View-Concepts: Knowledge-Based Access to Databases *

Jon A. Pastor
Paramax Systems Corp.
(A Unisys Company)
Valley Forge Laboratories
Research and Development
PO Box 517
Paoli, PA 19301
pastor@prc.unisys.com

Donald P. McKay
Paramax Systems Corp.
(A Unisys Company)
Valley Forge Laboratories
Research and Development
PO Box 517
Paoli, PA 19301
mckay@prc.unisys.com

Timothy W. Finin
Computer Science Department
University of Maryland,
Baltimore County
5401 Wilkens Avenue
Baltimore, MD 21228
finin@cs.umbc.edu

Abstract

Semantic data models for database systems provide powerful tools to assist database administrators in designing and maintaining schemas, but provide little or no direct support for users of the database. Some research has been done on mapping user models of a domain to the underlying database using semantic schemas. Little has been done, however, on mapping conceptually meaningful data structures to a database lacking a semantic schema, or to a multi-database system that lacks a consistent semantic schema. We argue for the appropriateness of a knowledge representation language as a language for describing the database schema, user data structures, and the mapping between them; present a problem domain in which an existing relational database without a semantic schema must be accessed by a knowledge-based application; and describe our implementation of a system that provides access to a relational database from a KL-ONE-style knowledge representation language.

1 Introduction

The integration of AI and DBMS technologies promises to play a significant role in shaping the future of computing. As noted in [7], AI/DB integration is crucial not only for next-generation computing, but also for the continued development of DBMS technology and, in many cases, for the effective application of AI technology. The motivations driving the integration of these two technologies include the need for

- access to large amounts of shared data for knowledge processing,
- efficient management of knowledge as well as data, and
- intelligent processing of data.

In addition, AI/DB integration at Paramax was motivated by the desire to preserve the substantial investment in most existing, or **legacy**, databases. To that end, a key design criterion was that our integration technology support the use of existing DBMSs as independent system components.

We distinguish four approaches to the integration of AI and DBMS technologies:

- extended AI system
- extended DBMS system
- loosely coupled AI/DB interface
- enhanced AI/DB interface

Our previous work on the Intelligent Database Interface (**IDI**) [11] focused on an enhanced AI/DB interface for logic-based systems; the work described in this paper builds on the IDI by defining a **view-concept** model demonstrated using the Loom Interface Module (**LIM**).

The IDI is a cache-based interface to DBMSs, and is designed to be easily integrated into various types of AI systems. The design of the IDI also allows it to be used as an interface between DBMSs and other types of applications, such as database browsers and

*This work is supported by Rome Laboratory and the Defense Advanced Research Projects Agency under USAF contract F30602-91-C-0040. The views and conclusions are the authors' and should not be interpreted as the official opinion or conclusions of the U.S. Government, the USAF, Rome Laboratory or DARPA.

general query processors. LIM is an extension of the IDI for structured knowledge representation systems. Since the IDI is modular in design, applying LIM to a different DBMS, or to a different data model, requires only implementation of the appropriate back-end for the other DB system.

While use of relational DBMSs is burgeoning, the appropriateness of the relational model as a user data model has been questioned. For example, proponents of the **view-object** model [15] have argued that while efficient retrieval, sharability, and other considerations make the relational model an ideal choice for storage management, users should be permitted to view their application domains in terms of conceptual objects. When a semantic schema for the relational database exists, considerable support can be provided for the design of view-objects, and automatic retrieval and update can be accomplished [3][4]. In the absence of a semantic schema, however, some means must be found to

- model the semantics of the database,
- model the semantics of the application domain, and
- provide a mapping between these two models that permits retrieval from and update to the database in terms of the application model.

If the retrieved objects are to be processed in a conventional manner, any representation language that meets these three requirements will suffice. If, on the other hand, the objects are to be manipulated by a knowledge-based system, it is desirable to choose a representation language that supports

- typing and classification of domain concepts,
- support for rule-based programming, and
- support for logic-based programming.

While some relational systems (e.g., POSTGRES [13]) provide support for rule-based programming, and any Prolog [9] system with a database interface will provide support for logic-based programming, the processing in both cases is applied not to conceptual objects in the domain, but to tuples in the database. In contrast, current-generation knowledge representation systems (**KRS**s) such as Loom [10] provide logical and procedural operators, and thus permit both logic- and rule-based programming to be applied to conceptual objects. Such a KRS is, we believe, the ideal language in which to implement object-based views on external databases (**EDB**s), since it provides ample representational and inferential power for describing both the explicit and implicit differences in the semantics of multi-database systems.

The Loom Knowledge Representation Language [10] traces its lineage back to KL-ONE [6], but has incorporated and extended the separation of the **terminological** and **assertional** components, where the terminological component is used for definition of generic concepts, and the assertional component is used for the creation of and reasoning about instances of those concepts. It has also incorporated a **procedural** component, to support rule-based programming. LIM augments Loom's terminological component with DB mapping constructs, and extends Loom's assertional component with information retrieved from databases. Instances retrieved from databases can be operated upon by both Loom's assertional language and its rule language. LIM uses Loom to represent both the semantic schema for a relational DB and application domain models.

LIM is being developed in the context of the DARPA/Rome Labs Planning Initiative (**DRPI**), a multi-site project whose goal is the development and introduction of a knowledge-based system to support military logistics planning for the the United States Transportation Command (**USTRANSCOM**), which is responsible for planning movements of troops and materiel. The current planning system is a 70s-vintage batch-oriented system that uses large, heavily-encoded records; the plan for even a moderate-sized operation is enormous, with some containing hundreds of thousands of records, effectively precluding any purely memory-resident storage scheme. The system has recently been extended to use a relational database in place of flat files for some applications. The designers of the relational DB schema were constrained by a need to adhere fairly closely to the original data format for the plan records, since the planners are quite familiar with the current structure of the data. Our project is thus faced with the task of interfacing a KRS to a collection of legacy databases that lack a coherent semantic schema.

Related work within the DRPI is being performed by groups at ISI [1][2] and UCLA [8]. ISI's **SIMS** (Services and Information Management for Decision Systems) is designed to map the queries of users, who are presumed to be ignorant of the structure and content of a collection of databases, into retrievals against those databases. UCLA's **COBASE** (Cooperative Database) is a knowledge-based extension to standard relational query languages (e.g., SQL) that provides fuzzy operators supporting query relaxation and approximate answers.

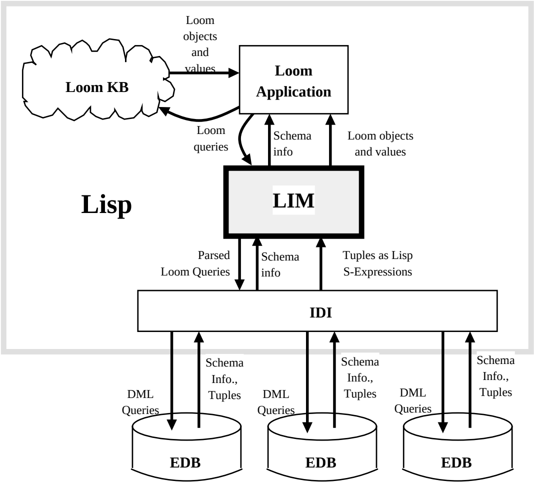

Figure 1: LIM Overview

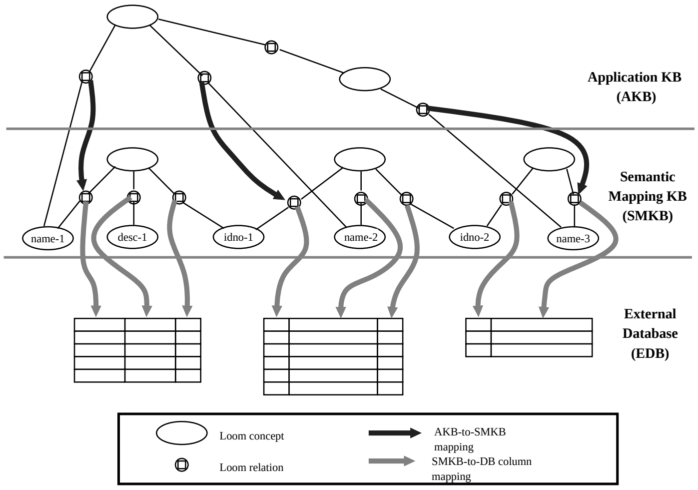

Figure 2: LIM Knowledge Base Architecture

Some work has been done toward implementing the view-object model on relational DBs for which a semantic schema exists. In particular, Barsalou and Wiederhold [3][4] describe a system based on the Structural Model [14], an extended entity-relationship model. In this system, the user selects a pivot relation that includes the intended key(s) for the object being defined. The various link-types in the structural schema are then traversed, and, using a relevancy metric, a tree of candidate relations - rooted at the pivot - is generated. The user then prunes this tree, leaving only the relations and attributes that s/he wishes to have in the defined object. Once the object is defined, it is linked into an object hierarchy. Barsalou and Wiederhold provide algorithms that assure that objects are retrievable, and, if desired, updatable. The **view-concept** model described here is intended as a knowledge-based extension of the view-object model; it is also based upon an Ingres DB interface we developed for the CYC KRS [12].

2 Architecture and Operation

LIM acts as an intermediary between a Loom application and one or more EDBs, using the services of the IDI to access the EDBs. The inter-relationships among the various components of the overall system are illustrated in Figure 1.

LIM uses the IDI to read the EDB schema, then builds a Loom representation of the schema based on this information. Subsequently, in response to a query from a Loom application that requires access to the EDB, LIM parses the query and uses the IDI to generate the appropriate DML, then processes the tuples returned to it by the IDI into the form requested by the application.

2.1 Overview

Processing within LIM is directed by a multi-layer KB architecture that is built in a mixed-initiative process. Figure 2 depicts the layers in this architecture.

The Semantic Mapping KB (**SMKB**) is an isomorphic representation of the EDB schema, with each table in the EDB represented by a Loom concept, and each column represented by a Loom relation. It is constructed by the Knowledge Base Administrator (**KBA**) from an automatically-generated schema model, primarily by defining semantic types and substituting these for the simple EDB types that appear in the original schema model. Application KBs (**AKB**s), built by application writers, refer to concepts and relations in the SMKB.

LIM, given a query involving a concept in the SMKB or AKB,

- obtains schema mapping information from the SMKB;
- translates the query into an equivalent DML query with the aid of the IDI, which submits the query and assembles the result; and
- restructures the returned tuples as necessary, generating any KB structures required to satisfy the query.

With regard to the last point, a fundamental principle of LIM is that KB structures are created only on demand: queries are satisfied without creation of KB objects whenever possible, to minimize overhead and bookkeeping. Control over object creation is entirely at the discretion of the application.

2.2 Operation

The processing performed by LIM is illustrated in Figure 3.

2.2.1 Schema Generation

When a DB is opened via the IDI, schema information is cached by the IDI in local data structures. The schema generation module reads this information, and generates one Loom concept per table and one Loom relation per column. The relations corresponding to the columns of a table are then added as roles of the corresponding concept via value restrictions.

2.2.2 Schema Augmentation

The automatically-generated schema model is a literal representation of the EDB schema. One implication of this is that the semantics of relationships among tables are not explicit, since the schema does not identify the columns in the EDB over which joins are semantically reasonable. In creating the SMKB, the KBA augments this literal schema representation by defining semantic types to explicate the semantics of joins. In particular, where two relations represent columns over which a join is semantically reasonable, the value restrictions on these relations in their respective concepts are changed to the same KB type. For example, two DB columns whose DB type is *integer*, but which both represent a particular kind of identification number, would have their value restrictions specialized to a concept representing that kind of identification number.

In addition to modification of role value restrictions, it may be desirable to represent the structural semantics of the domain more closely than is possible in the relational model. Such restructuring may be specified in the AKB by defining view-concepts, and mapping their roles to those of SMKB concepts. When such semantic restructuring takes place, it is necessary to ensure that the proposed structures are retrievable, and – if desired – updatable. Retrievability requires that all participating tables can be joined, and that sufficient information (i.e., keys) is preserved in the semantic representation to permit unambiguous access of all necessary tuples. Updatability requires that all key, index, and non-null columns in the DB tables underlying a view-concept in the AKB are included in the view-concept.

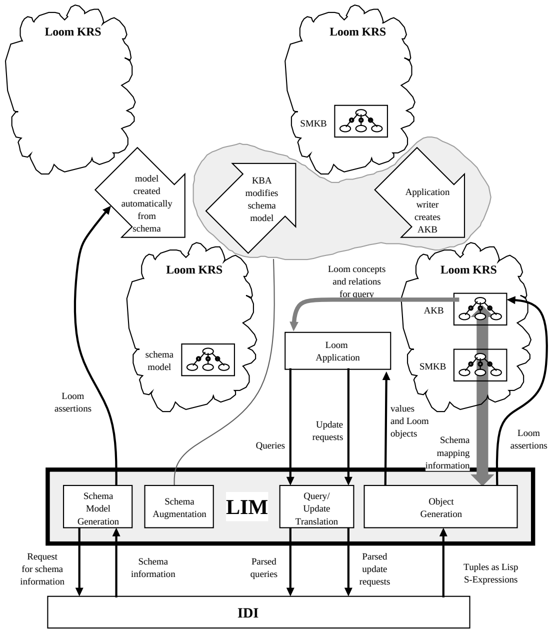

Figure 3: LIM Internal Architecture and Processing

Retrievability of view-concepts is assured via a mixed-initiative dialog, in which the system computes all semantically meaningful ways of joining all of the DB tables required for the construction of a view-concept; if there is more than one such alternative, LIM presents them for selection by the user. It is not possible for the system to compute join paths without user intervention, since any given pair of tables might be joinable in several ways, not all of which are semantically equivalent.

Updatability of view-concepts, when required, is checked automatically by the system. If any necessary information is unavailable in the view, the system enters another mixed-initiative dialog with the user to include the missing information.

2.2.3 Query Generation

Given a LIM query, the query generation module:

1. identifies variables in the query corresponding to relations that are derived from the EDB,

2. identifies variables in the query corresponding to concepts having roles derived from the EDB, and

3. constructs a DML query and submits it to the IDI for processing against the EDB.

If the query requests the return of Loom objects, rather than just values from the EDB, the DML query will select and return values in each tuple to permit generation of the appropriate Loom objects.

2.2.4 Object Generation

A LIM query consists of a list of output variables to be bound, and one or more statements, in a syntax similar to that of the Loom assertional language, that produce sets of bindings for these variables. It is easily determined from the syntax of a query whether a particular output variable corresponds to a role value or a concept. For a variable corresponding to a role value, the value retrieved from the EDB can be returned to the application, possibly with some conversion due to the differences between semantic types used in the KB and simple DB types. For a variable corresponding to a concept, however, the application will expect to have returned to it an instance of that concept; this requires that LIM be capable of creating Loom instances using values retrieved from the EDB. LIM's object generation module extracts from the returned tuples all values requested specifically for the purpose of building Loom objects, creates the objects, and returns them to the application.

2.2.5 Update

The present implementation of LIM does not support update; however, we have studied the issues, and will describe the planned implementation.

Loom instances can be constructed incrementally, by asserting the existence of the instance and then subsequently asserting facts about it. Furthermore, classification of an instance does not take place as a result of asserting its existence, or asserting facts about it, but must be explicitly requested. As noted in section 2.2.2, if an instance is to be stored into an EDB, an update cannot be performed unless sufficient information (e.g., values for joins) is available. While it would be possible, if somewhat difficult, to determine the point at which sufficient information has been asserted about an instance to permit storage in the EDB, it is not possible to determine whether this is what the user intends. We therefore separate instance creation from a request for storage in the EDB.

When a user issues a **store** request, LIM verifies that the instance contains sufficient information and does not violate any conditions imposed by the DBMS; if both of these criteria are met, the EDB is updated via the IDI. At this point, we are uncertain whether Loom instances that are successfully stored should be erased from the Loom KB, since they are now accessible from the EDB via LIM. This will be the subject of further study, and may require observation of usage patterns in an initial implementation.

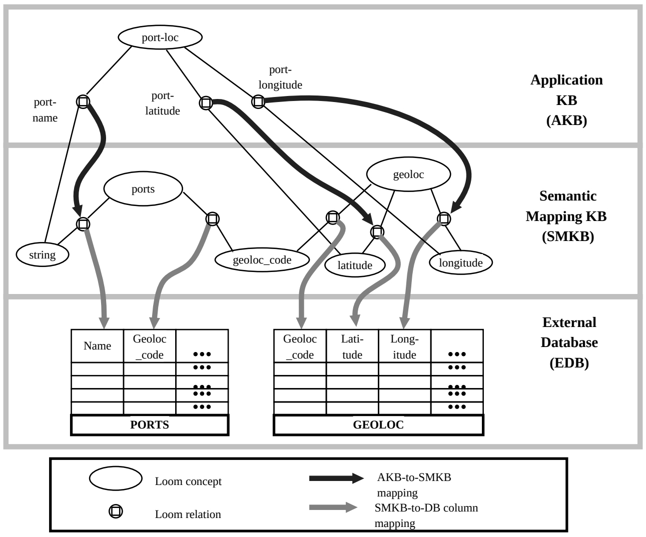

Figure 4: Example of use of KB mapping layers

3 Example

To illustrate the processes described in Section 2, we present a small, fairly simple example. Let us presume that an application requires information about the location of various ports. In the USTRANSCOM databases with which we are working, information about ports is stored in a table called PORTS, and information about geographic locations in a table called GEOLOC. The various KB layers representing the mapping from application to EDB are shown in Figure 4.

Note that from a user's perspective, the SMKB and DB pre-exist, and definition of application concepts thus appears to be a top-down process; however, in order to illustrate the process of defining the mappings, we will proceed bottom-up.

The bottom panel shows a simplified tabular representation of the schema definitions for the two tables, PORTS and GEOLOC. The middle panel shows the SMKB concepts representing the two tables. These were created by modifying the value-restrictions in the Loom definitions automatically generated by the schema generation module. For example, the initial Loom concept definition for the portion of the GEOLOC table shown is:

```
(defconcept Geoloc
    :is-primitive
        (:and semantic-db-concept
            (:the Geoloc.Geoloc_Code String)
            (:the Geoloc.Longitude Number)
            (:the Geoloc.Latitude Number)))
```

Note that role value restrictions correspond to the simple data types (e.g., *string*, *number*) that appear in relational databases. This definition is modified by the KBA to produce the SMKB definition:

```
(defconcept Geoloc
    :is-primitive
        (:and semantic-db-concept
            (:the Geoloc.Geoloc_Code Geoloc_Code)
            (:the Geoloc.Longitude Longitude)
            (:the Geoloc.Latitude Latitude)))
```

For example, the role of **Geoloc** that corresponds to the column **geoloc_code** has type *string*; this has been modified in the SMKB to **geoloc_code**. This permits LIM to infer that **Ports** and **Geoloc** can be joined over their **geoloc_code** roles.

Loom definitions for the semantic type hierarchies above the types used in **Geoloc** are:

```
(Defconcept Identifier :Is-Primitive Thing)
(Defconcept Code :Is-Primitive
    (:and String Identifier))
(Defconcept Location :Is-Primitive Code)
(Defconcept Geoloc_Code :Is-Primitive Location)

(Defconcept Measured_Qty :Is-Primitive Number)
(Defconcept Degrees :Is-Primitive Measured_Qty)
(Defconcept Latitude :Is-Primitive Degrees)
(Defconcept Longitude :Is-Primitive Degrees)
```

Finally, the top panel shows a simple application-level concept derived from information in both DB tables. The following is the Loom concept definition for the AKB concept **port-loc**, which was created manually:

```
(defconcept port-loc
    :is-primitive
        (:and view-concept
            (:the port-name string)
            (:the port-latitude latitude)
            (:the port-longitude longitude)))
```

This is mapped to the EDB by making the following declarations, which are stored as assertions in the Loom KB:

```
(def-db-mapping port-name port-loc ports.name)
(def-db-mapping port-latitude port-loc
    geoloc.latitude)
(def-db-mapping port-longitude port-loc
    geoloc.longitude)
```

Queries can be posed against either the SMKB or the AKB. (Note: the names used in the following examples have been changed; we have not yet obtained permission to publish the data in our test database.) For example, the query:

```
(db-retrieve (?name)
    (:and
        (Ports ?port)
        (Geoloc ?geoloc)
        (Ports.Geoloc_Code ?port ?geocode)
        (Geoloc.Port_Code ?geoloc ?geocode)
        (Ports.Name ?port ?name)
        (Geoloc.Country_State_Code ?geoloc "DP")
        (Ports.Clearance_Rail_Flag ?port "Y")))
```

("What are the names of ports in Dogpatch that have railroad capabilities at the port?") can be posed against the SMKB. The SQL generated by LIM and the IDI for this query is:

```
SELECT DISTINCT RV1.name
FROM   PORTS RV1, GEOLOC RV2
WHERE  RV2.geoloc_code = RV1.geoloc_code
AND    RV2.country_state_code = 'DP'
AND    RV1.clearance_rail_flag = 'Y'
```

The values returned are:

```
("Cair Paravel" "Minas Tirith"
 "Coheeries Town" "Lake Woebegon" "Oz")
```

The query:

```
(db-retrieve ?port
    (:and (port-loc ?port)
          (port-name ?port "Oz")))
```

("Return a **port-loc** object for the port whose name is 'Oz' ") can be posed against the AKB. The SQL generated by LIM and the IDI for this query is:

```
SELECT DISTINCT RV1.name,
       RV2.latitude,
       RV2.longitude
FROM   PORTS RV1, GEOLOC RV2
WHERE  RV2.geoloc_code = RV1.geoloc_code
AND    RV1.name = 'Oz'
```

The value returned by this query is an object whose Loom definition is:

```
(TELL
  (:ABOUT PORT-LOC59253
          PORT-LOC
          (PORT-LONGITUDE 98.6)
          (PORT-LATITUDE 3.14159)
          (PORT-NAME "Oz")))
```

4 Status

Our current system is implemented in Lucid Common Lisp and runs on a SUN SPARCstation 2. LIM uses both the IDI [11] and the Loom knowledge representation language [10]. The IDI uses one of several protocols to access Oracle databases on a remote server. We have developed the first prototype of the LIM system described above and a test set of approximately 50 queries. The queries have been executed from a running Loom system against a remote Oracle database; update is not yet supported. Several other participants in the DRPI, including ISI, UCLA, and BBN, are presently using the prototype implementation.

Our plans for extensions to LIM include support for:

update: LIM's query module extends Loom's retrieval capabilities. We are currently designing an update module which extends Loom's assertion language.

hierarchical concepts: Complex knowledge representation concepts have hierarchical structure that cannot be directly mapped from a relational database; LIM supports simple hierarchical models, and we are currently extending LIM's ability to retrieve and manipulate hierarchical Loom objects.

multi-DB access: Our current implementation of LIM assumes that all database entities referenced in a single Loom concept are from the same database. We plan to allow for multi-database access, ultimately by integrating our system with SIMS multi-database query planning system; our near-term solution may involve simple distributed query factoring at the semantic mapping layer.

data-rendering: It is often the case that semantic types do not have simple relationships to the simple types in the EDB; we are planning to implement a range of type-mapping, or data-rendering, strategies, possibly with support from SIMS[1][2].

In addition, we hope to integrate LIM with SIMS[1][2] and COBASE[8], and to produce jointly with their developers a system providing integration of access to multiple sources, approximate queries and answers, and fault tolerant knowledge base access to databases. Some of the initial integration results are reported elsewhere in these proceedings, c.f. [2].

5 Conclusion

We have described a view-concept model which uses a knowledge representation language, Loom, to define the semantic schema of a database. This definition has two levels, each of which is of utility to a knowledge-based application. Both are based on a verbatim model of the database; for legacy databases, this can be generated automatically from the database schema, and can be used by any knowledge-based application which would assist a knowledge base administrator in the development of the semantic mapping layer (i.e., a knowledge-based semantic schema).

The semantic mapping layer defines the relevant concepts supported by the database domain; in our current knowledge bases, the semantic mapping layer adds semantic types to the automatically-generated schema model. We envision additional information in the semantic mapping layer, including composites of database objects which form larger conceptual structures.

Finally, the view-concept model includes an application-specific layer that defines the mapping between an application domain's conceptual structures and the semantic definition of database concepts. We believe that the structured approach embodied in the view-concept model significantly elucidates the knowledge-base-to-database interface problem. Further, we expect that grounding the implementation in the IDI will support reasonable performance.

Our preliminary implementation includes algorithms for properly defining objects to determine retrievability and updatability, as well as retrievals against a database. In the coming year, we will be developing a more sophisticated application for the military transportation logistics domain using LIM. We expect feedback from this experience primarily to concern the completeness of the application knowledge base, and to give us valuable performance data.

Acknowledgements

The authors wish to acknowledge the contributions of the current Paramax Knowledge-Based Planning group, specifically Rebecca Davis, Rich Fritzson, Robin McEntire, and Barry Silk (US Government). In addition, the first version of the Intelligent Database Interface was developed by Tony O'Hare, currently at IBM - Research Triangle, and Larry Travis of the University of Wisconsin.

References

[1] Yigal Arens, "Services and Information Management for Decision Support," *AISIG-90: Proceedings of the Annual AI Systems in Government Conference*, George Washington University, Washington, DC, May, 1990.

[2] Yigal Arens, "Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems," *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, November, 1992.

[3] Thierry Barsalou and Gio Wiederhold, "Applying a Semantic Model to an Immunology Database", 1987.

[4] Thierry Barsalou, "An Object-Based Architecture for Biomedical Expert Database Systems", 1988.

[5] Thierry Barsalou, Arthur M, Keller, Niki Siambela, and Gio Wiederhold, "Updating Relational Databases through Object-Based Views", ACM, 1991.

[6] Ronald Brachman and James Schmolze, "An Overview of the KL-One Knowledge Representation System", *Cognitive Science* 9, 1985, pages 171-216.

[7] M. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, 1984.

[8] Wesley W. Chu, Andy Y. Hwang, Rei-Chi Lee, Qiming Chen, Matthew Merzbacher, and Herbert Hecht, "Fault Tolerant Distributed Database System via Data Inference", *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama, October 9-11, 1990.

[9] R. Kowalski, *Logic for Problem Solving*, Elsevier, 1979.

[10] Robert MacGregor and Robert Bates, "The Loom Knowledge Representation Language", *Proceedings of the Knowledge-Based Systems Workshop*, April 1987.

[11] Don McKay, Tim Finin, and Anthony O'Hare, "The Intelligent Database Interface", *Proceedings of the 7th National Conference on Artificial Intelligence*, 1990.

[12] G. Christian Overton, Kimberle Koile, and Jon A. Pastor, "GeneSys: A Knowledge Management System for Molecular Biology", *Computers and DNA*, Santa Fe Institute, G. Bell and T. Marr, editors, Addison-Wesley, Reading, MA, 1990.

[13] Michael Stonebraker and Larry Rowe, *The Postgres Papers*, University of California - Berkeley, 1987.

[14] Gio Wiederhold and R. ElMasri, "The Structural Model for Database Design", in *Entity-Relationship Approach to System Analysis and Design*, pages 237-257, North Holland, 1980.

[15] Gio Wiederhold, "Views, Objects, and Databases", *IEEE Computer*, Vol. 19, no. 12, December 1986, pages 37-44.