Statement of Interest
Workshop on Term Subsumption Languages for Knowledge Representation

Tim Finin, Rich Fritzon, Anthony O’Hare
Don Mckay and Robin McEntire *
Unisys Paoli Research Center
P.O. Box 517
Paoli, Pennsylvania 19301
June 1989

For the past year, our group has been working on the design and implementation of a new knowledge representation and reasoning system, the Intelligent System Server. This research has focused on three issues:

- Efficiently delivering knowledge representation and reasoning services to conventional applications.
- Extending term subsumption languages to do more reasoning.
- Efficiently integrating term subsumption languages with conventional databases.

This work has been based on a collective experience which includes eight years of practical use of a term subsumption language (KNET) [?, ?] in large scale applications [?]. The result of our current work is a prototype of a new system named the Intelligent System Server. In this note, we will motivate the research issues that we are addressing and describe the approach that we are taking. Additional details can be found in [?].

Introduction

Knowledge bases are best viewed as being akin to databases; they are global resources to be shared by users and application programs. A knowledge representation and reasoning (KR&I) system should offer its services to programs in the same way that a database does: as an independent process exchanging information via network streams or interprocess communication pipes.

*finin@prc.unisys.com (215-648-7446) or fritzson@prc.unisys.com (215-648-7606) or ohare@prc.unisys.com (215-648-7479) or McKay@prc.unisys.com (215-648-7569) or Robin@prc.unisys.com (215-648-7326)
The two traditional approaches to providing KR/R service to an application require either that the application be written in the language and environment of the AI system, or that the AI system be provided in the language and environment of the application. Often, neither of these alternatives is acceptable because the application can not be moved from its environment and because the AI systems available in that environment are inadequate.

A knowledge base server provides a third alternative for embedding AI in an application. When a knowledge representation and reasoning system is structured as an independent server process, it offers several advantages:

- **Applications can be smaller.** The application no longer needs to contain the general KR/R routines, but only the application-specific representation and inference procedures.
- **Access to the knowledge base is language independent.** There is no need to program in, or even be aware of the language in which the knowledge base is implemented. This allows both the application and the KR/R system to be implemented in the most appropriate language.
- **Persistent and sharable knowledge bases are easier to manage.** Having a single server for multiple users makes dealing with concurrency and locking easier.
- **The server can offer a uniform front-end to conventional database management systems (DBMS).** If the knowledge representation system is appropriately designed, it can also serve as a front end for database management systems that are attached to the network.

**Intelligent System Server Goals**

The primary goal for an Intelligent System Server is to provide full support for an application’s KR/R requirements, even though the Intelligent System Server is executing on a separate machine. If we meet this goal, we can expect to reap the advantages described above. However, a good Intelligent System Server will be designed to address several other goals as well. In particular,

- A good Intelligent System Server should be designed to reduce the overhead of the communications links as much as possible, and will encourage taking advantage of the parallelism inherent in a cooperating process design.

While communication links are generally quite fast, a good design will attempt to reduce traffic on it by:

- eliminating redundant transfer of information,
- using a compact encoding of information,
- allowing the user to augment the Intelligent System Server with custom reasoning routines which execute on the server's host.

Allowing the user to augment the reasoning processes of the knowledge server will make the system significantly more useful as well as allowing for efficiency improvements.

- The Intelligent System Server should be extensible.
It should allow the user to extend the reasoning procedures either by writing code for the Intelligent System Server (to be executed on the Intelligent System Server's host), or by writing application code (to be executed on the application's host) which can supplement the reasoning procedures of the Intelligent System Server.

The Intelligent System Server as proposed so far can take advantage of information that is already being shared by the existing applications in relational DBMSs. However, there is no facility available today which will allow a knowledge representation and reasoning tool to store its own knowledge representation structures on a mass storage device in a persistent and shareable form comparable to a database.

This is an essential step in the development of an effective Intelligent System Server because the performance of an AI tool which relies on data stored in an independent DBMS will never be as good as one which can store the data internally, in a directly usable form. However, if the data is internalized by the AI tool, it still must continue to be available to other programs in a persistent and shareable form. That is, the knowledge base must provide the services of a DBMS.

The forth major objective of this project is to design an extension of the Intelligent System Server which will provide this capability.

**Status**

We currently have a initial prototype of the Intelligent System Server running and are working on a a more substantial one. The current prototype (and future one) uses the Protem system [?] as the knowledge representation and reasoning engine. We have implemented an intelligent database interface (IDI) which will provide Protem with access to conventional DBMSs which support SQL [?]. Earlier work on the design of an intelligent cache management system [?] will be used in implementing an intelligent cache for the Intelligent System Server.

**The Protem Representation and Reasoning System**

Protem is a hybrid knowledge representation system, in the style of Krypton. It uses a structured semantic network as its terminological component and a Horn clause logic system for its assertional component.

The semantic network component is a highly simplified system owing more to the frame system of an object-oriented program environment than to our earlier KL-One style networks. It offers a limited and nonextendible set of role facets, but does include CLOS style “metaclasses”, providing a view of generic classes as instances of other classes.

The assertional component provides both forward and backward chaining reasoning using Horn clauses. It is built on a justification-based truth maintenance system and a contradiction resolution mechanism which provides a limited abductive reasoning facility.

Protem’s strong point is its integration of the two components: its rule-based half and its frame-based half. Neither can operate independently of the other. In particular:

- All assertions and rules are about frames. They either assign values to the attributes of a frame, or assert relations among frames.
• The class/subclass hierarchy of the frame system provides types for the variables of the rule system. This typing is fast and is directly utilized by the unification algorithm.

• Rules and assertions about typed variables are inherited via the subclass links in the frame system in a well defined way. This inheritance is supported by the truth maintenance system which treats them as default values.

Protem is implemented in Common Lisp (supported by the portable implementation of the Common Lisp Object System) and has been run on a variety of machines.

Efficient Access to Databases

The primary difficulty with creating an effective knowledge base to relational DBMS interface is performance. While some knowledge bases have been linked to DBMSs in one way or another [?], the performance of the resulting combination is always very poor. There are several underlying causes of this problem that need to be dealt with.

First there is the mismatch between the behavior of a typical expert system application and the expectations of a relational DBMS [?]. Databases expect requests to be made over relations; they frequently return lists or tables of results as answers. Expert systems more frequently make requests for a single entry at a time; however, they make many such requests. That is, instead of making a single query for a large number of table entries they make a large number of queries for a single table entry each. As a result the DBMS is unable to optimize queries effectively and instead must pay the “per-query” overhead of parsing and optimizing many times over.

Additionally there is a mismatch between the storage strategies of a relational DBMS and the storage patterns of a semantic network representation system. Databases group the entries in a relational table together on mass storage while semantic networks typically cluster all of the different relations about a single object together. The pattern of queries generated by the semantic network acts to defeat the internal storage and caching strategies of the DBMS.

The Intelligent Cache

Our approach to dealing with these difficulties is based on constructing an intelligent cache between the Intelligent System Server and the DBMS. The cache performs the usual tasks of storing the results of queries in anticipation of reuse and prefetching answers in anticipation of queries. It is described as an intelligent cache because it performs prefetches and cache memory management based on advice given to it by the Intelligent System Server which describe the patterns of database access it anticipates following in the near future. Ongoing research at Unisys [?] has developed techniques for generating several types of cache advice including:

• Simply generating a list of base relations which will be used in the solving of the current problem. While crude, this does provide the cache with significant and useful knowledge.
• Providing a database view which describes a join of some of the base relations used in the current problem. This allows a precalculation by the database/cache of the relevant relations.

• Providing a prediction of relation accessing order, repetition and binding patterns, that is, information about the order in which the inference engine will emit different database queries during the processing of a single AI query.

We expect that a fully implemented intelligent cache could improve the performance of a knowledge base/database interface by at least ten fold.

The Intelligent Database Interface

The Intelligent Database Interface (IDI) is designed to provide efficient access to one or more Database Management Systems (DBMS). The Intelligent Database Interface Language (IDIL) is the query language of the IDI. Essentially, an IDIL query is a single, function-free Horn clause where the head of the clause represents the target list (i.e., the form of the result relation) and the body is a conjunction of literals which denote database relations or operations on the relations and/or their attributes (e.g., aggregation, arithmetic operations, and negation).

The IDI also manages a cache of query results and is capable of performing DBMS operations on the cache contents. Thus, given an IDIL query, the IDI has three general courses of action which it may take to produce the results: (a) the entire IDIL query can be translated into SQL and sent to the remote DBMS for execution; (b) the entire IDIL query can be executed locally by the IDI; and (c) the IDIL query can be decomposed so that part of it is executed on the remote DBMS and part of it is executed locally by the IDI. The decision of which action to take depends on a number of factors including the current contents of the cache, the estimated execution times for each alternative, and any advice which the ISS has provided the IDI with concerning future queries.

As with any type of cache-based system, one of the more difficult design issues for the IDI involves the problem of cache validation. That is, determining when to invalidate cache entries because of updates to the relevant data in the DBMS. For example, one approach is to locks the relevant data in the DBMS. While this approach ensures valid cache contents it is probably too extreme in that access to the relevant data by any other users or application is prohibited until the locks are released by the IDI.

In addition to providing efficient access to remote DBMSs, the IDI offers several other distinct advantages. It can be used to interface with a wide variety of different DBMSs with little or no modification since SQL is used to communicate with the remote DBMS. Also, several connections to the same or different DBMSs can exist simultaneously and can be kept active across any number of queries because connections to remote DBMSs are abstract objects that are managed as resources by the IDI. Finally, accessing schema information is handled automatically by the IDI, i.e., the application is not required to maintain up-to-date schema information for the IDI. This significantly reduces the potential for errors introduced by stale schema information or by hand entered data.
Additional References

Related work that we have done includes work on various aspects of classification ([?], [?], [?], [?], [?]), implementations of frame-based representation languages ([?], [?], [?], [?]), work on reasoning techniques ([?], [?], [?], [?], [?], [?], [?]), and work on the integration of AI and Database technology ([?]).

References


