The Intelligent System Server
Delivering AI to Complex Systems

**Tim Finin, Rich Fritzson, Robin McEntire,
Don McKay and Anthony O'Hare**

**Unisys Paoli Research Center
Paoli, Pennsylvania**

Abstract

This paper describes work on the design of a *Intelligent System Server* - a distributed architecture for delivering knowledge representation and reasoning services to applications. This work is based on two simple ideas. The first is that knowledge bases are like databases and their services should be provided in a similar manner – in a client-server relationship. The second idea is that a convenient and efficient interface between the AI system and conventional database management systems is a necessity. The need arises out of the fact that many of the applications that need AI services use conventional database management systems to both store much of the information used by the application and to communicate between the various components making up the application. This paper motivates the ideas underlying the Intelligent System Server, discusses some of the design issues, and briefly describes our current approach.

1 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&R*) 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. Figure 1 shows a knowledge base server attached to a local area network.

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 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.

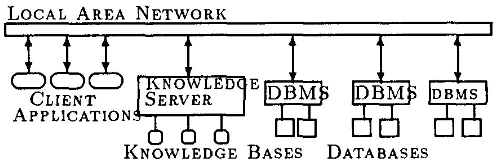

Figure 1: **From the perspective of the physical design of a server architecture, the client applications, the knowledge server and the database servers are all independent processes on a local area network.**

- **Persistent and sharable knowledge bases are easier to manage.** Having a single server for multiple users makes dealing with concurrency and locking easier.
- **A uniform front-end to conventional database management systems can be provided.** 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. (See Figure 2.)

We have designed and are implementing a system based on this knowledge server model which we call the *Intelligent System Server* or (ISS). The ISS has at its core a very expressive and powerful knowledge representation and reasoning system called PROTEM which is a rule-based knowledge representation system extended by a closely linked frame-based system. The second major component of the ISS is the *Intelligent Database Interface* which provides the system with an efficient interface to conventional relational databases.

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.

1984/89/0000/0306$01.00 © 1989 IEEE

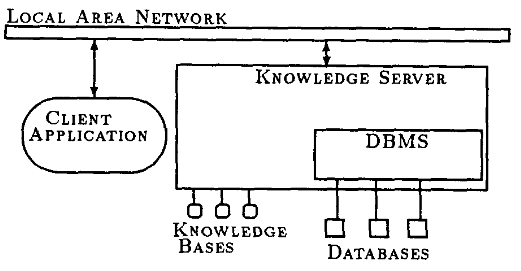

Figure 2: **From a client-oriented perspective, the knowledge server provides it with a single source for access to both knowledge bases and databases.**

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 Servers 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.

Intelligent System Server Size

In order to produce a flexible and useful Intelligent System Server there are several goals which should be kept in mind. The first is that we should be trying to provide a *big* system which supports a rich variety of knowledge representation structures and inferential procedures. The second goal is to provide a system which can also be *small*, that is, one which can be configured so that only the modules which are needed by an application are actually loaded, making delivery on modest machines possible. There are two senses to the notion of size in knowledge-based systems — the size of the knowledge bases that can be effectively managed and the size of basic *KR&R* system itself. In fact, it is reasonable to consider these to be independent dimensions along which a particular KB system (or a requirement for one) can vary.

Figure 3 shows the two-dimensional space defined by these two notions of size and places some sample points on them. The Kandor system [20,19] is an example of a KR system that was

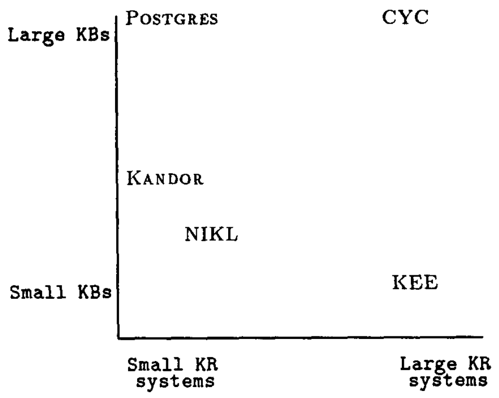

Figure 3: **Two notions of size exist for knowledge-base management systems – the size of the knowledge bases they are designed to handle and the size and complexity of the system itself**

explicitly designed to offer a small, well defined set of representation capabilities with a simple, functional interface. The Kandor system did not, however, address any of the issues which arise in managing very large knowledge bases. Postgress [23] is an example of a system which attempts to offer efficient access to very large knowledge bases by extending and enriching the traditional DBMS technologies to include some support for knowledge-based operations such as inference rules. These extensions result in an extremely simple and rudimentary knowledge representation system, however.

The KEE system [10] is a good example of a very large, rich *KR&R* system which has no special considerations for supporting very large knowledge bases. The very fact that the representation and reasoning system is large and complex makes systems like KEE poor performers on very large knowledge bases. The general correlation between the simplicity of the knowledge representation system and the size of the knowledge-base that it can support is reflected by our placement of the NIKL [9] system in this space. NIKL lies somewhere between Kandor and KEE with respect to the complexity of the representational constructs and inferential services it offers. It also is positioned between Kandor and KEE with respect to the size of the knowledge bases that it can practically support. The last sample point in the figure is CYC [12] which is somewhat unique in its goals. CYC employs a very complex and large knowledge representation system but is also explicitly targeted toward supporting extremely large knowledge bases. Its success at combining both aspects of bigness has not yet been tested.

Our goal is to provide a KBS system that can be both *large* and *small* in both of these dimensions. We would like to provide a system that can be *large* in that it supports a full range of standard knowledge representation structures and procedures which can operate efficiently on knowledge bases of significant size. Both frame-based and rule-based representations, with forward and backward chaining inference mechanisms, and some form of truth maintenance, should all be available to application programs. In addition there should be modules that supply other functions on knowledge bases, such as an editor, a knowledge base browser, classifier, consistency verifier, etc. The system should also be large in the sense of being able to manage large knowledge bases, including knowledge bases which are stored, in part, on external databases. Running on a fast "server system" host, it should be capable of supporting the development of large AI projects.

The second goal is to provide a system which can also be *small*, that is, one which can be delivered on modest machines. It should be possible to configure the system so that only the modules which are needed by an application are actually loaded. It should also be possible to execute the knowledge base server on the same host as the application program. In such a configuration, the penalty for the server architecture should be minimized. This is not an unreasonable expectation. The X Window System is designed in just this way. It is designed to work across a network, but it is most frequently executed on the same host as the application code which uses it.

Feasibility

There are several objections that can be raised against the Intelligent System Server architecture.

- It will be too slow. Transferring the data over the network will reduce performance to unacceptable levels.

This is not likely to be true. Since the reasoning component of the system is on the same side as the knowledge, the amount of information transferred between the application and the server should not be that great. In fact, less information should pass between the Intelligent System Server and the application than would usually pass between a conventional application and a DBMS. These conventional applications do not find the network connection to be a bottleneck.

For applications which do need to transfer large amounts of data between a client application and the Intelligent System Server the data transfer rates of current technology networks is approaching the transfer rates of large hard disks. Even information intensive applications such as the X Window System can operate acceptably over a network. So the speed of the network link is not likely to be a bottleneck for the application.

In fact, a well designed application could actually run faster with an Intelligent System Server because of the opportunity for parallelism that the architecture introduces.

- Applications are too tightly coupled with the knowledge representation and reasoning components to be divided by a low bandwidth connection.

This objection is more serious. However, the Intelligent System Server design doesn't try to separate general purpose reasoning from knowledge representation; that would be too difficult and would be likely to fail. On the other hand, most real applications utilize a mixture of general purpose reasoning and special purpose reasoning. The latter can be simple or complex; it may only be called once or it might have to be available as a subroutine to be called by the general purpose reasoner, or it might have to be available as a coroutine, maintaining an internal state and swapping control back and forth between the Intelligent System Server and itself. There are several ways to deal with this requirement, some of which we discuss in the next section.

2 Design Issues

A functional interface to a knowledge representation system has to provide access to the services provided by the system, such as its frame representation or its backward chaining reasoning mechanism. But when the interface is going to be provided via an interprocess stream to an independent process, rather than via procedure calls to code within the same process, new issues arise. Among these are:

- Can the server process handle more than one user simultaneously?
- Can the server process handle more than one knowledge base?
- What communication protocol will be used between client and server?
- How will data be passed from the server to the client's programming language?
- How can the client supplement the server's inference processes? Will it be possible to use the client's programming language to do this?

2.1 Multiple Users & Knowledge Bases

While it is important that multiple users be able to make use of the Intelligent System Server whether it can support multiple users simultaneously is often simply a matter of the operating system it runs under, e.g. does it support shared code? However, whether a single instance of the Intelligent System Server code can share multiple users, or whether multiple instances are needed, isn't really the important point. The important questions are:

- Can the Intelligent System Server support more than one knowledge base? That is, are the procedures and the knowledge base itself distinct?
- Can two users access the same knowledge base simultaneously? If so, then how are updating conflicts handled?
- Can a user combine two knowledge bases? That is, can a user have access to two or more knowledge bases simultaneously?

The last item is probably the most difficult to answer. A collection of *modular* knowledge bases which can be combined with one another would be a useful resource. But a procedure for combining two arbitrary knowledge bases may be impossible to devise.

2.2 Communication Protocol

The proposed server/client communication channel is an interprocess communication channel (IPC). This will typically be implemented via a TCP/IP stream, but might also be a UNIX domain IPC if the server and client are executing on the same host.

When a knowledge representation system and application share the same address space, a reference to a unit in the knowledge base is usually just a pointer within the shared address space. When there is no shared address space, as in the proposed architecture, an alternative must be developed.

A simple solution is to rely on symbolic identifiers. While this is easy to design and implement, it is not particularly efficient since the server has to maintain and use a symbol-to-address mapping for each request from the client. Also, symbolic identifiers are typically larger than numeric identifiers (i.e.. that is, they are composed of more bytes).

Another solution is to continue to pass pointers to the data structure, using the address space of the Intelligent System Server and arrange for a client to locate a particular unit using a naming or indexing scheme.

2.3 Extensibility

This is the most difficult design issue that has to be addressed when constructing an Intelligent System Server. While a powerful general purpose reasoner is an important part of an expert system, it is nearly always not enough. Real applications always have a need for special purpose additions to the reasoning routines. In a frame-based representation system, these typically take the form of *attached procedures* written directly in the host language and executed in specified circumstances (e.g. upon the addition or removal of a slot value). A similar situation arises when one wants to directly link the *KR&R* state to the application state, such as reflecting a slot's value in the appearance of a graphic icon.

When the knowledge representation and reasoning components are available as subroutines, it is easy to modify them, or to take advantage of designer provided hooks to add in special purpose procedures. However, when the reasoning routines are located remotely, and may be written in an unknown or inappropriate language, it is more difficult. The questions to be answered in a design are:

- Can the application augment the reasoning process with procedures written in the language of the server?
- Can the application augment the reasoning process with procedures written in the language of the application?

If a user is willing to write special purpose reasoning routines in the same programming language that the server is written in, then there are several traditional techniques for doing this. Most knowledge representation and reasoning tools provide escapes to the implementation programming language specifically to support this need. However, if we assume that the extensions to the reasoning procedures will not be written in the server's programming language, and in fact, can not be executed on the same host that the server is executing on, then other techniques must be devised.

One possible solution is to store the code for the special purpose reasoning routines in the knowledge base and have them returned to the application program at appropriate times to be executed. For some languages, such as Lisp, this may be a reasonable approach, though it is likely to be inefficient. For other languages, it may be difficult or impossible.

A more efficient solution is to have the application implement a *remote procedure call* (RPC) service for the server. The rules of the knowledge representation server can provide the same type of escape to this remote procedure call mechanism that it usually provides for escaping to its native language.

Implementing an RPC service is not necessarily a difficult task. In its simplest form it is just a matter of receiving encoded requests from the Intelligent System Server and dispatching them to the appropriate routines for processing. If the application language has the necessary multiprocessing facilities (as do many Lisp systems, such as Franz Common Lisp), an RPC server could be implemented as a separate process using a separate communication channel. Alternatively, it could use the same process and channel as the application program, along with a communication protocol which is prepared to handle Intelligent System Server queries after each command to the Intelligent System Server. This latter design provides an adequate degree of flexibility, but it may be too slow for intensive use, and it puts a burden on the designer of the application language interface to support this RPC service.

2.4 Programming Language Interface

Techniques for accessing the knowledge base server from several languages need to be investigated. For conventional languages, a library of procedure calls will probably suffice. For AI languages like Lisp and Prolog, such libraries will be a good starting point, but when tighter integration with the language is desirable, alternative strategies for accessing the knowledge base can be investigated.

For example, the knowledge base might be accessed as an extension of the Prolog database, or, in Common Lisp, it might return and support a stream datatype to the calling program.

2.5 Interfaces to Databases

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 sharable 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 sharable form. That is, the knowledge base must provide the services of a DBMS.

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

Our approach to this problem will be based on three technologies: current techniques for constructing object-oriented DBMSs [11], new techniques being developed for the introduction of limited forms of reasoning into relational DBMSs [23,3], and the set of advice-based optimization techniques developed while constructing our intelligent database interface.

Object-oriented DBMSs are designed to handle the storage of large quantities of "linked" data. Because their organization is frequently like that of many object-oriented programming languages they tend to have *inheritance hierarchies* like many semantic networks. However, because they are modelled after programming languages, and not knowledge representation systems, they are not directly usable as persistent forms of knowledge representation structures. But, the similarity of their structure means that many of the techniques used to implement object-oriented DBMSs can be applied to the implementation of persistent knowledge bases. For example, the storage management strategies of an object-oriented DBMS are like those of a semantic network and not like those of a relational database.

Another source of implementation techniques comes from the current effort to extend relational DBMSs to include a limited form of reasoning. This work is typically concerned with introducing restricted forms of forward chaining into the traditional DBMS realm. While the work may not be directly applicable, some of the techniques developed to maintain a high level of performance appear to be useful.

We fully expect that the optimization techniques we develop in constructing the intelligent database interface will be fully applicable to the task of developing a high performance persistent and sharable knowledge base server.

3 Related Work

Although there has been considerable work that is related to the Intelligent System Server idea, none is quite identical with it.

There is a body of work on designing formal interfaces to DBMSs which presupposes a formal database model (the relational model). No such common model exists for knowledge representation. There is work in progress to extend formal database models to incorporate more of what we would consider to be the standard set of *KR&R* services [3].

The MCC CYC system [12] is configured with a central knowledge base which many independent CYC processes can access. However, the CYC model assumes that the server and all of the client processes are in Lisp and that each client process loads a copy of the entire CYC knowledge base and the entire CYCL representation and reasoning engine into its image. Thus, the server is used only to provide a central agent to manage the persistence and sharability of the knowledge base and not to actually provide *KR&R* services.

One commercial knowledge representation product, KEE, is available in a network based model [7]. The *PC Host* product offers an expert system shell (KEE) running on a central server machine, supporting users who are running applications on PC class machines attached to the server via a network. However, the division of labor between the KEE server machine and the client machine is significantly different than what we propose. With PC-Host, *all* of the code for the application, both the general purpose and the application specific code, is executed on the server while the application machine simply takes care of the display of output and the low-level handling of input. PC-Host is more like a single process AI application relying on a networked windowing system such as the X Window System, or NeWS, for its low-level user interface.

4 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 [6] 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 [14]. Earlier work on the design of an intelligent cache management system [18,16] will be used in implementing an intelligent cache for the Intelligent System Server.

4.1 Protem Representation & Reasoning System

Protem is a rule-based knowledge representation system extended by a tightly linked frame-based system and a truth maintenance system. Protem's rule-based component offers both forward and backward chaining based on Horn clauses. The frame-based system provides a semantic network which is simple but sophisticated. While it offers the standard *class* and *instance* frames, with *superclass* relations among the frames, it also provides a *metaclass* level for describing the class level. That is, it views classes as instances of other classes. The truth maintenance component (TMS) is a simple, but efficiently implemented extension of a justification based truth maintenance[5] which includes a limited abductive reasoning component for resolving contradictions.

Protem's strong point is its integration of the two representation components: the rule-based half and the frame-based half. Neither can operate independently of the other. Like KL-Two [24] and Krypton [2], Protem uses the frame component as a *terminological component* to provide a lexicon, a grammar, and even a semantic interpretation for the statements of the rule based, or *assertional component*. In particular:

- 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.
- All predicates in the rule-based system are either frames or slots of frames. All arguments to predicates are either instances of frames or variables typed by class frames.
- 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.

- The attributes of the slots of frames control the reasoning engine. They determine, for example, whether multiple values asserted to a single slot override one another or accumulate in some fashion.

The rule-based component is essentially Prolog with a Lisp style syntax, e.g.

```
((adult ?x:person) <--
        (age ?x ?y)
        (>= ?x 21))
```

The "?x:person" notation denotes a variable typed by the class "person". Negation as failure, the Prolog cut operation and the use of arbitrary lisp forms as antecedents are supported.

```
(has-child ?x) <--
        (mother ?x ?y)
        UNLESS (adult ?y)

(not-member ?x ?c) <--
        (member ?x ?c)  !
        (fail)
(not-member ?x ?c)

(adult ?x) <--
        (age ?x ?y)
        (>= ?y 21)
```

The forward chaining rules are straightforward extensions of Horn clauses; they permit multiple consequents as well as multiple-antecedents.

```
(mother ?x ?y) -->
        (parent ?x ?y)
        (female ?x)
```

The TMS provides a simple explanation facility and a data-directed backtracking component which can function in either a user-guided or domain knowledge guided mode. The domain knowledge is encoded as a set of meta-rules. The rules describe:

- Which of the assumptions underlying a contradictory state are good candidates for further investigations.
- Ways of ordering the systems investigation into these underlying assumptions.
- Ways of augmenting the set of assumptions built into the model.

The last item allows you to write a simple model which doesn't include explicit representation of all exceptional cases. That is, it isn't necessary to write a rule such as "**If** *You have a boat, and the boat isn't damaged, and if you have a paddle, and if the river isn't too rough and if the weather is acceptable, ...* **Then** *You can paddle the boat across the stream.*" because you can write metarules which will add the exception clauses *only when it is necessary to try to disprove a deduction based on the rule.*

Because the metarules are encoded as knowledge base assertions, they can be embedded in other rules which reorder backtracking strategies dynamically, i.e. the strategies used to try to resolve contradictions will change depending on the current state of the knowledge base.

Protem is implemented in Common Lisp (supported by the portable implementation of the Common Lisp Object System [1]) and has been run on a variety of machines. The current implementation of Protem is derived from Proteus [21].

4.2 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 [8], 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 [17]. 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.

4.2.1 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 [18,15] 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 an 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.

4.2.2 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 lock 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. Figure 4 shows the organization of some of the components of the IDI.

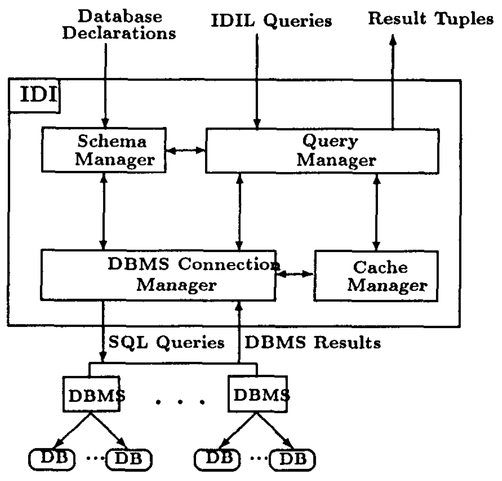

Figure 4: **The Intelligent Database Interface is designed to provide efficient access to one or more Database Management Systems.**

4.3 Application to Network Fault Diagnosis

The past decade has seen an enormous increase in the number and complexity of computer networks. These range from global data networks such as the ARPANET and MILNET to isolated local campus LANS serving anywhere from a few nodes to tens of thousands of nodes. Managing these networks is a difficult job and a primary management task is the isolation of faults and maintenance or restoration of service to users [13].

The task of diagnosing a network fault can, in part be automated and implemented as an "expert system". Effective versions of such systems would be invaluable to managers of large computer networks. However, the suite of programs used to manage networks usually run in environments which do not support AI programs and are frequently not integrated by an overall "network management supervisor program". They often use one or more databases to store information such as network configuration and trouble reports. It is difficult for an AI program to be effectively integrated into such an environment and consequently it is an ideal domain for testing the the effectiveness of the Intelligent System Server/Database Interface combination.

Network fault diagnosis is a difficult domain for an expert system, in part because large computer networks are new and undergo regular changes. There are no experts with "years of experience" debugging a ten thousand node campus LAN. There is an insufficient body of expertise on which to base the development of a typical rule-based network expert system. The problem requires the use of a model-based approach to fault diagnosis. [22,4]

Protem, the core knowledge representation and reasoning system for the Intelligent System Server, is well suited for building a model-based fault diagnosis expert.

- Its hybrid representation language (the frame-based/rule-based system describe above) can be used to construct a logical model of a process where deductions based on the model actually simulate the operation of the process.
- The truth maintenance links (or "proof tree") which remain after the deductions have been completed constitute a complete internal modelling of a particular occurrence of a process.
- The contradiction resolution component allows the system designer to provide domain dependent advice to the system on how to manipulate the "proof tree" in order to account for discrepancies between the simulated model and the observable facts. That is, it can follow *heuristic rules* which describe appropriate ways to debug the model.

We are currently testing this approach to fault diagnosis by developing a model of electronic mail delivery and a set of heuristics for debugging or explaining mail delivery problems.

5 Conclusion

This paper has described work on the design and implementation of an *Intelligent System Server* - a distributed architecture for delivering knowledge representation and reasoning services to applications. This work is based on two simple ideas. The first is that knowledge bases are like databases and their services should be provided in a similar manner – in a client-server relationship. The second idea is that a convenient and efficient interface between the AI system and conventional database management systems is a necessity. The need arises out of the fact that many of the applications that need AI services use conventional database management systems to both store much of the needed information and to communicate between the various components making up the application. In this paper we have tried to motivate the ideas underlying the Intelligent System Server, discuss some of the design issues, and briefly describe our current approach.

References

[1] D. Bobrow, L. DeMichiel, R. Gabriel, S. Keene, G. Kiczales, and D. Moon. Common lisp object system specification. March 1988. submitted to ANSI Lisp standards committee X3J13.

[2] R. J. Brachman, V. G. Pigman, and H. J. Levesque. An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. In 9th *International Conference on Artificial Intelligence*, pages 532–539, William Kaufmann, Inc., Los Altos, California, 1985.

[3] Michael Brodie. Future intelligent information systems: AI and database technologies working together. In *Readings in Artificial Intelligence and Databases*, Morgan kaufman, San Mateo, 1988.

[4] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97 - 130, 1987.

[5] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[6] Rich Fritzson and Tim Finin. *Protem - An Integrated Expert SystemsTool.* Technical Report LBS Technical Memo Number 84, Unisys Paoli Research Center, May 1988.

[7] Intellicorp. *KEE Core Reference Manual.* Intellicorp, 1986.

[8] Intellicorp. *KEEConnection: A Bridge Between Databases and Knowledge Bases.* Technical Report An Intellicorp Technical Article, Intellicorp, 1987.

[9] Thomas S. Kaczmarek, Raymond Bates, and Gabriel Robins. Recent developments in NIKL. In *Proceedings of the* 5th *National Conference on Artificial Intelligence*, pages 978–985, AAAI, August 1986.

[10] Tom Kehler and D. G. Clemenson. An application development system for expert systems. *Systems and Software*, January 1984.

[11] Won Kim, Nat Ballou, Jay Banerjee, Hong-Tai Chou, and Jorge Garza. Integrating and object-oriented programming system with a database system. In Norman Meyrowitz, editor, *Object-Oriented Programming Systems, Languages and Applications Conference Proceedings*, Microelectronics and Computer Technology Corporation, Association for Computing Machinery, 1988.

[12] D. Lenat, M. Prakash, and M. Shepherd. CYC: using common sense knowledge to overcome brittleness and knowledge acquistion bottlenecks. *AI Magazine*, 6:65–84, 1986.

[13] Donald McKay and Korrinn Fu. *FaME: Fault Management Expert for BELLMASTER I.V Network.* technical memo 93, Logic Based Systems, Paoli Research Center, Unisys, 1988.

[14] Anthony B. O'Hare. *The Intelligent Data Interface Language.* Technical Report LBS-8908, Unisys Paoli Research Center, June 1989.

[15] Anthony B. O'Hare. *The Intelligent Database Interface: Design and Rationale.* Technical Report LBS-8909, Unisys Paoli Research Center, June 1989.

[16] Anthony B. O'Hare and Amit Sheth. *The architecture of BrAID: A system for efficient AI/DB Integration.* Technical Report LBS-8907, Unisys Paoli Research Center, June 1989.

[17] Anthony B. O'Hare and Amit P. Sheth. The interpreted-compiled range of AI/DB systems. *SIGMOD Record*, 18(1), March 1989.

[18] Anthony B. O'Hare and Larry E. Travis. *The KMS Inference Engine: Rationale and Design Objectives.* Technical Report TM-8484/003/00, Unisys - West Coast Research Center, 1989.

[19] Peter F. Patel-Schneider. Small can be beautiful in knowledge representation. In *IEEE Workshop on Prnciples of Knowledge-Based Systems*, pages 11–16, 1984.

[20] Peter F. Patel-Schneider, Ronald J. Brachman, and Hector J. Levesque. ARGON: knowledge representation meets information retrieval. In *Proceedings of the First Conference on Artificial Intelligence Applications.*, pages 280–286, 1984.

[21] Charles J. Petrie, David M. Russinoff, and Donald D. Steiner. *Proteus 2: System Description.* Technical Report AI-136-87-Q, MCC, 1988.

[22] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 - 96, 1987.

[23] Michael Stonebraker and Larry Rowe. The design of POSTGRESS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 340–355, Washington, D.C., May 1986.

[24] Mark Vilain. *KL-TWO, A Hybrid Knowledge Representation System.* Technical Report 5694, Bolt, Beranek and Newman, 1984.