Parka-DB: Integrating knowledge- and data-based technologies *

Kilian Stoffel, Merwyn Taylor, and James Hendler

Department of Computer Science
University of Maryland
Colloge Park, Maryland 20742
stoffel,mtaylor,hendler@cs.umd.edu

Keywords: Implemented KR&R Systems: Reports, Evaluations, Updates, and (Parallel and Distributed Implementations)

Abstract

Real world applications are demanding that KR systems provide support for knowledge bases containing millions of assertions. We present Parka-DB, a high-performance reimplementation of the Parka KR language which uses a standard relational DBMS. The integration of a DBMS and the Parka KR language allows us to efficiently support complex queries on extremely large KBs using a single processor, as opposed to our earlier massively parallel system. In addition, the system can make good use of secondary memory, with the whole system needing less than 16MB of RAM to hold a KB of over 2,000,000 assertions. We demonstrate empirically that this reduction in primary storage requires only about 10% overhead in time, and decreases the load time of very large KBs by more than two orders of magnitude.

*This research was supported in part by grants from NSF(IRI-9306580), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039, DABT-63-95-C-0037), the ARPA I3 Initiative (N00014-94-10907). Dr. Hendler is also affiliated with the UM Institute for Systems Research (NSF Grant NSF EEC 94-02384).

1 Introduction

In the past, research in the field of (very) large knowledge bases has primarily concentrated on defining KR languages and analyzing their complexity. This research has resulted in a wide range of well-studied languages, with much known about their theoretical performance. Less work, however, has been spent examining details of efficiently implementing such languages, or on the scaling properties of these KR systems using large, real-world, applications. Up until recently, this negligence has largely been justified with the excuse that very large KBs don't yet exist, and that the community could postpone dealing with these issues. However, this excuse is no longer valid. Current research has resulted in the development of extremely large KBs including not only the "common sense" KB of Lenat's CYC [14], but also (much larger) KBs including machine-readable dictionaries [13], large ontologies [7], and very large case-bases for AI planning systems [12]. Despite this, most of the current KR systems are not able to accommodate these KBs, which may contain millions of assertions about many thousands of objects.

In this paper, we describe a new implementation of the Parka knowledge representation system which is based on a relational database system. The new implementation, Parka-DB, is therefore a KR system tightly coupled with a DBMS system. This new architecture allows us not only access to bigger KBs but at the same time improves the performance of Parka's inference engine in terms of memory utilization. In this way it allows the querying of KBs of virtually arbitrary size, while still remaining extremely efficient.

In this paper, we first overview the UM Parka KR project and present some of the applications for which we are developing very large KBs. We then describe the architecture of the hybrid knowledge and database implementation (Section 3). A series of empirical test results and performance analysis is then presented in section 4. Section 5 lists some important related projects, and we conclude with a summary of the contribution of this work and future research directions.

2 Parka Overview

The Parka system is a frame-based AI language (sometimes called a "property/class" system in today's literature) which was designed to be supported efficiently using high-performance computing techniques. The goal of the project is, and has always been, to develop a fairly traditional AI language/tool that can scale to the extremely large size applications mandated by the needs of today's information technology revolution.

More specifically, Parka allows the user to define a frame-based knowledge base with class, subclass, and property links used to encode the ontology. Property values can themselves be frames, or alternatively can be string, numeric values, or specialized data structures (used primarily in the implementation). The language allows exceptions, in the form of multiple-inheritance, and provides extremely efficient (and efficiently parallelizable) algorithms for performing inheritance using a true inferential-distance-ordering calculation [10].[1] Parka has also been shown to effectively compute recognition, and also to handle extremely complex "structure matching" queries – a class of conjunctive queries relating a set of variables and constraints and unifying these against the larger KB. While it is difficult to exactly compare KR languages, a very loose categorization would put Parka as more expressive than Classic [4], due to the presence of exception handling, although slightly less expressive than Loom [15] due to the lack of extensive numerical capabilities. A full description of the language, and more details on past results can be found in http://www.cs.umd.edu/projects/plus/Parka.

One of the key features of Parka is that it has been shown to efficiently handle its inferencing on KBs containing millions of assertions. Early work on the system gained most of its efficiency through massive parallelism [6], however in recent years we've made increasing use of database management techniques to remove the need for parallelism (although still allowing for efficient parallelization). The version we describe in this paper uses DBMS technologies to support inferencing and data management. In particular, we will describe Parka-DB which was developed to run on generic, single processor (or parallel) systems with significantly less primary memory requirements than the previous versions.

Parka-DB is being used in a number of research initiatives at the University of Maryland. Some of the projects using Parka-DB include

- A **KDD** [20] application for a large medical knowledge base. In particu-

[1] Although it has been shown that IDO with fully general exceptions (i.e. cancellation links) is exponentially hard, we've demonstrated that IDO with multiple inheritance exceptions can be computed in polynomial time and is efficiently parallelizable [18].

lar, it has been noted that data mining systems can benefit from the use of KBs to provide semantic information for datamining [5]. We are exploiting this sort of information by using medical knowledge in the datamining of an OB-Gyn patient database containing records on over 20,000 patients. The current Parka-DB KB has over 1.2 million assertions.[2].

• Several **Case-based planning** applications including the memory-intensive case-based planning system, Caper, developed by Brian Kettler in his recent doctoral thesis [12]. The Caper KBs, the largest of which contains over 2,000,000 assertions, are described later in this paper (section 4). Parka is also being used to support a logistics planning project, jointly being developed with MITRE Corp. [9], with a KB currently containing over 250,000 assertions.

• A recent project involves the use of Parka as a hybrid knowledge and database for storing and retrieving designs and process plans for mechanical products, as part of ongoing effort to develop a new hybrid variant/generative approach to **process planning** in manufacturing. Parka's content-based retrieval of designs and plans is important in order to achieve the desired functionality of the process planning system, and will represent a significant advance over current "variant" approaches to process planning, which (among other things) involves retrieving process plans from databases based on fixed-length alphanumeric keys. The KB for this project is currently under construction, but is expected to dwarf those described previously, since current part databases are extremely large, containing information about the machining of thousands of parts.

3 Parka-DB

To support the demands of the extremely large KBs which are needed in applications such as those above, without requiring (but still allowing) the use of supercomputers, we have been using an increasing number of database algorithms to perform the inferencing in Parka. This led to a reimplementation of the system, which we call Parka-DB, in which the Parka KR sys-

[2]We report KB sizes in "assertions," basically the number of links in the semantic network corresponding to the frames, as this accurately reflects the total number of relations between items in the KB and corresponds directly to the "concepts" in a description logic representation.

tem is implemented directly using a relational database management system (RDBMS). The system thus blends the knowledge-based inferencing capabilities of Parka with the standard database capabilities of the RDBMS. In particular, Parka-DB uses the RDBMS as a run time storage medium. Thus, since the RDBMS uses external devices to store data, Parka-DB can manage knowledge bases that are too large to be maintained in internal memory. In fact, Parka-DB can handle KBs that are as large as available disk space, and can make use of the RDBMS to efficiently manage the I/O between primary and secondary storage. This allows Parka-DB to scale to essentially arbitrarily large KBs while still outperforming other KR systems.

KB-DB Interface To maintain efficiency in inferencing, we differentiate two categories of concepts: structural and non-structural assertions. Structural assertions consist of the ISA, INSTANCE, INSTANCEOF, and SUBCAT relations that encode the class/subclass ontology in a Parka KB. Non-structural assertions are all other concepts in the KB.

This differentiation is important from a practical viewpoint – the structural assertions are used in property inheritance. In particular, Parka-DB must scan the structural assertions for computing inherited properties, and inheritance evaluation is a critical aspect of all queries issued to Parka-DB. Thus, the structural assertions are kept in internal memory for fast access. In all of the cases we've examined, the number of structural assertions is less (often much less) than the number of non-structural assertions. Therefore, we can keep these structural assertions in internal memory even for our largest KBs (we discuss actual memory requirements in section 4).

On the other hand, the knowledge bases we are using in the previously described applications require more memory than is available on all but the largest current computer systems (particularly supercomputers). They certainly don't fit in the amount of internal memory expected to be found in "affordable" computer systems any time in the near future. Furthermore, we find that it is very unlikely that all of the data contained in the set of non-structural assertions will have to be processed for a given query (unlike the structural assertions, which are often all scanned). Therefore, the non-structural assertions are not required to be kept in primary memory at all times. For these reasons, Parka-DB stores non-structural assertions in specialized tables within the database. This allows the database system to handle

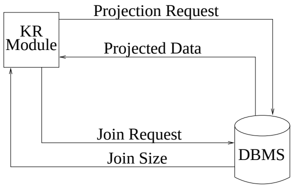

Figure 1: Communication between Modules

the loading of only that subset of non-structural assertions required to evaluate a given query, a process which DBMS systems manage efficiently. Thus, to summarize, by dividing data into two categories, Parka-DB can access the highly utilized structural assertions quickly, and rely on the database system to manage the much larger and less frequently accessed individual items of non-structural data efficiently.

To facilitate the efficient moving of data between the inferencing algorithms and the database, we have had to slightly modify the database system. This will be described in the next section, but first we wish to briefly clarify the interaction between the two components. Parka, as a KR system, is primarily concerned with answering queries. In the process of evaluating these queries, Parka issues two main requests to the database. Figure 1 illustrates the interaction between the modules. One request is a command to project data from a non-structural assertion table. The other is a request to perform a join over two tables.

In this manner, Parka-DB utilizes the processing power of the database for those services that a RDBMS can perform more efficiently, mainly join processing. In fact, Parka-DB's query algorithms have been designed such that many of the computations can be directly executed in the form of a join over two relational tables. Thus, the KR component of Parka-DB relies on the database not only for storage and retrieval services, but also for the intermediate processing of complex queries. In this manner, Parka-DB can also take direct advantage of database optimizations (for example, the use of parallelism for joins in a parallel DBMS).

Parka-REL Parka-DB is implemented on top of an RDBMS system we have created at the University of Maryland. This system, Parka-REL, offers the standard DDL interface for creating databases. Creating relational tables is straightforward and can be achieved through direct function calls from a C program or through a parser. To offer efficient access to user data, the system provides users the opportunity to create indexes on attributes of relational tables. Once the tables have been defined and created, users can seamlessly insert and delete records. Parka-REL provides storage and retrieval functionality, and all of the standard database operations required in the framework of Parka-DB. Parka-DB communicates with Parka-REL directly through the RDBMS's library of C function calls, rather than through SQL (although a limited SQL interface is provided for user access to the DB). The RDBMS, though limited in functionality, is very useful and supports Parka-DB adequately. We believe the limitations of Parka-REL are more than outweighed by the fact that Parka-REL can be made publicly available and thus Parka-DB distributed without the need for users to buy an expensive DBMS system. (We are, however, interested in some other database features not provided in Parka-REL, see Future Work.)

Data Storage One of the most important aspects of our implementation is that although Parka-DB is in all ways a frame-based KR system, it's knowledge base is actually stored as relational tables to take full advantage of Parka-REL's processing services. In particular, each assertion type is associated with a relational table in Parka-REL. Frames in Parka-DB thus have their information distributed amongst several tables.

This contrasts with other attempts to use database technology for supporting knowledge bases (see section 5). Other systems [11, 16] store frames in a much more centralized manner – they attempt to gather as much information about a frame as possible when loading data from external storage. Parka-DB only loads into memory that information which is needed for processing a query. In fact, we believe that one reason Parka-DB is more efficient than these other systems is that its inferencing algorithms have been designed to take full advantage of this distributed relational(frame) data.

When evaluating complex queries on large knowledge bases the sizes of intermediate results can be enormous. Computing the results of such queries using only internal memory can place a big demand on system resources, especially if the data to be processed is also stored in internal memory (as in most KR systems). Parka-DB overcomes this problem by distributing the workload such that the KR component prepares relational tables for the RDBMS component to process using external storage as it needs. The RDBMS unit can consume available disk space while processing intermediate results thus reducing the demand for valuable internal system resources.

4 Performance

As discussed above, the redesign of the previous version of Parka into Parka-DB was primarily to make greater use of secondary storage over primary memory (using the RDBMS). This clearly was expected to result in smaller use of internal memory and faster KB loading times (since less is moved from disk to memory) at the expense of greater querying times. In this section, we describe how Parka has performed on these issues.

Memory use Although all the information needed by Parka could be kept in Parka-REL and loaded into memory as needed, thus fully minimizing memory use, we keep the inheritance structure (structural assertions) in primary memory for efficiency reasons (as discussed previously). For the biggest KB we used in our tests (Caper-200, see below) the space required for this data was only 1.1 M-Bytes of RAM.

Additional use of memory is made by the inference engine which must maintain a small amount of information about each frame in the system. The space currently used is 12 Bytes per frame, and the number of frames is typically low compared to the number of assertions, so this also is not a major use of space. Where there is enough physical memory to guarantee this space Parka-DB executes most of the inferencing algorithms in memory. I/O is only performed to save intermediate and final query results in the DBMS.

Parka-REL also consumes physical memory to provide its services. The RDBMS allocates memory for a buffer pool with 40 buffers, each using 4 K-Bytes. This is a small amount of memory and the performance of Parka-REL would improve with more memory. A larger buffer pool and/or a bigger page size would allow Parka-REL to store more data in physical memory and reduce the amount of disk I/O, in situations where Parka-DB repeatedly requests the same information. Thus, including the resources consumed by the graphical query interface, Parka-REL, and Parka-DB, the total physical memory consumed never exceeded 16 M-Bytes of physical memory even for

Knowledge Base	#Frames	#struct. Links	#Asserts.	in-memory	Parka-DB
Caper-20	11103	26404	118615	28	0.29
Caper-100	56516	130526	807480	214	1.34
Caper-200	116297	266100	1635782	640	2.80

Table 1: Comparison of previous version and Parka-DB KB Loading time (in seconds)

the Caper-200 KB. As a comparison, holding this entire KB in the memory-based version would take 162MB of RAM, so we see a considerable advantage to the new version.

KB Loading times In the previous version of Parka the process of loading a KB into memory was very expensive. Since all the data had to be loaded into primary memory, the KB was kept in the form of a flat ASCII file containing the frame descriptions — there was no advantage to storing intermediate structures, since everything had to be read in. Instead, special hashing and other functions were developed to make the loading as efficient as possible.

Table 1 shows the sizes (in terms of frames and assertions) of the three KBs we have used in testing the system. All three of these were generated for the Case-based planning work described in section 2 and discussed in [12]. The case-bases were created via a generative planning system, and contain information on the plans, sub-plan structure, and causal information used by the planner, as well as ontology information about the logistics-planning domain. Details on this work and the planning domain can be found in

http://www.cs.umd.edu/projects/plus/Caper/ and
http://www.cs.umd.edu/projects/plus/UMT respectively.

As can be seen, even the biggest KB, which has over 2,000,000 assertions, can be loaded in under 3 seconds (over 200 times as quickly as when the whole thing is processed and loaded). This incredible improvement is created since in Parka-DB the KB is saved in a denser format which drastically reduces the number of I/O operations, since only a very small part of the KB is actually loaded.

Query	Preproc.	Inferencing		I/O		Total
		KB	DB	Input	Output	
Q1(Caper-020)	0.001	0.049	0.157	0.059	0.344	0.609
Q2(Caper-020)	0.001	2.258	0.652	0.139	0.929	3.978
Q1(Caper-100)	0.001	0.336	3.379	0.296	0.856	4.4867
Q2(Caper-100)	0.001	13.761	10.223	0.014	4.111	28.860
Q1(Caper-200)	0.001	0.590	12.329	0.713	1.893	15.575
Q2(Caper-200)	0.001	27.345	33.519	1.713	7.731	70.177

Table 2: Timings for query processing (seconds)

Query Processing Obviously, by making greater use of (slower) secondary memory we lose efficiency since we are trading space in primary memory for time. In this section, we analyze this trade-off directly, examining how much of the query time is spent in I/O, how much in the KB, and how much in the DB. Our analysis was based on a number of queries generated by the Caper system during its case-based planning. In this abstract, we show two of these, the fastest and slowest to process. Others were in between, with most closer to the faster times. The first query, which contains 10 conjuncts concerning 5 variables, corresponds to the query "Show all plans in which a train was used to move a package through Hub-TS", and the second query, with 8 variables and 15 conjuncts asks to "show all plans in which the package was a liquid package, was moved in a tanker truck, and was taken to a location that is of type 'city-location'." [3]

The process of querying the KB is divided into three phases. The query interface provided to the user is the same independent of the version of Parka used. For this reason issued queries are first preprocessed for evaluation by the appropriate version of Parka. Once the preprocessing is done the query is processed by a sequence of KB and DB operations. Based on table 2 we see the following:

• *Query Preprocessing* With Parka-DB some preprocessing is needed. In the first phase, the consistency of the query is checked. The second phase is used to define a join tree that specifies the order in which expressions in queries

[3] *Note to reviewers:* Full paper will include more queries and the actual queries will be shown using screen dumps of our graphical query interface - space limitations prohibit including these here.

are to be evaluated. During this process several optimization strategies are used to define a reasonable query evaluation order. Preprocessing is currently extremely fast (column 2), although we believe more time spent here can bring down the total, this is a direction of our current work.

• *Inferencing:* The timings for the inferencing part of query processing is divided into subparts, the amount of time spent by KB-operations (particularly IDO-inheritance) and the amount of time used by the DB-operations (selects, joins, ...). These timings are shown in column 3 and 4. The time spent by the DB-operations, in particular, could be reduced by using better strategies during preprocessing.

• *I/O Operations* The I/O operations are the overhead we must pay since the whole KB is no longer loaded into memory. The numbers in columns 5 and 6 show this time. The input cost is incurred by loading data into memory and by loading intermediate results of the query processing. The output cost is indicative of the processing time required by Parka-REL to write intermediate and final results. Its notable that the I/O process accounts for only approximately 10% of the total processing time. Based on these results we see that the additional I/O time caused by moving items between primary and secondary storage imposes only a small overhead on the system. In addition, significantly faster loading times are an added benefit for the databased reimplementation.

5 Related Work

Several research projects have dealt with the use of secondary storage to support large frame- based systems. Karp and Paley [11] discuss integrating database storage technology into KR systems. They pull in much data about individual frames, rather than distributing the frames as we discussed earlier. Thus, their system functions more as a "persistent store" than as a query system such as ours. The IDI (Intelligent Database Interface) project [16] can also be viewed in a similar manner, although that work is meant to deal more with access to distributed DBs then as a KR system per se.

Another system dealing with disk-based knowledge is Haase's Framer system [8]. Framer is an object oriented storage system designed to be used to implement a KR system. The system provides the basic routines for simple inferencing mechanisms, but provides neither a full KR language nor the sort of conjunctive querying provided by Parka-DB. Another alternate approach is that of Thenetsys [17] which is a semantic network system that employs somewhat of a virtual memory approach to semantic network management. In Thenetsys, semantic networks are stored on disk. When a node from a network is requested, Thenetsys has to find room for the node and its related data. If memory if nearly full, Thenetsys has to remove some nodes of a semantic network from memory, much like a virtual memory manager. Again, this system will not support the same sorts of conjunctive queries as supported in Parka-DB, nor have either of these systems yet been shown to scale to the size of the KBs being used at Maryland.

In fairness to these other systems, one problem with demonstrating these (and other KB systems) on very large KBs is the difficulty in obtaining these. In an effort to make such comparisons possible, the case-based KBs described in this paper are being made publicly available on the WWW - See http://www.cs.umd.edu/projects/plus/Parka

6 Conclusion & Future Work

Real world applications are demanding that KR systems provide support for KB's containing millions of assertions. We have presented Parka-DB, a high performance KR system based on implementing a standard KR using a relational DBMS system. The integration of a small DBMS (Parka-REL) with a standard KRS (Parka) allows us to efficiently support complex queries on extremely large KBs using a single processor. In addition, the system can make good use of secondary memory, with the whole system needing less than 16MB of RAM to hold a KB of over 2,000,000 assertions. We demonstrate empirically that this reduction in primary storage requires only about 10% overhead in time, and decreases the load time of very large KBs by more than two orders of magnitude.

Based on these results, we believe that the Parka-DB architecture is a cost-effective approach to scaling knowledge representation to very large KBs. Parka-DB is designed to consume as little internal memory as possible while using more (less expensive) disk space. Even though the price of RAM is falling, the price of RAM is still more expensive than disk space. By using less internal memory and more external memory, Parka-DB's utilization of system resources is quite efficient. Considering the efficient use of system resources and its high performance, Parka-DB offers a framework for knowledge representation that is adequate for managing huge KB's while not requiring extensive system resources.

We see two important directions to take this work. The first is to make more use of the database features than done to date. While Parka-REL has obvious advantages (it's home-grown and freely distributable), it doesn't have some features we believe might be of use in future KB systems. These include the security, concurrency, and user profiling features that many commercial RDBMS systems provide. Parka-DB has been designed to take advantage of such features if available, and we have begun discussions with various database researchers to explore the possibility of joint work in this direction.

The second direction is to improve the performance of our algorithms. As alluded to previously, more time spent in preprocessing should yield better performance. In recent experiments, we have found that we can improve efficiency by a full order of magnitude if we hand optimize the queries to minimize the size of internal results. This reduces the size of the inheritance scans (thus reducing the KB inferencing times) and also reduces the size of the relational joins (reducing the DB times). A CM-5 multiprocessor version of Parka [2] used heuristics to do just this sort of optimization, and we believe we can duplicate this work in the new framework at the cost of only a minimal amount of additional preprocessing time.

Secondly, Parka was originally designed for parallel supercomputing environments [6] and the algorithms have been proven to scale well to a wide range of supercomputing systems [19] guaranteeing a parallel efficiency better than 50% up to 16 processors for the case-bases described in this paper. In addition, we have recently prototyped a version of Parka's inferencing algorithms on the distributed Mosix system developed at Hebrew University [3] with speedups through more than 12 processors on an ethernet-based system. We are now working on porting Parka-DB to both parallel and distributed systems, and expect to see similar speedups.

Combining these two promising research directions with the low overhead of the I/O times shown in this paper, we are commited to two performance goals for the Parka-DB system: to be able to handle queries of the complexity of those shown in this paper in sub-second times for KBs of several million assertions (by the end of 1996), and to scale the system to handle these queries on KBs containing over one hundred million assertions (by the end of 1997).

References

[1] *Proceedings of the 12th National Conference of the American Association for Artificial Intelligence*. AAAI Press, MIT Press, 1994.

[2] W. Andersen, M. Evett, J. Hendler, and B. Kettler. *Massively Parallel Matching of Knowledge Structures*. Massively Parallel Artificial Intelligence. AAAI/MIT Press, 1994.

[3] A. Barak, O. Laden, and Y. Yarom. The now mosix and its preemptive process migration scheme. *Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments*, 7(2):5–11, 1995.

[4] A. Borgida and P.F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1, 1994.

[5] Paolo Bresciani. The challenge of integrating knowledge representation and databases. In *KARP-95 Second International Symposium on Knowledge Acquisition, Representation and Processing*, pages 16–18. Auburn University, AL, USA, 1995. Extended version in IRST TR 9412-06, IRST, Povo TN (Italy).

[6] M. Evett, J. Hendler, and L. Spector. Parallel knowledge representation on the connection machine. *International Journal of Parallel and Distributed Computing*, 22(2), 1994.

[7] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[8] Kenneth Haase. Framer. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*. John Wiley & Sons, Chichester, New York, 1994.

[9] J. Hendler, K. Stoffel, and A. Mulvehill. *High Performance Support for Case-Based Planning Applications*. Advanced Planning Technology. MIT/AAAI Press, Menlo Park, CA., USA, May 1996.

[10] J.F. Horty, R.H. Thomason, and D.S. Touretzky. A skeptical theroy of inheritance in nonmonotonic semantic networks. Technical Report

CMU-CS-87-175, Carnegie Mellon, Department of Computer Science, Pittsburgh, PA, USA, Oct. 1987.

[11] Peter D. Karp and Suzanne M. Paley. Knowledge representation in the large. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 751–758. Morgan Kaufmann, San Francisco, CA, 1995.

[12] Brian B. Kettler. *Case-based Planning with High-Performance Parallel Memory*. PhD thesis, University of Maryland, College Park, November 1995.

[13] K. Knight and S. Luk. Building a large knowledge base for machine translation. In AAAI-94 [1].

[14] Douglas B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems*. Addison-Wesley, 1991.

[15] Robert M. MacGregor. A description classifier for the predicate calculus. In AAAI-94 [1].

[16] D.P. McKay, T.W. Finin, and A.O'Hara. The intelligent database interface: Integrating ai and database systems. In *Proceedings of the 1990 National Conference on Artificial Intelligence*, pages 677–684. Morgan Kaufmann Publishers, 1990.

[17] Edward Pattison-Gordon. Thenetsys: A semantic network system. Technical Report DSG -93-02, Decision Systems Group, Harward Medical School, Boston, Massachusetts, 02115, 1993.

[18] K. Stoffel and J. Hendler. An efficient inferential-distance-ordering algorithm for parallel and distributed systems. submitted for publication, 1996.

[19] K. Stoffel, J. Hendler, and J. Saltz. Parka on mimd-supercomputers. In *3rd International Workshop on Parallel Processing in AI*, Montral, Canada, August 1995. IJCAI.

[20] Merwyn Taylor. Hybrid kb-db. In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence*. AAAI Press, MIT Press, 1996. to appear.