Pathway and Protein Interaction Data: from XML to FDM Database

Graham J.L. Kemp and Selpi

Department of Computing Science, Chalmers University of Technology,
SE-412 96 Göteborg, Sweden

Abstract. This paper describes our experience with the first steps towards integrating pathway and protein interaction data with other data sets within the framework of a federated database system based on the functional data model. We have made use of DTD and XML files produced by the BIND project. The DTD provides a specification for information about biomolecular interactions, complexes and pathways, and can be translated semi-automatically to a database schema. The load utility uses metadata derived from this schema to help identify data items of interest when recursively traversing a Prolog tree structure representing the XML data. We also show how derived functions can be used to make explicit those relationships that are present in data sets but which are not fully described in DTD files.

1 Introduction

In recent years there has been a rapid expansion in the quantity and variety of biological data available to researchers. These include data on protein and genome sequences and structure, gene and protein expression, molecular interactions and biological pathways. Scientists' ability to use these data resources effectively to explore hypotheses *in silico* is enhanced if it is easy to ask precise and complex questions that span across several different kinds of data resources in order to find the answer. In our earlier work we have built a federated system in which queries requiring data values from distributed heterogeneous data resources are processed by a prototype program called the P/FDM Mediator [8], which is based on the P/FDM object database system [6]. Tasks performed by the P/FDM Mediator include determining which external databases are relevant in answering users' queries, dividing queries into parts that will be sent to different external databases, translating these subqueries into the language(s) of the external databases, and combining the results for presentation.

A first step in adding a new data resource to our federation is to describe the contents of the new data resource using the functional data model (FDM) [10], which is an example of a semantic data model. Data resources including SRS [5] and ACEDB [4] have been mapped in this way in earlier work. Once this has been done code generators within the P/FDM mediator can produce *ad hoc* queries in the query language used by these systems and these can be dispatched to the remote systems for execution.

E. Rahm (Ed.): DILS 2004, LNBI 2994, pp. 212–219, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Since there wasn't an existing pathway resource to which *ad hoc* queries could be submitted we undertook to load data from XML files provided by another project into a database management system which could ultimately be used as a data resource within a federated system. We decided to use the P/FDM database for storing pathway and interaction data in preference to a database based on the relational model. This was because the XML files are "non-relational", for example these may contain examples of multi-valued attributes and relationships which can be modelled directly using the FDM, but which require a more complex mapping if using the relational model. Further, given the graph structure of interaction networks and the hierarchical part-subpart structure of biomolecular complexes we anticipated being able to benefit from the ability to express recursive queries in Daplex, the FDM's data definition language and query language, against these naturally recursive structures in the data.

In Section 2 we briefly describe some of the candidate pathway data resources that we considered using in this work, and the reasons why BIND was selected. We describe how we generated a functional data model schema from BIND's DTD and how XML files containing pathways, interactions and biomolecular complexes were loaded into the P/FDM database management system in Section 3. In Section 4 we show how the resulting database is queried. Finally, we reflect on our experience in this project, and the lessons that can be learned in designing a data exchange format for pathway and interaction data.

2 Biological Pathway and Interaction Data Sources

Several data resources with information on biological pathways and interactions are under development. These include BIND [2], KEGG [7], MIPS [9] and BioPAX[1]. These differ in the quantity and variety of data available, the file formats used for data distribution, the design of their metadata, and their state of development.

The BIND project [2, 1] has produced a data specification for information about biomolecular interactions, complexes and pathways. This specification was initially defined in ASN.1 [3]. However, tools provided by NCBI have been used to transform this description into an XML DTD[2]. Similarly, data files conforming to the ASN.1 specification have been transformed systematically into XML. Both ASN.1 and XML versions of data files are available from the BIND web site.

KEGG [7] is another biochemical pathway resource. Data from this project are available in KGML (KEGG Markup Language) format and, recently, in XML. However, XML tags have been designed differently in BIND and KEGG. In KEGG the focus is on the graphical presentation of pathway diagrams and the XML tags include layout and presentation elements, whereas in BIND the XML tags relate more directly to biological concepts. BIND and KEGG also use different conventions for naming attributes in their XML DTDs.

[1] http://www.biopax.org/

[2] http://www.ncbi.nih.gov/IEB/ToolBox/XML/ncbixml.txt

The MIPS Comprehensive Yeast Genome Database [9] includes detailed data on protein-protein interactions, pathways and complexes in yeast. However, this resource does not currently provide data in an XML format, and the project web site does not include a description of the resource's metadata.

The Biological Pathways Exchange project (BioPAX) aims to produce a common exchange format for biological pathway data. That project uses frame-like structures with classes and slots for describing pathways. This relatively new project is at the draft release stage, and no data are currently available in this format.

We decided to use data from BIND in our work since this is available in an XML format, and is accompanied by a description in XML DTD that defines tags that are based on biological concepts. Further, since the DTD file is generated in a systematic way from an ASN.1 specification, the entity and attribute names have a consistent form.

3 Implementing an FDM Database from BIND's DTD and XML Files

Our implementation consists of two main software components: (i) a Perl program that reads BIND XML DTD and produces Daplex data definition statements, and (ii) an XML load utility that can load data from XML data files into the P/FDM database.

First we shall consider the task of producing the Daplex schema. Entity types can be recognised as those elements defined the DTD file that have names that do not contain an underscore character and have at least one non-optional attribute. There are two entity types in the extract from the BIND DTD in Figure 1: BIND-Interaction and BIND-object. Scalar attributes and relationships are also identified from the DTD file. Daplex data definition statements generated for this DTD fragment are shown in Figure 2, and these are compiled by the P/FDM system into metadata that are stored in the database.

A serious difficulty when trying to produce a database schema automatically from a DTD file is that the DTD does not contain information about which attributes constitute the keys of the entity classes. We anticipated having to specify these manually, however for many classes a satisfactory candidate key can be formed by taking the union of all non-optional attributes and relationships defined on that class in the DTD. In some cases (including BIND_Interaction) this simple approach produces a super-key from which we have to remove attributes and relationships to form a candidate key. For some others (including BIND_object) the set of non-optional attributes and relationships is insufficient to uniquely identify instances of the class and so one or more optional attributes need to be added to form a candidate key. In one case it was necessary to remove a non-optional relationship and add one optional attribute to form a candidate key. However, for the majority of cases this simple approach yielded a satisfactory key automatically and fewer than ten keys definitions had to be adjusted by hand.

```
<!ELEMENT BIND-Interaction (
               BIND-Interaction_iid ,
               BIND-Interaction_a ,
               BIND-Interaction_b ,
               BIND-Interaction_priv? )>

<!ELEMENT BIND-Interaction_iid ( Interaction-id )>
<!ELEMENT BIND-Interaction_a ( BIND-object )>
<!ELEMENT BIND-Interaction_b ( BIND-object )>
<!ELEMENT BIND-Interaction_priv %BOOLEAN; >
<!ATTLIST BIND-Interaction_priv value ( true | false ) "false" >

<!ELEMENT Interaction-id ( %INTEGER; )>

<!ELEMENT BIND-object (
               BIND-object_short-label ,
               BIND-object_descr? ,
               BIND-object_user-id? )>

<!ELEMENT BIND-object_short-label ( #PCDATA )>
<!ELEMENT BIND-object_descr ( #PCDATA )>
<!ELEMENT BIND-object_user-id ( %INTEGER; )>
```

Fig. 1. An extract from BIND's DTD. Several elements have been omitted to make the figure concise, while still illustrating features described in the text.

```
declare BIND_Interaction ->> entity
declare BIND_object ->> entity

declare iid(BIND_Interaction) -> integer
declare a(BIND_Interaction) -> BIND_object
declare b(BIND_Interaction) -> BIND_object
declare priv(BIND_Interaction) -> boolean

declare short_label(BIND_object) -> string
declare descr(BIND_object) -> string
declare user_id(BIND_object) -> integer

key_of BIND_Interaction is iid
key_of BIND_object is short_label, descr
```

Fig. 2. Daplex data definition statements generated from the DTD fragment shown in Figure 1.

```
<BIND_Interaction>
  <BIND_Interaction-iid>
    <Interaction_id>118</Interaction_id>
  </BIND_Interaction-iid>
  <BIND_Interaction-a>
    <BIND_object>
      <BIND_object-short_label>EGF_EGFR complex</BIND_object-short_label>
      <BIND_object-descr>Epidermal Growth Factor (EGF) bound to Epidermal
Growth Factor Receptor (EGFR)</BIND_object-descr>
      <BIND_object-user_id>0</BIND_object-user_id>
    </BIND_object>
  </BIND_Interaction-a>
  <BIND_Interaction-priv value=""false""/>
</BIND_Interaction>
```

Fig. 3. An extract from the BIND XML data file for interaction number 118.

The XML load utility has been implemented in Prolog — the language in which most of the P/FDM database management system is written. Figure 3 illustrates the structure of data in a BIND XML file. An XML document can be parsed and compiled into a nested Prolog term structure which has the same tree structure as the original XML document. The XML load utility then recursively traverses this tree to find data items that will be loaded into the database. This program first identifies the type of the top-level element in the XML file (near the root of the tree), and retrieves the metadata entry for this type from the P/FDM database. The names of the key attributes for this type are extracted from the metadata entry, and the Prolog program then searches through the tree to find values for these attributes. If values can be found for all of these then a new entity instance is created using the P/FDM update predicate *newentity*. The first two arguments in the call to *newentity* are the entity type name and a list of values for the key attributes. The third argument becomes instantiated to the internal identifier of the entity instance created by the call (e.g. BIND_Interaction(10) in the first call in Figure 4).

The XML load utility then proceeds to look for values for any other attributes and relationships defined on this type. When values are found these are added to the database using the P/FDM update predicate *addfnval* (add function value), which takes the attribute name, the internal identifier of the entity instance and the value of the attribute as its three arguments.

When a relationship is present in the XML data file the load utility must use *addfnval* giving the internal identifier of the related entity instance as the value of the relationship function. If this entity instance is already in the database then this function value can be added immediately. Otherwise, it is necessary to create the related entity instance and add it to the database first, before the relationship function is added. An example of this can be seen in Figure 4, where the BIND object with the key components "EGF_EGFR complex" and "Epidermal Growth Factor (EGF) bound to Epidermal Growth Factor Receptor

```
newentity('BIND_Interaction', [118], 'BIND_Interaction'(10)),
newentity('BIND_object', ['EGF_EGFR complex','Epidermal Growth Factor(EGF)
 bound to Epidermal Growth Factor Receptor (EGFR)'], 'BIND_object'(18)),
addfnval(user_id, ['BIND_object'(18)], 0),
addfnval(a, ['BIND_Interaction'(10)], 'BIND_object'(18)),
addfnval(priv, ['BIND_Interaction'(10)], false)
```

Fig. 4. Instantiated Prolog update goals corresponding to the XML data in Figure 3.

(EGFR)" has to be created before a value for relationship function*a* is added to the database.

We have loaded into P/FDM all BIND pathways that were available in an XML format, together with all related BIND objects and BIND interactions. The XML load utility does not attempt to load all of the data items contained in the original XML documents. Rather, it searches only for data items corresponding to attributes and relationships declared in the schema. Any other data items or elements in the XML data are ignored by the load utility.

4 Querying the Database

The definitions of several derived relationship functions are shown in Figure 5. It is useful to be able to define such functions so that implicit relationships in the database can be made explicit. For example, the BIND Interactions that are related to a BIND Molecular Complex are represented in the XML version as a list of integers that correspond to interaction identifiers, implicitly defining a multi-valued relationship between BIND Molecular Complexes and BIND Interactions. The function definition for *complex_interactions* in Figure 5 makes this relationship explicit. Function *all_complex_subcomplexes* is a recursive function that finds all BIND Molecular Complexes that are part of other BIND Molecular Complexes. This function is used in the Daplex query in Figure 5.

5 Discussion and Conclusions

Providing data in an XML format, even when accompanied by a DTD, is not sufficient to ensure that it will be easy to import these data automatically into a database management system. To facilitate this, it is important that the XML tags are well designed. The systematic naming of DTD elements in BIND is very useful for this purpose. However, it would be helpful if key attributes were declared explicitly in the DTD.

While we have been able to capture most of the information present in the DTD files, we have not been able to capture it all. In particular, we do not map those attributes whose value might be one of several different types since a function in P/FDM must return values of a single type.

Daplex function definitions:

```
define interaction_objects(i in BIND_Interaction)
        ->> BIND_object
  {a(i), b(i)};


define complex_interactions(c in BIND_Molecular_Complex)
        ->> BIND_Interaction
  i in BIND_Interaction such that iid(i) in interaction_list(c);


define complex_objects(c in BIND_Molecular_Complex)
        ->> BIND_object
  interaction_objects(complex_interactions(c));


define complex_subcomplexes(c in BIND_Molecular_Complex)
        ->> BIND_Molecular_Complex
  s in BIND_Molecular_Complex such that
    descr(s) in short_label(complex_objects(c));


define all_complex_subcomplexes(c in BIND_Molecular_Complex)
        ->> BIND_Molecular_Complex
  ( complex_subcomplexes(c)
  union
    all_complex_subcomplexes(complex_subcomplexes(c)) );
```

Daplex query:

```
for each c in BIND_Molecular_Complex
  for each s in all_complex_subcomplexes(c)
    print(mcid(c), descr(c), mcid(s), descr(s));
```

Query results:

```
12971 (EGF_EGFR) dimer complex bound to ATP 12970 (EGF_EGFR) dimer complex
12971 (EGF_EGFR) dimer complex bound to ATP 12966 EGF_EGFR complex
12970 (EGF_EGFR) dimer complex              12966 EGF_EGFR complex
```

Fig. 5. Function definitions and a query.

Derived functions in the functional data model, such as those shown in Figure 5, can make explicit those relationships that are present but which are not fully described in a DTD file. These functions make it more convenient to express queries involving related objects.

The XML load utility is general, and it should be possible to use this with other XML data sets. The Perl program, however, is specific to the style of DTD files produced by the NCBI tools and would have to be modified before it can be used with other DTDs.

The work described in this paper demonstrates that a database schema can be produced from an XML DTD file with only a little manual intervention, and data from XML files can then be loaded into a database management system within which it can be explored using an *ad hoc* query language. This represents a useful first step towards integrating pathway and protein interaction data with other data stored in the P/FDM database management system, or with other data sets accessible within a federated architecture in which a P/FDM database is a constituent data resource.

Acknowledgements

We are grateful for a scholarship from the Swedish Foundation for International Cooperation in Research and Higher Education (S.).

References

[1] Gary D. Bader, Doron Betel, and Christopher W.V. Hogue. BIND: the Biomolecular Interaction Network Database. *Nucleic Acids Research*, 31:248–250, 2003.

[2] Gary D. Bader and Christopher W.V. Hogue. BIND – a data specification for storing and describing biomolecular interactions, molecular complexes and pathways. *Bioinformatics*, 16:465–477, 2000.

[3] O. Dubuisson. *ASN.I Communication Between Heterogeneous Systems*. Morgan Kaufmann Publishers, 2000.

[4] R. Durbin and J. Thierry-Mieg. *Syntactic Definitions for the ACEDB Data Base Manager*, 1992.

[5] T. Etzold and P. Argos. SRS an indexing and retrieval tool for flat file data libraries. *CABIOS*, 9:49–57, 1993.

[6] P.M.D. Gray, K.G. Kulkarni, and N.W. Paton. *Object-Oriented Databases: a Semantic Data Model Approach*. Prentice Hall Series in Computer Science. Prentice Hall Int. Ltd., 1992.

[7] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28:27–30, 2000.

[8] G. J. L. Kemp, N. Angelopoulos, and P. M. D. Gray. Architecture of a Mediator for a Bioinformatics Database Federation. *IEEE Transactions on Information Technology in Biomedicine*, 6:116–122, 2002.

[9] H.W. Mewes, D. Frishman, U. Güldener, G. Mannhaupt, K. Mayer, M. Mokrejs, B. Morgenstern, M. Münsterkötter, S. Rudd, and B. Weil. MIPS: a database for genomes and protein sequences. *Nucleic Acids Research*, 28:31–34, 2002.

[10] D.W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, 1981.