowed-history lists of the target database. However the reverse transformation, from the nested relational database to the ACEDB database, presents a problem: though it is possible to instantiate the history attribute of a ?TShirt with a list formed by appending the translations of the Damage-history and Borrowed-history lists, it is not possible to restore the original interleaving of these entries without using some additional information. In this respect, information is being lost by the transformation.

The clauses of the WOL transformation program are then transformed into clauses in a normal form, in which a complete entry in the target database is described in terms of the source databases. While the number of clauses in the WOL transformation specification are linear in the number of variants in the source database, the number of normal form clauses will be exponential: one for each possible set of choices (black/plain/front-graphics-only, black/plain/back-graphics-only,..., purple/floral-pattern/back-and-front-graphics, purple/abstract-pattern/back-and-front-graphics, and so on). Though this might be manageable with simple structures like the one in this example, it turns out to be a serious problem when dealing with realistic examples in data-models such as ACEDB or ASN.1, where variants and optional attributes are common. We can avoid this exponential blowup by introducing an intermediate representation of data, with separate structures corresponding to each variant. For the above example, the intermediate representation would have four separate tables, each with an Id attribute and one of the attributes Color, Design, Borrowed-history or Damage-history. The transformation then proceeds in two stages: first populating the intermediate relations, and then joining them on the Id attributes to form the final target relation.

It is also possible to construct a reverse transformation from the nested relational to the ACEDB database. Such a transformation will show up certain incompatibilities in the information capacities of the two databases. We already noted that information about the interleaving of damaged and borrowed events in the history of a ?TShirt is lost. In other respects the nested relational database is much more expressive than the ACEDB version. For example a t-shirt in the ACEDB database can have one of three colours: black, purple or pink. The colour attribute of a TShirt in the relational database can however be any string, such as "red", "green" or "chartreuse". In order for the transformations to be information preserving it is necessary ensure additional constraints, such as that the colour of a t-shirt is classified as either "black", "purple" or "pink".

CONCLUSIONS

The need for database transformations occurs frequently when dealing with Human Genome Project or other molecular biology databases. Implementing such transformations by hand on a case by case basis is time consuming and error prone. Consequently there is a need for a method of specifying and implementing such transformations in a uniform way, allowing transformations to be specified across a

wide variety of different data-models, and to be formally analyzed and verified.

Our initial prototype implementation of Morphase dealt with the special case of transformations to flat relational target databases, and is being used in test case studies at the Human Genome Center for Chromosome 22 at the University of Pennsylvania. Work on an extended version of Morphase for implementing transformations between more general databases is underway.

ACKNOWLEDGMENTS

This research was supported in part by DOE DE-FG02-94-ER-61923 Sub 1, NSF BIR94-02292 PRIME, ARO AASERT DAAH04-93-G0129, and a scholarship from CNPq - Brasilia/Brazil.

REFERENCES

- [1] Anthony Kosky, "Types with Extents: On transforming and querying self-referential data-structures", PhD Thesis Proposal, Dept. of Computer and Information Science, University of Pennsylvania. 1995.
- [2] P. Buneman, L. Libkin, D. Suciu, V. Tannen and L. Wong, "Comprehension Syntax". SIGMOD Record, Vol 23. March 1994.
- [3] P. Buneman, S. B. Davidson, K. Hart, C. Overton and L. Wong, "A Data Transformation System for Biological Data Sources," Technical report MS-CIS-95-10, Dept. of Computer and Information Science, University of Pennsylvania. March 1995.
- [4] S. B. Davidson and A. S. Kosky and B. Eckman, "Facilitating Transformations in a Human Genome Project Database", Proc. of CIKM, December 1994.
- [5] J. Thierry-Mieg and R. Durbin, "Syntactic Definitions for the ACEDB Data Base Manager." Technical report, MRC Laboratory for Molecular Biology, Cambridge. 1992.
- [6] Sam Cartinhour, "Exploring Models Design and Structure".