Practical Inter-Operation of CAD Tools Using a Flexible Procedural Interface*

Zahir Moosa Nick Filer Mike Brown
Department of Computer Science, University of Manchester
Oxford Road, Manchester, UK, M13 9PL

J Heaton J.Pye
ICL, Wenlock Way, West Gorton, Manchester, UK, M12 5DR

Abstract

This paper addresses the problem of semantic heterogeneity between data representations with particular emphasis on CAD tool data representations. The combination of powerful mapping operations and a flexible procedural interface are proposed as a possible solution to this problem. A practical application of the inter-operation of data representations is used to illustrate the techniques. The data representations used are the ICL COT data format [1] and the TRACKER data format [2].

1 Introduction

Inter-operability is fast becoming a major issue in information technology. It may be regarded as an essential facility for controlling, preserving, exchanging and utilising information between two or more tools. Inter-operability, of CAD tools in particular, is complicated by the need to frequently adapt to changes in the design representations used by the individual tools. This has led to the advent of software environments called frameworks [3, 4, 5, 6, 7] which help reduce the time and cost needed to develop or modify a component in response to changing end user requirements. Inter-operability is enabled by integrating tools into the framework environment [8].

The need for tool inter-operability cannot be underestimated. Data representations (conceptual models) of different tools are typically developed independently, which means that they are usually incompatible with one another (semantic heterogeneity). In addition, conceptual models are often in a state of continuous evolution, in order to support changing tool requirements, so that different versions of the same tool are likely to have incompatible models.

As a consequence of semantic heterogeneity, equivalent information may have completely different naming, typing, level of abstraction and underlying structures in two tool-specific conceptual models [9]. However, increasingly sophisticated user requirements dictate that the users should no longer be required to manage low level details of sending and receiving data in different representations in a number of information sources. An ideal scenario would be one in which a user is provided global, integrated access to multiple data representations with a single, relatively simple request. This paper describes the efforts made towards making this scenario a practical reality.

In [8], a fine-grained procedural interface for supporting tool integration is described. The procedural interface (GPIC) is based on a minimal set of generic procedures. Flexibility is obtained by providing additional interface procedures to interpret conceptual models of data. This paper describes a practical application of the procedural interface in bridging underlying semantic differences in the conceptual models of multiple tools in a CAD framework.

This paper is organised as follows: Section 2 briefly reviews the principles of GPIC and its underlying conceptual model CMDL (CAD Meta Data Language). The need for mappings to support tool inter-operability is introduced. Section 3 gives a practical example of inter-operation of data representations using mappings and GPIC. The representations used are the ICL COT data format [1] and the TRACKER data format [2]. The paper is concluded with a comparison to similar work and an outline of future work.

2 Background

Two components have been identified as the key requirements for the definition of a procedural interface to a data repository [10]: a conceptual model describing the organisation of data in the repository, and the access procedures linking the environment to the repository. Since the nature and content of the data stored in a CAD repository is constantly evolving, it is necessary to define conceptual model components that can survive these changes. Examples of domain independent concepts include objects, at-

*Some of the research described here has been carried out with support from the SERC in the UK. Current research is supported by ESPRIT project 7364 JESSI-COMMON-FRAME.

tributes and relations. In contrast, examples of domain dependent concepts include nets, cells and ports. A domain independent conceptual model ensures a relatively stable interface that is independent of the complexity or changes in the application domain. A comparison with other styles of interface (e.g CFI DR [11]) is given in [8, 10, 12] and is beyond the scope of this paper.

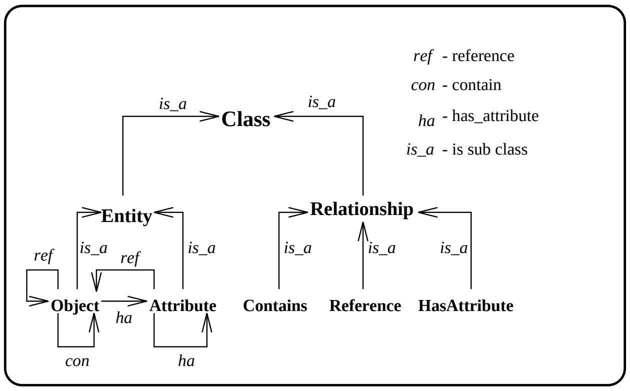

Figure 1: Principles of CMDL

The GPIC interface is based on an underlying conceptual modelling notation called CMDL. The principles of CMDL are summarised in figure 1. The interface has two layers of access procedures: meta-data access and data access. Design data may be regarded as occurrences of concepts defined by a conceptual model whereas meta-data may be regarded as occurrences of meta-concepts defined in a meta-conceptual model. This specialisation of concepts is further illustrated in figure 2.

A major requirement for tool inter-operability is that tools must agree on the meaning of exchanged data. The manifestation of this agreement is in the form of a set of mappings between the concepts of the respective tools. These mappings describe removal, addition and structural changes to entities within a schema as well as transformations on the individual attributes. The need for mappings in the CAD domain has already been recognised [8, 13, 14]. Extension of GPIC with mappings has many potential applications for CAD frameworks including:

- Mappings between different views of the same data,
- Mappings between data and semantic information for the same object,
- Mappings between different versions of conceptual models,
- Mappings between different CAD standards.

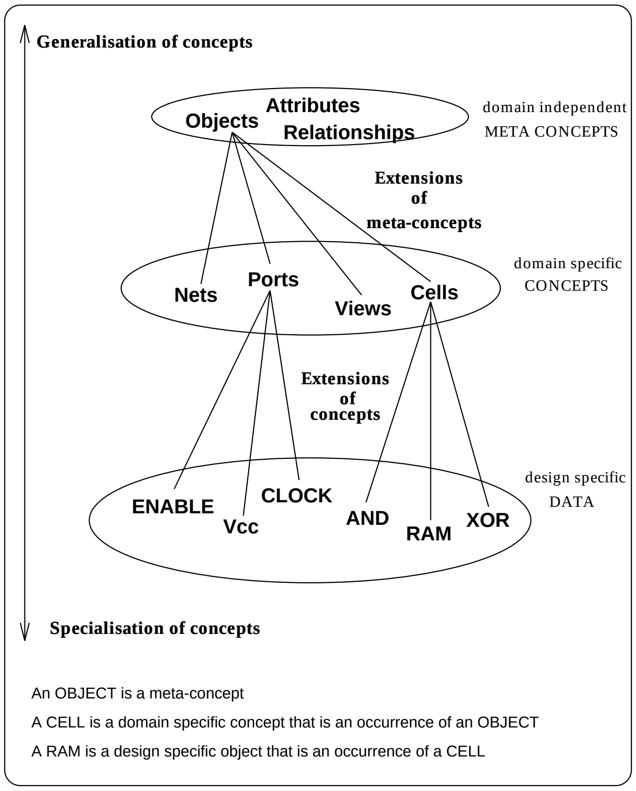
Generalisation of concepts
Objects
Attributes
Relationships
domain independent
META CONCEPTS
Extensions of meta-concepts
Nets
Ports
Views
Cells
domain specific
CONCEPTS
Extensions of concepts
ENABLE
Vcc
CLOCK
AND
RAM
XOR
design specific
DATA
Specialisation of concepts
An OBJECT is a meta-concept
A CELL is a domain specific concept that is an occurrence of an OBJECT
A RAM is a design specific object that is an occurrence of a CELL

Figure 2: Hierarchy of Models

3 A Practical Integration Example

This section describes the practical efforts made towards integrating the ICL COT [1] data format with the TRACKER routing data format [2]. The ICL COT format is a fixed field text representation used to describe ICL DAX data. DAX is the ICL design system for PCB and chip design. The TRACKER format is also a fixed field text representation used to describe connectivity and tracking information for a routing tool. Figure 3 illustrates how the respective environments interface to each other using GPIC as a central data repository.

The interface requires conceptual models of ICL-COT and TRACKER along with their associated mappings, stored as data within the GPIC repository. Reader/writers for both these representations have been constructed so as to avoid making modification to existing tools (though it is quite feasible for applications to be integrated directly through GPIC). The readers and writers also use GPIC to build a set of parsing/writing rules for each data object in the representation. These rules consist of a list of legal attribute classes that have to be defined for a data object. At run-time, the meta-data routines of GPIC are used to retrieve the terminal types (e.g string, character, integer ...) of a data object by interpreting the respective conceptual models. The use of GPIC in this manner allows the conceptual

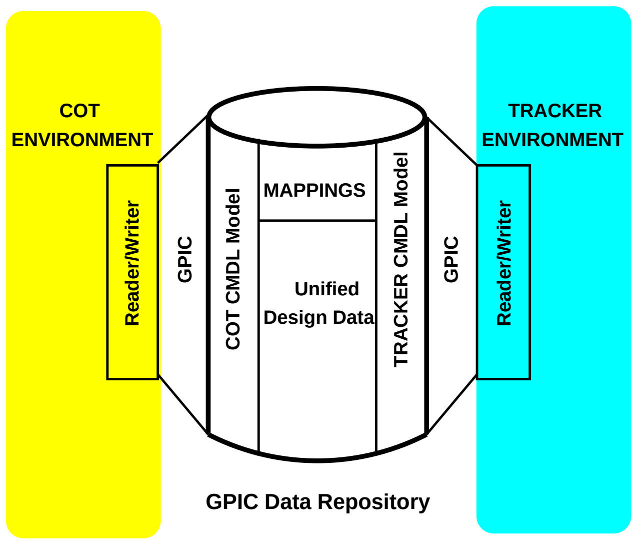

Figure 3: Integration of ICL-DAX and TRACKER environments

model (and hence the representation rules) to evolve without the need to change the readers or writers.

The problem that has to be addressed in this case is that both environments have essentially a local view of the data in the repository. It is the responsibility of the GPIC interface to solve conflicts between the views and present an integrated view to each environment. This is achieved by using mappings to ensure consistent names, values and formats. Thus, the integrated system makes heterogeneous components appear homogeneous by hiding the differences amongst them. The following section gives a detailed account on how mappings can be used to solve semantic heterogeneity between data representations.

3.1 Mapping

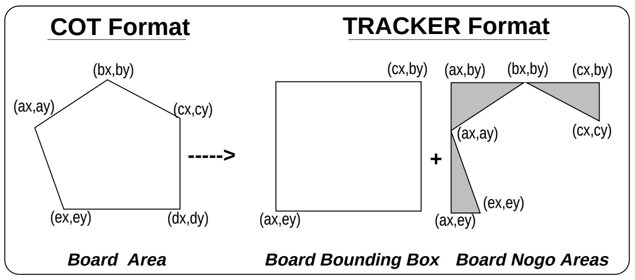

Figure 4: An Example of Semantic Heterogeneity Between COT and the TRACKER Formats

As already noted, the problem of semantic heterogeneity refers to the storage of related information in incompatible data representations. The application described here concerns the problem that the information required for running TRACKER is stored as COT data, but the COT format is not interpretable by TRACKER. A simple example of this is shown in figure 4; in COT a board area is generally represented as an arbitrary polygon, whereas TRACKER requires the board area to be represented as a bounding rectangle and an appropriate collection of *no-go areas*.

The proposed general solution to semantic heterogeneity is to support mapping facilities in GPIC. Mappings are directional relationships from a *source* schema to a *target* schema. In general, the definition of a mapping requires careful consideration of the structural differences and computational transformations between two representations. There are two crucial issues involved in the structural consideration. Firstly, *equivalence relationships* must be established between components of the two schemes; equivalence being loosely defined as representing the same real world concept, without constraining the way in which the information is represented. Non-equivalent source components are deleted and non-equivalent target components are added in a mapping. Secondly, the differences in the detailed structure of the two schemas must be determined. This difference is manifest by the existence of one-to-many or many-to-one relationships between components of the two schemas, even though the real world concepts being represented in the source and target schemas may be the same. This representation of identical information by different representational structures will be referred to as *non-isomorphism*.

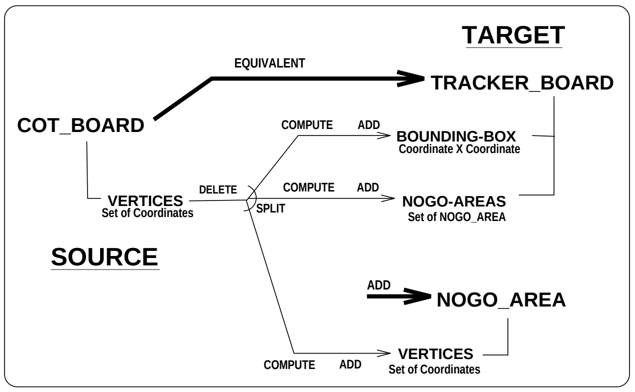

Figure 5: Mapping from COT Format to TRACKER Format

The explicit specification of the structure of a mapping is achieved by insertion of various structural operators (see figure 5) that establish unique paths for deriving target entities and attributes from the source schema. For example, in figure 5, the *split* operator generates a unique path to each of the target attributes. The definition of a mapping in terms of the structural operators is important meta-data that should be associated with the respective schemas within the framework. This meta-data will provide valuable information for the maintenance of mapping relationships between schemas, for example providing a log of the changes between different version of an evolving schema.

In addition to structural operators, a *compute* operator may be associated with each mapping path. The compute operator represents a path-specific transformation of the actual data values stored for a source schema into the required values in the target schema. Hence, the compute operator captures the deep semantics of the mapping, while the structural operators establish constraints on the values transformed by the compute operator. This methodology for describing mappings is illustrated below by a worked example.

```
EQUIVALENT(COT_BOARD,ROUTER_BOARD);
ADD(NOGO_AREA);

DELETE(COT_BOARD.VERTICES);
ADD(ROUTER_BOARD.BOUNDING_BOX);
ADD(ROUTER_BOARD.NOGO_AREAS);
ADD(NOGO_AREA.VERTICES);

SPLIT(COT_BOARD.VERTICES,[A1,A2,A3]);
/* NB, A1, A2 and A3 are copies of
                COT_BOARD.VERTICES */

COMPUTE([A1],
      [ROUTER_BOARD.BOUNDING_BOX]);
COMPUTE([A2],
        [ROUTER_BOARD.NOGO_AREAS]);
COMPUTE([A3],[NOGO_AREA.VERTICES]);
```

Figure 6: An Example of Mapping Pseudo-Code

Figures 5 and 6 depict the structural specification of the mapping from the COT to TRACKER formats for the representation of the routing area of a board. It can be seen that there exists an equivalence between the source and target entities representing the concept of a board area. In addition, the TRACKER format introduces a new entity representing the concept of a no-go area for routing. There exists non-isomorphism in that all attributes in the target schema are derivable from the single source schema attribute; the list of coordinates representing the COT board's vertices. A copy of the original source attribute value (variables A1, A2 and A3 in figure 6) is passed independently along each one of the paths in the mapping and can be transformed independently of the other paths.

As noted above, the function represented by the compute operator is unique to each path, though the structural constraints may be used to provide a template for that function (e.g, identifying the type and source of its parameters). Example pseudo-code for one of the compute functions for the board area mapping is depicted in figure 7, showing how the required pair of coordinates representing the bounding box of the routing area, as used by TRACKER, can be extracted from the list of coordinates representing the vertices of the board area in COT. In the current implementation of GPIC, this type of mapping function is represented as handwritten C code, optionally associated with the CMDL relationships represented in GPIC.

```
ROUTER_BOARD.BOUNDING_BOX.COMPUTE(
           A1 : SET OF Coordinate) :
             [Coordinate,Coordinate];
{
  Coordinate min_coord,max_coord;
  min_coord.x = Min(X_values(A1));
  min_coord.y = Min(Y_values(A1));
  max_coord.x = Max(X_values(A1));
  max_coord.y = Max(Y_values(A1));
  RETURN [min_coord,max_coord];
}
```

Figure 7: An Example of Path-Specific Mapping Function

The examples so far in this section have assumed data is stored in the COT format and accessed by TRACKER. For true inter-operability the reverse must also be true; board data stored in TRACKER format must be accessible by external COT applications. Indeed, the results produced by TRACKER must be returnable in COT format. In this case, implementation of the reverse mapping presents no major problems as (with respect to routing data only) the information content of both representations is identical. It follows that the mapping functions are *bijective* [14].

More generally, the information content of two schemas will not be identical. Mapping becomes problematic because a mapping function itself may need to provide additional data, beyond that held by the source schema. For example, the COT format is capable of representing all VLSI design data, not just that associated with routing. To produce complete COT data from the router results, in theory, requires all non-routing data to be accessed from elsewhere during mapping. This problem was avoided in this application because it was possible to isolate the COT data concerned solely with routing, hence the mapping is confined to the relevant subset of COT.

A detailed discussion of the problem of non-bijective mappings is beyond the scope of this paper. A number of possible solutions can be identified, however. The question is how to supply the additional data when that held in the source schema is insufficient. In some cases, the appropriate action will be to prompt a user at the time the mapping function is carried out. It will then be the users responsibility to provide the additional information. For some applications, it may also be acceptable to rely on default values for when a specific value is not provided by a mapping. Alternatively, where mappings are bi-directional, it may be possible to generate the data required to complete the reverse mapping while carrying out the forward mapping. It is necessary to identify where the forward mapping results in information loss. The discarded information should be stored in a new data repository specific to the mapping functions. As an example, assume that the source schema holds the *x*, *y* and *z* dimensions for a 3D artifact and that these values are used to compute a *volume* value in the target schema. Information loss can be identified (because COMPUTE is a many-to-one operator). Hence, the forward mapping should not only generate a value for the target schema, but record the original *x*, *y* and *z* values in the mapping data repository. This information can then be recovered during the reverse mapping.

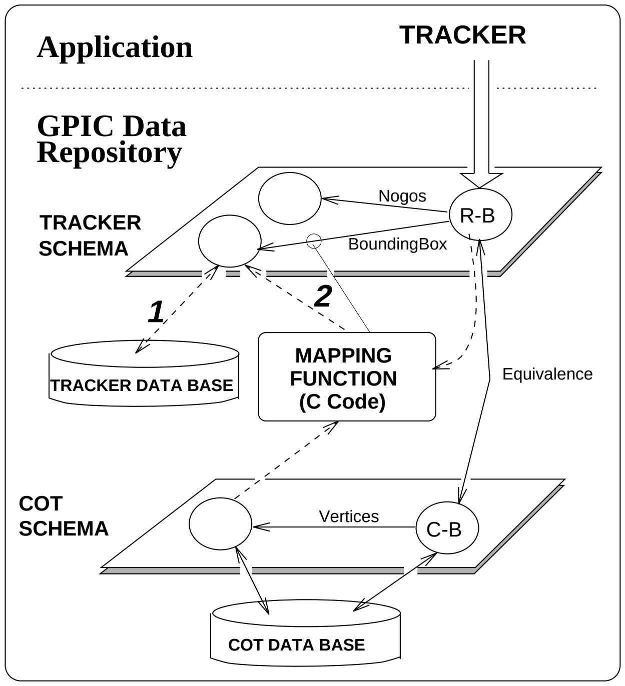

Figure 8: Indirect Data Access via a Mapping Function

Finally, the practical implementation of mapping needs to be considered. A general scheme is shown in figure 8, for access of routing data by TRACKER. The required data can be stored either within the COT data repository or within a new data repository that is local to TRACKER. When TRACKER makes a request for data, such as an attempt to navigate a particular CMDL relation, two possible access paths exists:

1. The data can be found directly within the TRACKER database, hence the access is as for a normal GPIC call.

2. The data is not found in the TRACKER database. This implies that data access must occur via the COT schema. Where equivalence relationships have been established between the schemas, the return of a data object handle can be direct. However, the more general case will be to trigger an external mapping function associated with the CMDL relation being navigated in the TRACKER schema. This function will access the appropriate data via the COT schema, carry out necessary transformations and return the relevant value for the TRACKER's schema.

The decision as to which access route to support is application dependent. For the first mode of data access, the mappings can be used to implement a *reader/writer* translation such that data in the source database (i.e. COT data in this case) can be read via GPIC and the equivalent data stored in a new target (i.e TRACKER) database. This will increase the efficiency of subsequent data access but carries the over-head of redundant data storage. The second alternative is to avoid data redundancy (and hence, avoid possible problems with data inconsistency), at the cost of requiring a large number of mapping function calls at the run-time for the application. For the routing scenario discussed in this paper, the first alternative was favoured as the nature of routing requires a very large number of data accesses at run time, while at the same time it was known that the routing data was not to be shared between multiple applications, hence possible data inconsistency is not a major problem.

4 Conclusions

In summary, this paper has described the use of a flexible and adaptable procedural interface as a means of tackling the problem of semantic heterogeneity that frequently occurs when two incompatible tools are required to communicate with each other. The procedural interface allows two different data representations to be mapped onto a more general conceptual model in a straightforward manner.

Once two schemas are converted to the same representation, mappings between them can be established. The approach advocated here is to use a set of operators to systematically capture the structure of the mapping in a declarative manner. This provides a useful guide for maintaining inter-schema mappings over time. In addition, the declarative description of a mapping can be used to guide the generation of the its underlying C-code implementation.

A demonstration of the use of mappings has been built using the GPIC procedural interface on top of the OMS database. The material presented here has illustrated a practical approach to inter-operability of CAD tools.

4.1 Related Work

The problem of overcoming semantic heterogeneity is currently a hot topic in database research. Numerous approaches have been suggested (e.g. [15, 16]), of which just a few notable examples can be covered here.

In [9], the approach advocated for integrating multiple, heterogeneous databases is to construct a global schema that is the union of all local schemas. The approach is similar to the one taken here in that the integration is achieved by first converting the data model of each specific database into a general data model. Each object in a database schema is represented by a single object in the global schema. Integration is represented by sets of equivalent objects from different databases mapping to the same object in the global schema. In [9], a comprehensive, formal definition of the sets of constraints and relationships between schemas integrated in this way is given. However, many practical problems are not fully addressed. In particular, though it is recognised that explicit mapping knowledge is required, in order to convert data between the local and global schemas, no methodology is provided for generating and applying the mappings. The work described in this paper is a first step towards tackling this general problem.

The alternative to integration via a global schema is that of federated database systems [15]. In federated database systems, each databaseremains autonomous despite the existence of inter-database cooperation. Unification only exists locally, between small, related groups of databases. Again, the central issue is the construction and maintenance of inter-schema mappings. The practical example to integration described in this paper fits well with the federated model, as it demonstrates inter-schema communication via mapping while, through integration with GPIC, allowing each of the external applications to remain independent of each other. It can be concluded that continued development of the GPIC procedural interface to incorporate mapping facilities will impact on research into heterogeneous database systems.

4.2 Future Work

The work that remains to be done in tackling the interoperability problem using GPIC is as follows:

- Automating/semi-automating the declarative specification of a mapping using the mapping operators.
- Providing assistance for the matching of objects and concepts during mapping generation, e.g. using an enhanced data dictionary.
- Providing access to mapping data/meta-data through the GPIC interface procedures.
- Providing facilities to deal with non-bijective mappings. This includes identification of which mapping operators lead to information loss (and hence may require additional information to be stored to support the reverse mapping). In addition, GPIC needs to be provided with extra facilities, such as default inheritance.
- Providing other practical examples of inter-operation, e.g integration of CFI/EDIF, or schema evolution of existing EDIF versions.
- Benchmark tests with respect to other approaches.

References

[1] International Computers Limited. *DA-X User Manual, Volume I.*

[2] D. Edwards, A. MacIntosh, Z. Moosa, and F. Hoyle. *University of Manchester PCB Software*. University of Manchester, October 1994. Version 2.4.

[3] *Design Framework II User Guide*, September 1992. Version 4.2.1.

[4] *JCF 2.0 System Overview*, 1992.

[5] T.G.R. van Leuken J. Wissenburgh, P. van der Wolf and P. Bingley. An Introduction to the NELSIS CAD Framework. Technical report, Delft University of Technology, March 1991.

[6] Digital. *PowerFrame Handbook*, 1991.

[7] D. Harrison, A. Newton, R. Spickelmier, and T. Barnes. Electronic CAD Frameworks. *Proceedings of the IEEE*, 78(2):393 – 416, 1990.

[8] N. Filer, M. Brown, and Z. Moosa. Integrating CAD Tools into a Framework Environment Using a Flexible and Adaptable Procedural Interface. In *European Design Automation Conference*, pages 200 – 205, September 1994.

[9] M P Reddy, B E Prasad, P G Reddy, and Amar Gupta. A Methodology for Integration of Heterogeneous Databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):920–933, December 1994.

[10] T.C. Young. A Generic Procedural Interface to CAD Data. Master's thesis, University of Manchester, 1991.

[11] CAD FRAMEWORK INITIATIVE, INC. *Design Representation Programming Interface Electrical Connectivity*. CFI Version 1.0.0.

[12] Z. Moosa. GPIC User Manual. University of Manchester, Draft Document, January 1994.

[13] P. Britton, M. Brown, Z. Moosa, and N. Filer. Storing and Using Semantic Knowledge in the Framework. Technical Report JCF/MAN/102-02, University of Manchester, 1993.

[14] P. Pun. *Knowledge-Based Applications = Knowledge-Base + Mappings + Applications*. PhD thesis, University of Manchester, 1993.

[15] Special Issue on Heterogeneous Databases. *acm computing surveys*, 22(3), September 1990.

[16] Special Issue on Heterogeneous Distributed Database Systems. *Computer*, 24(12), December 1991.