

DOES LOOSE AI-DBMS COUPLING STAND A CHANCE?

Amit P. Sheth
Unisys West Coast Research Center
2400 Colorado Avenue
Santa Monica CA 90406

The need to integrate knowledge processing and data processing have been realized for many reasons, including (1) access to large amounts of data for knowledge processing, (2) need for efficient management of data as well as knowledge, and (3) need for intelligent processing of data. Researchers and developers have been looking at various architectures for supporting their needs. Technologies for AI systems, particularly expert systems, to support knowledge processing needs, and for Database Management Systems (DBMSs), to support data processing needs, have significantly matured. Three approaches have been investigated to support both knowledge processing and data processing:

- Extend knowledge processing systems to make them efficient at accessing and managing large stored databases.
- Extend DBMSs to have advanced knowledge representation and inferencing capabilities.
- Integrate the two types of systems.

The first approach will need to duplicate the DBMS technology or will amount to building a new generation knowledge based system such as LDL [Chimenti et al 87]. The second approach has been taken by several projects including POSTGRES [Stonebraker et al 87], but the ability to perform inferencing and the knowledge representation capabilities have been limited. Neither of the first two approaches can use existing DBMSs, which may often be required due to significant investments in existing DBMSs and databases. Both the expert systems and the DBMSs have attained a level of maturity and integrating them allows taking immediate advantage of these existing technologies as well as future advances in them. The third approach offers wider applicability and flexibility. However, it involves two cooperating systems which may not perform as efficiently as a monolithic system.

There have been several efforts in integrating the two types of systems. However, most of them either use Prolog to access a relational DBMS [Marque-Pacheu et al 84, Jarke et al 84, Ceri et al 86, Ioannidis et al 88]; or provide a simple interface between a commercial expert system shell and a relational DBMS (e.g., KEE-connection [Abarbanel and Williams 86]). Exceptions include the DADM [Kellogg and Travis 81] and the Flexible Deductive Engine (FDE) [Van Buer et al 85]. The emphasis of the Prolog-DBMS interfaces has been on easing the mismatch between the tuple oriented Prolog access and the set oriented relational database operations.

Integration of the two types of systems have been classified as *tight* or *loose*. As the position statements of the panelists of this panel confirm, there is also no agreement on the definition of what is a *tight* or a *loose* integration. Indeed there is a range of solutions between the two extremes. The integration type can be said to be a function of, among other things,

- how much one system knows about the working of the other system,
- what types of information are shared or exchanged, and
- in case of preexisting systems, what changes are made to the systems to facilitate its integration with the other system.

To facilitate comparison among the opinions, we define an integration to be loose if the following hold:

- There is a well defined interface between the two systems. One system knows nothing about the operation of the other system except for what is specified in the interface.
- If a system being integrated is preexisting, it is not changed or the changes are very minor. After the integration, the two systems continue to maintain separate identities.

The purpose of our panel is to examine the scope of loose integration. Specifically, we intend to discuss the following issues:

- (a) When is a loose integration a more desirable approach? Which criteria such as applications, preexisting environment, and organizational factors, make a loose integration the preferred approach? What functions may be provided by a loosely coupled system? What types of applications can be supported?
- (b) What types of systems have been and can be loosely coupled? The majority of the reported work involves integrating a logic based system, often Prolog, with a relation DBMS. What about other AI or knowledge based systems? What are the desired functionalities of the systems being integrated?
- (c) What are the possible architectures and interfaces to couple systems loosely? Which techniques can be used to improve performance? What ranges of performance can be expected from different architectures? When and how do loosely coupled systems fall short of application requirements?

Based on the discussions, the panelist are likely to adopt some of the following positions (and a few more not mentioned here):

- Loose coupling has to work- there is a political and economic necessity.
- Very loose coupling just can't work because what an AI system system wants is very different from what (at least current) DBMSs provide.
- Very loose coupling can work only if current AI and DBMS systems change their interfaces and provide additional functionality.
- Loose coupling is a short term solution- it may meet some of the immediate needs but falls short in terms of the expressiveness, uniformity in representation, and/or efficiency.
- There are several alternative promising architectures that can provide varying levels of functionality and efficiency.

A Cache-based Architecture

In the remainder of this statement, I will focus on the last position mentioned above by briefly discussing a promising architecture we are designing and implementing at Unisys in the "Knowledge Management System" project funded by DARPA.

Our objective is to develop an interface that is useful for generalized database access from an expert system. Since we have chosen a logic based environment, we intend to experiment with a set of inference engines that vary in terms of the *degree of compilation*. The *interpreted - compiled range* (I-C range) [O'Hare and Sheth 88] provides a simple framework for understanding a variety of possible approaches to the coupling of expert systems and DBMSs. In particular, it allows focusing on critical performance related parameters of the integrated system, such as complexity of the queries sent to the DBMS, frequency of queries, and access patterns. In our architecture, the application and the inference engine will be located on a workstation, and a large database will be located on a remote relational DBMS accessible through a communication subsystem.

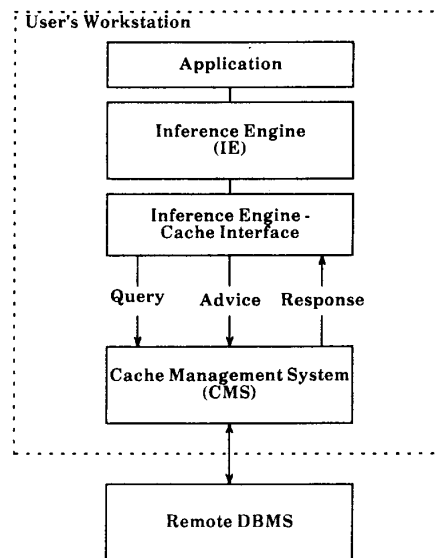
There are significant differences between the data access needs of an inference engine and the data access functions provided by a relational DBMS. Simple integration techniques will undoubtedly yield a system that is too slow and impractical for all but the least demanding KMS applications. Our emphasis is on providing efficient access to a large remote database by an expert system application on a workstation. Use of a cache is the central feature of our architecture. The cache management system (CMS) is expected to be instrumental in providing efficient access by integrating an inference engine controlling the execution of an application and the DBMS managing a remote database. As explained below, our cache is very different from cache memories.

Besides the use of a cache, other significant features of our project include use of:

- a DoD relevant realistic application,
- a large database,

- several integration strategies along the I-C range, and
- global information (called advice) provided by the inference engine that is used by the CMS in query planning and optimization.

Figure below shows the system architecture. The system is comprised of four main components: an application, an inference engine, a CMS, and a remote DBMS. The application is assumed to require inferencing and access to a large database. The inference engine will be capable of supporting several different inference search strategies to emulate points along the I-C range. The CMS manages a cache of database objects and is used to improve the efficiency of accessing data from the remote DBMS. The remote DBMS manages a large database (larger than 100 Mbytes). It is assumed to be relational due to the prevalence and popularity of relational DBMSs as well as because of elegance, generality and a good fit of the relational model to logic based systems. The application, inference engine and the CMS are assumed to be located on a workstation connected to the remote DBMS via a communication subsystem.



The cache has been implemented in the virtual memory of the workstation [Sheth et al 88]. The application along with the inference engine also reside on the workstation and share its resources. The main functions of the CMS are:

- interfacing with the inference engine,
- interfacing with the remote DBMS,
- managing the cache of database objects, and
- planning and executing queries.

The techniques employed by the CMS to improve the efficiency of data access and query processing include the following.

- Cache relevant data and thus reduce communication and other overhead of accessing the remote DBMS.

- Cache results of the data accessed by the inference engine thus eliminating the need to recompute frequently accessed results. Use advice from the inference engine to improve reusability of the data.
- Perform database operations on the cached data so that the cached data can be used for different queries, thereby improving the reusability of the data. Performing some operations in the cache will be more efficient in some cases than performing them in the remote DBMS or the inference engine.
- Perform some operations that are not supported by the remote DBMS. This will be more efficient in some cases than to perform them programmatically in the application. Operations can be evaluated in an eager manner (generate all extensions as in a DBMS) or in a lazy manner (use a *generator* to calculate next tuple in result on demand).
- Allow parallel execution of a subquery on the cache and a subquery on the remote DBMS. The subquery on the cache performs operations on the cached data at the same time a subquery on the remote DBMS accesses required data not in the cache.

References

- [Abarbanel and Williams] R. Abarbanel and M. Williams, "A Relational Representation for Knowledge Bases," Technical Report, Intellicorp, Mountain View, CA, April 1986.
- [Ceri et al 86] S. Ceri, G. Gottlob, and G. Wiederhold, "Interfacing Relational Databases and Prolog Efficiently," Proc. of the 1st Intl. Conf. on Expert Database Systems," South Carolina, April 86.
- [Chimenti et al. 87] Chimenti, D., A. O'Hare, R. Krishnamurthy, and C. Zaniolo, "An Overview of the LDL System," *IEEE Data Engineering*, vol. 10, no. 4, December 1987.
- [Ioannidis et al 88] Ioannidis, Y., J. Chen, M. Friedman, and M. Tsangaris, "BERMUDA -- An Architectural Perspective on Interfacing Prolog to a Database Machine," in *Proceedings of the Second International Conference on Expert Database Systems*, April 1988.
- [Jarke et al. 84] Jarke M., J. Clifford, and Y. Vassiliou, "An Optimizing Prolog Front-End to a Relational Query System," in *Proceedings of the 1984 ACM-SIGMOD Conference on the Management of Data*, Boston, MA, June 1984.
- [Kellogg and Travis 81] Kellogg, C. and L. Travis, "Reasoning with Data in a Deductively Augmented Data Management System," in *Advances in Data Base Theory*, ed. H. Gallaire, J. Minker and J. M. Nicholas, vol. 1, pp. 261-295, Plenum, New York, NY, 1981.
- [Marque-Pacheu et al. 84] Marque-Pacheu, G., J. Martin-Gallausiaux, and G. Jomier, "Interfacing Prolog and Relational Data Base Management Systems," in *New Applications of Data Bases*, ed. E. Gelenbe, Academic Press, 1984.
- [O'Hare and Sheth] A. O'Hare and A. Sheth, "The Interpreted - Compiled Range of AI/DB Systems," Technical Memo (unpublished), Unisys Corp., July 1988.
- [Sheth et al 88] A. Sheth, D. Van Buer, S. Russell, and S. Dao, "Cache management System: Preliminary Design and Evaluation Criteria," Technical Report TM-8484/000/00, Unisys Corp., CA, October 1988.
- [Stonebraker et al 87] M. Stonebraker, E. Hanson and S. Potamianos, "A Rule Manager for Relational Database Systems," in *The Postgres Papers*, M. Stonebraker and L. Rowe (eds), Memo UCM/ERL M86/85, Univ. of California, Berkeley, 1987.
- [Van Buer et al 85] D. Van Buer, D. McKay, D. Kogan, L. Hirschman, M. Heineman, and L. Travis, "The Flexible Deductive Engine: An Environment for Prototyping Knowledge Based Systems," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles CA, August 1985.