A Schema-based Approach to Building a Bioinformatics Database Federation

Graham J.L. Kemp[1], Nicos Angelopoulos[1,2] and Peter M.D. Gray[1]

[1]Department of Computing Science,
University of Aberdeen, King's College,
Aberdeen, Scotland, UK, AB24 3UE

[2]Department of Molecular and Cell Biology,
University of Aberdeen, Foresterhill,
Aberdeen, Scotland, UK, AB25 2ZD

Abstract

Developments in our ability to integrate and analyse data held in existing heterogeneous data resources can lead to an increase in our understanding of biological function at all levels. However, supporting ad hoc queries across multiple data resources and correlating data retrieved from these is still difficult. To address this, we are building a mediator based on the functional data model database, P/FDM, which integrates access to heterogeneous, distributed biological databases, while making use of existing search engines and indexes, without infringing on the autonomy of the underlying databases. Central to our design philosophy is the use of schemas. We have adopted a federated architecture with a five-level schema, arising from the use of the ANSI-SPARC three-level schema to describe both the existing autonomous data resources and the mediator itself. We describe the use of mapping functions and list comprehensions in query splitting, producing execution plans, code generation and result fusion. We give an example of cross-database querying involving data held locally in P/FDM systems and external data in SRS.

1 Introduction

The internet is an increasingly important research tool for scientists working in biotechnology and the biological sciences, and many collections of biological data can be accessed via the World-Wide Web. Some on-line data resources provide search facilities to enable scientists to find items of interest in a particular database more easily. However, working interactively with an internet browser is extremely limited when one wants to ask complex questions involving related data held at different locations and in different formats since one must formulate a series of data access requests, run these against the various databanks and databases, and then combine the results retrieved from the different sources. This is both awkward and time-consuming for the user.

To streamline this process, we are developing a federated architecture and *mediator* to integrate access to heterogeneous, distributed biological databases. The suitability of a federated multi-database approach for integrating biological databases is advocated by Robbins [15] and also proposed by Karp [9]. The spectrum of choices for data integration is summarised in Figure 1. In order to get the desired federated information infrastructure we believe, with Robbins, that we do not require the adoption of a common hardware platform or vendor DBMS, but we do need a "shared data model across participating sites".

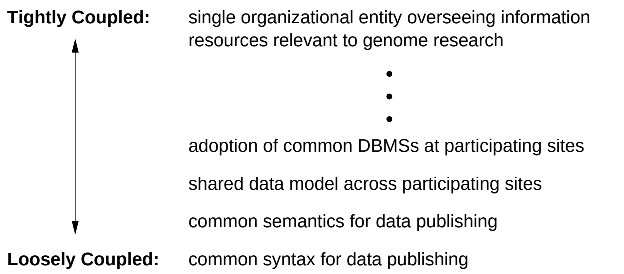

Figure 1: Continuum from tightly coupled to loosely coupled systems, involving multiple databases (from [15]).

Our work is based around the P/FDM object database system, which has been developed at Aberdeen for storing and integrating protein structure data. In this paper we discuss the different levels of schemas present in our federated architecture, and we relate these to the classic ANSI-SPARC three layer model. We describe the implementation of our mediator software with particular emphasis on the use of list comprehensions in transforming queries against a conceptual schema into queries expressed against schemas in other layers of the system, with an example that accesses external bioinformatics databases. Finally we discuss the role of mapping functions and wrappers in our architecture, and compare these with similar software modules in other systems.

2 Data resources in the federation

P/FDM [6] is an object database management system that is based on a semantic data model: the Functional Data Model (FDM) [17]. The basic concepts in the P/FDM database are entities and functions. Entities are used to represent conceptual objects, while functions represent the properties of an object. Functions are used to model both scalar attributes and relationships. Functions may be single-valued or multi-valued, and their values can either be stored or computed on demand. Entity classes can be arranged in subtype hierarchies, with subclasses inheriting the properties of their superclass, as well as having their own specialised properties. P/FDM is being used at Aberdeen to store data on special classes of proteins, but it is increasingly used for data integration. The *ad hoc* query language used in P/FDM is an implementation of Daplex. Daplex queries are normally processed in P/FDM by first being compiled into an internal Intermediate Code ("ICode") [5] which is a Prolog term structure that resembles a list comprehension. The ICode form of a query is then optimised, and a code generator translates the ICode into Prolog code for execution. As we shall see in Section 4, ICode plays an important role in processing multi-database queries within the mediator. Remote access to a P/FDM database is provided through a CORBA server [11]. Our mediator, which is described in Section 4, is implemented as an extension of the P/FDM database management system.

The Sequence Retrieval System (SRS) [4] maintains indexes relating entries in one flat file databank (e.g. the SWISS-PROT protein sequence databank, or the BRENDA enzyme databank) to entries in others. Thus, SRS maintains a large network of related databank entries. SRS provides search tools that enable users to retrieve particular entries or to find the accession codes of entries that satisfy specified criteria, e.g. entries with fields that match a given pattern string, or fields with numeric values in a given range. SRS also provides "link" operators that enable the user to follow cross references and find all entries in one databank with corresponding entries in another. These can even follow a chain of cross references where there is no "direct" link between the two. The SRS system has its own query language that can be accessed via an HTTP server.

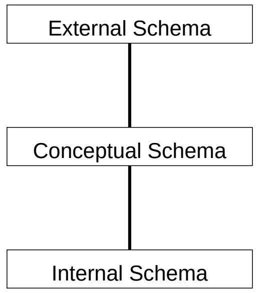

Figure 2: ANSI-SPARC schema architecture

3 Schemas in the federation

The design philosophy of our architecture can be illustrated with reference to the *three-level schema* that was proposed by the ANSI-SPARC standards working party [1]. The three-level schema is illustrated in Figure 2. This promotes data independence by demanding that database systems be constructed so that they provided both logical and physical data independence. Logical data independence provides that the conceptual data model must be able to *evolve* without changing external application programs. Only view definitions and mappings may need changing, for example to replace access to a stored field by access to a derived field calculated from others in the revised schema. Physical data independence allows us to refine the internal schema for improved performance, without needing to alter the way queries are formulated.

The clear separation between schemas at different levels helps us in building a database federation in a modular fashion. In our architecture we see the ANSI-SPARC three-level schema in two situations: in each of the individual data resources, and in the mediator itself.

First, let us consider an external data resource. The resource's *conceptual schema* (which we call C_R) describes the logical structure of the data contained in that resource. If the resource is a relational database then this will include information about table names and column names, and type information about stored values. With SRS [4], it is the databank names and field names. These systems also provide a mechanism for querying the data resource

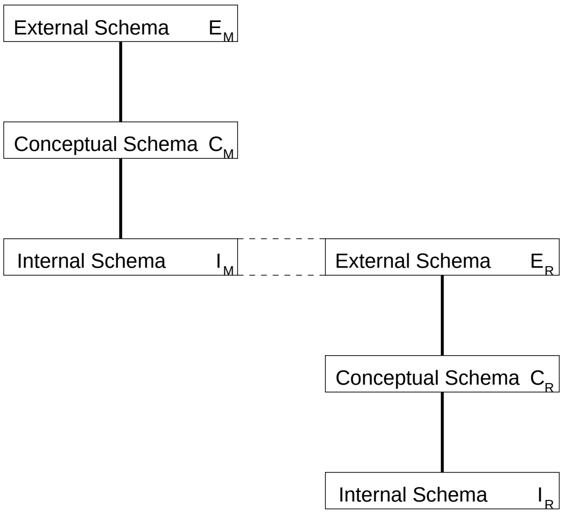

Figure 3: ANSI-SPARC schema architecture describing the mediator (left) and an external data resource (right).

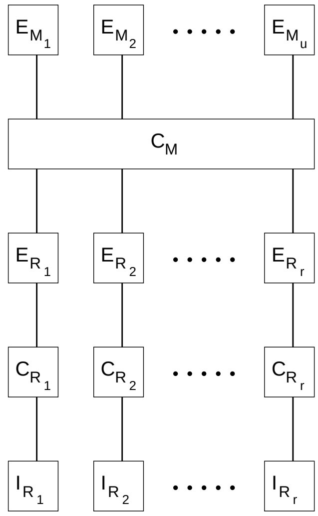

Figure 4: Schemas in a database federation.

in terms of the table/class/databank names and column/tag/attribute/field names that are presented in the conceptual schema.

The *internal schema* (or *storage schema*, which we call I_R) contains details of allocation of data records to storage areas, placement strategy, use of indexes, set ordering and internal data structures that impact on efficiency and implementation details [6]. In this paper we do not concern ourselves with the internal schemas of individual data resources. The mapping from the conceptual schema to the internal schema has already been implemented by others within each of the individual resources that we use, and we assume that this has been done to make best use of the resources' internal organisation.

A resource's *external schema* (which we call E_R) describes a view onto the data resource's conceptual schema. At its simplest, the external schema could be identical to the conceptual schema. However, the ANSI-SPARC model allows for there to be differences between the schemas at these layers so that different users and application programmers can each be presented with a view that best suits their individual requirements and access privileges. Thus, there can be many external schemas, each providing users with a different view onto the resource's conceptual schema. A resource's external, conceptual and internal schemas are represented on the right hand side of Figure 3.

We can also use the ANSI-SPARC three-layer model in describing the mediator that is central to our database federation, and this is shown on the left hand side of Figure 3. The mediator's conceptual schema, C_M, describes the content of the data resources that are members of the federation, including the semantic relationships that hold between data items in these resources. We also refer to this as the federation's integration schema. We have chosen to express this schema using the functional data model. As far as possible, the C_M is designed based on the semantics of the domain, rather than consideration of the actual partitioning and organisation of data in the external resources. For example, different attributes of the same conceptual entity can be spread across different external data resources, and subclass-superclass relationships between entities in the conceptual model of the domain might not be present explicitly in the external resources [8].

We cannot expect scientists to agree on a single schema. Different scientists are interested in different aspects of the data, and will want to see data structured in a way that matches the concepts, attributes and relationships in their own personal model. This is made possible by following the ANSI-SPARC model; the principle of logical data independence means that the system can provide different users with different views onto the integration schema. We shall use E_M to refer to an external schema presented to a user of the mediator.

A vital task performed by the mediator is to map between C_M and C_R. To facilitate this process, we introduce another schema layer which, in contrast to C_M, *is* based on the structure and content of the external data resources. This schema is internal to the mediator, and is referred to as I_M. The mediator needs to have a view onto the data resource that matches this internal schema, thus I_M and E_R should be the same. In our system we have chosen to use the functional data model to represent I_M/E_R. Having the same data model for C_M and I_M/E_R brings advantages in processing multi-database queries, as will be seen in Section 4.

Redrawing Figure 3, the situation where there are u different external schemas presented to users, and there are r data resources, the relationship between schemas in the federation is as shown in Figure 4. The five-level schema shown here is similar to that described by Sheth and Larson [16].

4 Mediator architecture

The role of the mediator is to process queries expressed against the federation's integration schema (C_M). The mediator has five main components: a Daplex compiler, an ICode rewriter, query splitter, code generators and a result fuser. These components are shown in Figure 5.

The mediator holds metadata describing the integration schema and also the external schemas of each of the federation's data resources (E_R). In P/FDM, metadata takes the form of Prolog clauses that are compiled from high level schema descriptions.

The Daplex compiler translates queries expressed against the integration schema into ICode that also refers to entities, attributes and relationships in the integration schema. The ICode rewriter then performs semantic and syntactic optimisations, and modifies the original ICode so that it now refers to entities, attributes and relationships in the data resources' external schemas. Thus, query transformation from C_M to E_R is performed by the ICode rewriter. This is done by applying mapping function that are implemented as rewrite rules. The query splitter performs code reordering, code grouping and variable dependency analysis and cost estimation. After this step, the ICode has been separated into a number of "chunks", each referring to a different actual data resource. Each ICode chunk is sent to one of several code generators. These translate ICode into queries that are executable by the remote databases, transforming query fragments from E_R to C_R. The remote queries are wrapped with Prolog code which oversees their sending, and reception of results. The result fuser provides a synchronisation layer, combining results retrieved from external databases.

In this paper, queries are expressed directly against an integration schema (C_M). However, these could al-

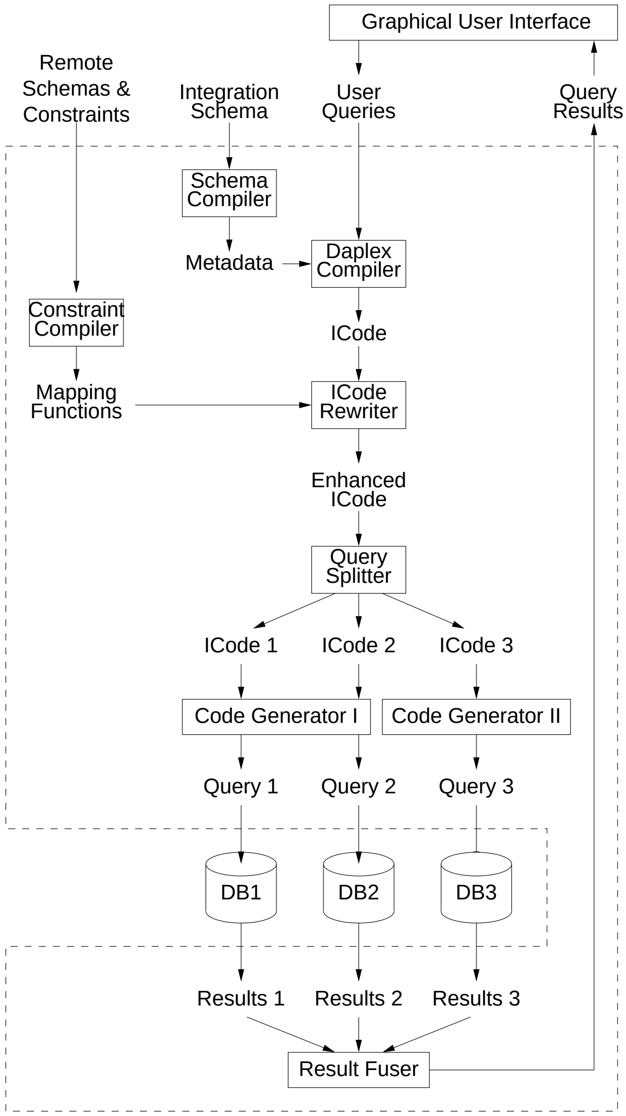

Figure 5: Mediator architecture. The components of the mediator are shown inside the dashed line. The external data resources have not been modified.

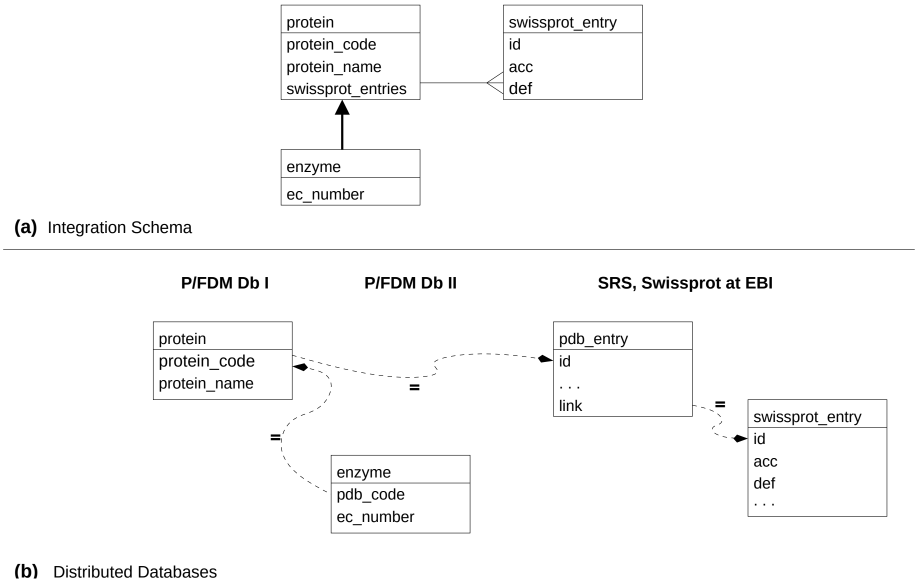

Figure 6: Integration schema and distributed databases (P/FDM and SRS)

```
for each e in enzyme such that ec_number(e) = "1.1.1.1"
  for each s in swissprot_entries(e)
print(protein_name(e), def(s), acc(s));
```

Figure 7: Daplex query expressed against an integration schema.

$$
\begin{array}{ll}
[\ \mathrm{V6},\ \mathrm{V4},\ \mathrm{V3}\ | & \mathrm{V1} \leftarrow \mathrm{enzyme}\ ;\\
& \mathrm{V2} \leftarrow \mathrm{swissprot\_entries(V1)}\ ;\\
& \mathrm{V3} = \mathrm{acc(V2)}\ ;\ \mathrm{V4} = \mathrm{def(V2)}\ ;\\
& \mathrm{V5} = \mathrm{ec\_number(V1)}\ ;\ \mathrm{V5} = \text{“1.1.1.1”}\ ;\\
& \mathrm{V6} = \mathrm{protein\_name(V1)}\ ]
\end{array}
$$

Figure 8: ICode corresponding to the query in Figure 7.

```
foreign( swissprot_entries, [protein], srs_sprot, entity, KeyICode, ebi_db ) :-
     KeyICode = (V1,V2,[V3,V4,V5,V6],
                  [
                    restrict(ebi_db:id,[ebi_db:srs_sprot],[V2],V6),
                    restrict_subquery(some(V5),
                      [ generate(ebi_db:pdb_entry,V3),
                        restrict(ebi_db:id,[ebi_db:pdb_entry],[V3],V4),
                        restrict(protein_code,[protein],V1,V4),
                        restrict(ebi_db:link,[ebi_db:pdb_entry],[V3],V5) ],
                      [ expression([],[V6,V5],expr(=,V6,V5)) ]
                     )
                  ] ).
```

Figure 9: Mapping function used to expand the relationship swissprot_entries in the integration schema into ICode that refers to data held at the EBI.

ternatively be expressed against an external schema presented by the mediator (E_M). If the external schema is also an FDM schema then an additional layer of mapping functions would be required to translate the query from E_M to C_M.

5 Example

A prototype mediator has been used to combine access databanks at the EBI via an SRS server [4] and (remote) P/FDM test servers. This example, using a mini-integration schema, illustrates the steps involved in processing multi-database queries.

In this example, three different databases are viewed through a unifying integration schema, which is shown in the upper part of Figure 6. There are three classes in this schema: *protein*, *enzyme* and *swissprot_entry*. A function representing the enzyme classification number (ec_number) is defined on the class *enzyme*, and enzymes inherit those functions that are declared on the superclass *protein*. Each instance of the class *protein* can be related to a set of *swissprot_entry* instances.

The lower part of Figure 6 shows the actual distribution of data across the three databases. Db I is a P/FDM database that contains the codes and name of proteins. Db II is also a P/FDM database, and contains the protein code (here called pdb_code) and enzyme classification code of enzymes. To identify SWISS-PROT entries at the EBI that are related to a given protein instance we must first identify the Protein Data Bank (PDB) entry whose id matches the protein code and then follow further links to find related SWISS-PROT entries. Relationships between data in remote databases can be defined by conditions that must hold between the values of the related objects. Constraints on identifying values are represented by dashed arrows in Figure 6.

Figure 7 shows a Daplex query expressed against this integration schema. This query prints information about enzymes that satisfy certain selection criteria, and their related SWISS-PROT entries. Figure 8 shows a "pretty-printed" version of the ICode that is produced when this query is compiled. This ICode is then processed by the query splitter, producing ICode that will be turned into queries that will be sent to the three external data resources (Figure 10). Note that the variable $V1$ is common to all three query fragments. Values for this variable that are retrieved from P/FDM Db II are used in constructing queries to be sent to the other data resources.

```
ICode for P/FDM Db II:
  [ V1 |   V2 ← enzyme ; V3 = ec_number(V2) ;
           V1 = pdb_code(V2) ; V3 = "1.1.1.1" ]

ICode for P/FDM Db I:
  [ V6 |   V4 ← protein ; V5 = protein_code(V4) ;
           V6 = protein_name(V4) ; V1 = V5 ]

ICode for SRS:
  [ V7, V8 |   V9 ← pdb_entry ; V10 = id(V9) ;
               V1 = V10 ; V11 ← link(V9) ;
               V12 ← swissprot_entry ;
               V13 = id(V12) ; V7 = def(V12) ;
               V8 = acc(V12) ; V13 in V11 ]
```

Figure 10: ICode sub-queries against the actual data resources that need to be accessed in answering the query in Figure 7.

In the example, the class *protein* is related to the class *swissprot_entry* in the integration schema by a multi-valued relationship function called *swissprot_entries*. The mapping function given in Figure 9 is used in transforming queries that contain this relationship into "enhanced" ICode that refers to the external schemas of the actual data resources. This code shows an actual example of P/FDM ICode. Mapping functions such as this can be compiled from high level declarative rewrite rules, and do not have to be written by hand.

6 Discussion

The use of mediators was originally proposed by Wiederhold [18] and became an important part of the Knowledge Sharing Effort architecture [13]. Examples of such intelligent information-seeking architectures are Infosleuth [2] and KRAFT [7]. In this architecture, the mediator can run on the client machine, or else be available as middleware on some shared machine, while the wrapper is on the remote machine containing the knowledge source. The idea behind this is that existing knowledge sources can evolve their schemas, yet present a consistent interface to the mediator by suitable changes to the wrapper. For this purpose the wrapper may be as simple as an SQL view, or it may be more complex, involving mapping of code. In any case, the site is able to preserve some local autonomy. Other mediators do not have to worry how the site evolves internally. Also new sites can join a growing network by registering themselves with a facilitator. All the mediator needs to know is how to contact the facilitator, and that any knowledge sources the facilitator recommends will conform to the integration schema.

In this paper we describe an alternative architecture, where the wrappers reside with the mediator. This has the advantage that there is no need to get the knowledge source to install and maintain custom-provided wrapper software.

In our architecture, as shown in Figure 5, the wrappers are used to generate code in a different query language or constraint language. Thus they are used in two directions. In one direction they map queries or constraints into a language that can be used directly at the knowledge source. This can be crucial for efficiency, by allowing one to move selection predicates closer to the knowledge source in a form that is capable of using local indexes etc. This can have a very big effect with database queries, because it saves bringing many "penny packets" of data back through the interface, only to be filtered and rejected on the far side [10]. In the other direction, wrappers are used to map data values, for example by using scaling factors to change units, or by using a lookup table to replace values by their new identifiers.

Note that we are not advocating building a so-called *global* integration schema. This has often been criticised on the grounds that attempts to map every single concept in one all-embracing schema is both laborious and never-ending. Instead we visualise an incrementally growing integration schema, driven by user needs. Ideally the schema would be built interactively using a GUI and rules that suggest various mappings, as proposed by Mitra *et al.* [12] in their ONION system for incremental development of ontology mappings. The crucial thing to realise is that the integration schema represents a virtual database, which allows it to evolve much more easily than a physical database.

An alternative to a federated approach is to drive the system from a classification ontology, as in the TAMBIS approach [14]. They maintain a classification hierarchy for proteins with many pre-defined subclasses (such as guppy proteins, or enzymes with zinc cofactors). For each such subclass they maintain a recipe for iterating over its members, expressed in the CPL language [3]. They code-generate CPL and say "we view CPL as a wrapping mechanism tightly coupled with convenient language facilities for accumulating and transmitting results from different sources". Thus they use the term "wrapper" as a kind of black box interface to remote sources, whereas we are considering it as a processing step in the context of various levels of schema mapping.

In TAMBIS, it is easy to get filtered subsets of the subclasses, by putting restrictions on values of various attributes. This works well for queries that closely follow the framework of the hierarchy and return members of specific subclasses, and so their GUI makes it very easy to build such queries. What we gain, by contrast, from the federated approach is a full *ad hoc* query language, with the freedom to follow any access path we choose through the virtual federated database. Likewise we can gather compound results which include values gathered from a number of related entities. These are of particular interest where one is searching for unsuspected connections and chains of connections between various biological components, rather than following those that are documented in the classification hierarchy.

7 Conclusions

We have described the role of schemas in a bioinformatics database federation, and the implementation of a system that supports *ad hoc* querying across multiple heterogeneous data resources.

Intermediate Code (ICode) provides a high level, declarative representation for queries. We have implemented ICode as Prolog term structures, and our mediator make extensive use of Prolog's powerful pattern matching and list processing capabilities in performing query transformation. The examples presented in this paper demonstrate how ICode can be manipulated easily by the mediator to rewrite and split queries expressed against an integration schema into sub-queries that refer to the external schemas of distributed databases.

Bioinformatics faces a "crisis of data integration" [15], which we believe is best addressed through federations that allow their constituent databases to develop autonomously and independently. The existence of schemas at different levels, as shown in Section 3, makes apparent the requirements for query transformation in a mediator in a database federation. This results in a modular design for the mediator that enables the federation to evolve incrementally.

Acknowledgements

This work is supported by a grant from the BBSRC/EPSRC Joint Programme in Bioinformatics (Grant Ref. 1/BIF06716).

References

[1] ANSI. Interim Report of the ANSI/X3/SPARC Study Group on Data Base Management Systems. *ACM SIGFIDET*, 7:3–139, 1975.

[2] R.J. Bayardo, B. Bohrer, R.S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments (Experience Paper). In J. Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 195–206. ACM Press, 1997.

[3] P. Buneman, S.B. Davidson, K. Hart, G.C. Overton, and L. Wong. A Data Transformation System for Biological Data Sources. In U. Dayal, P.M.D. Gray, and S. Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pages 158–169. Morgan Kaufmann, 1995.

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

[5] P.M.D. Gray, S.M. Embury, K.Y. Hui, and G.J.L. Kemp. The Evolving Role of Constraints in the Functional Data Model. *J. Intelligent Information Systems*, 12:113–137, 1999.

[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] P.M.D. Gray, A.D. Preece, N.J. Fiddian, W.A. Gray, T.J.M. Bench-Capon, M.J.R. Shave, N. Azarmi, M. Wiegand, M. Ashwell, M. Beer, Z. Cui, B. Diaz, S.M.Embury, K.Hui, A.C.Jones, D.M.Jones, G.J.L.Kemp, E.W.Lawson, K.Lunn, P.Marti, J.Shao, and P.R.S.Visser. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases. In R.R. Wagner, editor, *Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, pages 682–691. IEEE Computer Society Press, 1997.

[8] S. Grufman, F. Samson, S.M. Embury, and P.M.D. Gray. Distributing Semantic Constraints Between Heterogeneous Databases. In A. Gray and P.-Å. Larson, editors, *Proceedings: 13th International Conference on Data Engineering*, pages 33–42. IEEE Computer Society Press, 1997.

[9] P.D. Karp. A Vision of DB Interoperation. Meeting on the Interconnection of Molecular Biology Databases, 1995.

[10] G.J.L. Kemp, J.J. Iriarte, and P.M.D. Gray. Efficient Access to FDM Objects Stored in a Relational Database. In D.S. Bowers, editor, *Directions in Databases: Proceedings of the Twelfth British National Conference on Databases*, pages 170–186. Springer-Verlag, 1994.

[11] G.J.L. Kemp, C.J. Robertson, P.M.D. Gray, and N. Angelopoulos. CORBA and XML: Design Choices for Database Federations. In B. Lings and K. Jeffery, editors, *Proceedings of the Seventeenth British National Conference on Databases*, pages 191–208. Springer-Verlag, 2000.

[12] P. Mitra, G. Weiderhold, and M. Kersten. A Graph-Oriented Model for Articulation of Ontology Interdependencies. In C. Zaniolo, P.C. Lockerman, M.H. Scholl, and T. Grust, editors, *Advanced in Database Technology – EDBT 2000*, pages 86–100. Springer-Verlag, 2000.

[13] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senatir, and W.R. Swartout. Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):36–56, 1991.

[14] N.W. Paton, R. Stevens, P. Baker, C.A. Goble, S. Bechhofer, and A. Brass. Query Processing in the TAMBIS Bioinformatics Source Integration System. In *11th International Conference on Scientific and Statistical Database Management, Proceedings*, pages 138–147. IEEE Computer Society Press, 1999.

[15] R. J. Robbins. Bioinformatics: Essential infrastructure for global biology. *J. Comp. Biol.*, 3:465–478, 1996.

[16] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, 22:183–236, 1990.

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

[18] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.