During the last few years intensive discussions have developed on whether the coupling of knowledge-based systems with large underlying information bases should be loose or tight. One of the immediate problems of course is that nobody gives a precise enough definition on what loose or tight coupling actually means. From completely separate systems that communicate via messages to a homogeneous knowledge management and storage system many variations can be presented.

For the purpose of our discussion let us define that loose coupling means that between a knowledge-based system and a (conventional) database system shared main memory exists. From the point of view of the knowledge-based system this memory contains facts, rules, meta-rules and work spaces. Of e.g., a relational database system, it contains relation segments, tuples and fields. To manipulate such a database, database commands (e.g., SQL for a relational system) will be used.

Tight coupling on the other hand means that the database system not only is able to handle generic knowledge-oriented functions (e.g., dynamic structuring of data, inferencing mechanisms) but also application-oriented operations. An indexing function to create bibliographic references in the face of scanned text, an image analysis mechanism that identifies and stores in the database objects found in video images, or a chemical data retrieval function that works on multi-media representations of chemical data, are examples of such operations.

Two questions arise:
1. Is it possible to achieve tight coupling of the kind defined above?
2. Is it of some benefit (e.g., for efficiency, flexibility, ease of use) to reach for tight coupling?

The first question has to be qualified immediately. Over the years we moved from machine code to assembly language, to procedural languages, to nonprocedural (4th generation) languages, and now to non-language or natural language-oriented application building tools. Each step pushed more of the algorithmic specification responsibility into the computing system. The first step addressing, the second register storage and instruction selection, the third execution control functions and distributed processing facilities, and the last modularization, packaging and interconnection. Unfortunately, the external data store (and the inter-node communication mechanism) took no part in this development. From the computational complexity point of view there is no reason to keep them out as all complexity and computability problems already can arise with simple machine coded programs. It was I believe more a question of divide and conquer a complex problem. But now we have conquered it and why should we not achieve a unified empire?

Loose coupling between applications and the database have been shown in the past to have sometimes intrinsic performance problems. One frequently explored area is CAD/CAM where the use of relational databases always leads to unacceptable performance and specialized solutions have to be adopted. Of course here we deal with a procedural language on one side (e.g., FORTRAN) that supports only navigational processing of the data. In nonprocedural knowledge-oriented languages (LISP, PROLOG), the argument goes, that will not happen. But with loose coupling to be efficient we have to be able during processing to automatically generate SQL statements that in one step retrieve as many as possible of the
data that will be relevant in the near future. LISP relies heavily on list manipulation, an intrinsically sequential process whether we talk about sequences, trees or even graphs. Always a thread has to follow in sequence even if parallel threads can be processed in parallel. If there is too much branching out, what data will be relevant? In the case of PROLOG (even including the control flow-oriented optimization features) backouts, wrong inferencing steps, etc. will be the rule. To bring relevant data into memory faces us with the problem of transporting either few at a time or running the risk of bringing a large amount of irrelevant data with them.

If we include the respective inferencing mechanisms into the database then the selection of relevant data will be done much closer to the data store and can be handled much more effectively. For example, by adding predicate-oriented control, sharing, recovery, retrieval and storage functions at the internal level of the database system, transport of irrelevant data can be reduced considerably, even in the database system itself.

Adding operations, in the sense of abstract datatypes or object-oriented systems, as we propose, allows the database system to go even beyond the above optimization. In effect, it will know where the data are kept externally, how they are structured there, what they mean (in the sense of a semantic model-oriented typing system), and even how they are going to be manipulated (in the sense of an abstract datatype or object-oriented system). As a consequence, optimization techniques that are now largely restricted to the program and main memory components of an application will be applied to the total program, main memory, secondary storage, and, if included, even the source/sink and communication environment.

In addition to the gain in performance, the tight integration of all system components will result in a homogeneous treatment of many of the application specific system components of today. Different applications will be easier to integrate, homogeneous user interfaces will become the rule instead of the exception, and the sharing of components between different applications will reduce the system development effort considerably.

Before I close, however, a note of warning. Yes, if we could start all over, such a homogeneous integrated and powerful solution should be followed by everybody. But we live in the real world. Many sometimes excellent, sometimes atrocious, solutions already exist. We have to live with them. Therefore, we should follow the above goal and strive to achieve it, but a large number of us will be faced with the problem of interfacing already existing solutions with each other, making the best of sometimes ad-hoc solutions and, in general, keeping our computing environment running. A transition to the integrated - tightly coupled, if you will - solution, however, is necessary when we want to handle the huge increase in complexity and volume of information that "intelligent" systems of the future have to handle. Of course for me "intelligent system" means nothing more or less than that a system does the expected and does not do the unexpected at whatever level of functionality that may be.