The Role of Databases in Knowledge-Based Systems

Mark S. Fox and John McDermott

CMU-RI-TR-86-3

Computer Science Department
and
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

February 1986

This research was supported, in part, by Digital Equipment Corporation, Air Force Office of Scientific Research under contract F49620-82-K0017, and Westinghouse Electric Corporation.

This paper will appear in *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, M. L. Brodie and J. Mylopoulos, Eds., Springer-Verlag, 1986.

Table of Contents

List of Figures

ABSTRACT

This paper explores the requirements for database techniques in the construction of knowledge-based systems. Three knowledge-based systems are reviewed: XCON/R1, ISIS and Callisto in order to ascertain database requirements. These requirements result in the introduction of the *Organization level*, an extension to the symbol and knowledge levels introduced by Newell. An implementation of these requirements is explored in the SRL knowledge representation and problem-solving system.

1. Introduction

This paper explores the requirements for database techniques in the construction of knowledge-based systems (KBS). While early work in Artificial Intelligence (AI) has focused on techniques such as representation and problem-solving, scant attention was paid to the issues to which database (DB) research has focused (e.g., data sharing, query optimization, transaction processing). Our principal premise is that although it has appeared that there was little intersection between the particular focus of each group, there is a significant overlap in needs. The maturing of AI techniques has recently led to their application outside of the laboratory, thus thrusting upon them problems requiring DB solutions. On the other hand, DB needs have expanded to include more expressive data models and more powerful query languages (e.g., supporting inference).

To ascertain KBS requirements for DBs, three KBSs are described. Each is analyzed from the perspective of the symbol and knowledge level concepts developed by Newell [NEWE81]. Limitations inherent in this perspective are identified. A new level, the organization level, is proposed as a means of identifying and dealing with these limitations. Lastly, we discuss implementations of some of the requirements in the SRL knowledge enginering system.

2. Database Requirement Analysis

In 1979-80 work began in both the Computer Science Department and Robotics Institute at Carnegie-Mellon University exploring the application of AI to engineering and manufacturing problems. Over the years at least 15 KBSs have been created and are in various stages of development and production use. From this set we have chosen three systems which exhibit a variety of DB requirements.

2.1. R1: Computer Configuration

The configuration task that R1 performs takes, as input, a list of components ordered by a customer and produces, as output, a set of diagrams that display the interrelationships among those components. If the initial list of components is incomplete (i.e., it may not be possible to configure a functional system with that set of components) the configurer adds appropriate components in the course of determining what the interrelationships among the components should be. Because the various combinations of components that might be ordered is too large to enumerate, the only approach to a configuration task of this type is to construct (as opposed to select) an appropriate system configuration.

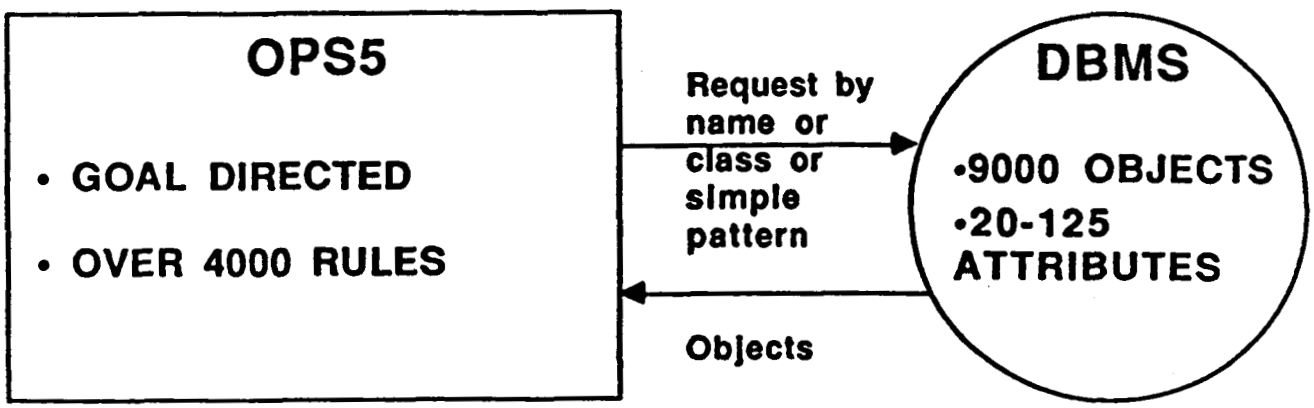
It would appear that constructive tasks require heuristic search, that is, a combinatorial search in

which candidate partial solutions are constructed and their potential evaluated. The nature of R1's knowledge of its task is such that R1 can for the most part avoid the combinatorial search (and thus avoid backtracking) by using small local searches at steps where there is ambiguity about what next action is most appropriate [MCDE82]. In other words, local cues are ordinarily sufficient to drive R1 along a path to a solution. R1's search techniques make only one sort of informational demand on its environment: it requires information that will allow it to determine which of several competing actions within the same or within competing subtasks ought to be performed next. Viewing the task from this perspective, there are at most two roles that knowledge can play. One of the roles is as selector of the next action within some subtask. The other role is as initiator of a local search when there is ambiguity about which subtask and which action to select.

R1's problem-solving method always selects the next piece of knowledge (i.e., rule) to apply from among those pieces of knowledge associated with the currently active subtask. Ordinarily only a few pieces of knowledge are relevant at any given time; a piece of knowledge is considered relevant whenever the pattern defining its relevance can be instantiated by elements describing the current state of the world. When more than one piece of knowledge is relevant, the problem-solving method relies on very general heuristics such as the recency of the elements instantiating each pattern and the degree of specificity of each pattern to determine which piece of knowledge to apply. Thus R1's problem-solving can be characterized as follows: Given that it's involved in some task, it will take whatever next step (ie, apply whatever knowledge) is relevant (and the step could be to initiate some subtask); if more than one piece of knowledge is potentially relevant, the choice of which step to take will be made on the basis of very general considerations. If there is no more knowledge relevant to the current task, R1's attention returns to the parent task. Whenever R1 does not have enough information to confidently prefer one possible step to all other candidates, it does some local problem-solving (searching) until sufficient information has been collected.

R1's rule base is written in OPS5 [FORG81] and contains over 4000 rules. It performs goal-directed forward chaining in order to synthesize a configuration. Information on Digital's standard parts is contained in the DBMS database system. It contains over 9000 part descriptions with anywhere from 25 to 125 different attributes describing each part. A subset of R1's rule base implements an interface to the DBMS system. When parts and/or their attributes are required during the configuration process, the DB interface rules' actions query the DB and provide the retrieved information in the format required by OPS5 and R1. In order to support R1's processing the DBMS provides query processing on a transaction basis. In the situation where more than one R1 is running, the DBMS supports transactions from more than one process.

R1/XCON COMPUTER CONFIGURATION



A Sample Rule

IF: The current subtask is assigning devices to unibus modules
And there is an unassigned dual port disk drive
And the type of controller it requires is known
And there are two such controllers neither of which has any devices assigned to it
And the number of devices which these controllers can support is known

THEN: Assign the disk drive to each controller
And note that each controller supports one device

Figure 2-1: R1 Database Architecture

2.2. ISIS: Job-Shop Scheduling

ISIS is a knowledge-based system designed to provide intelligent support in the domain of job shop production management and control [FS84]. Job-shop scheduling is a "uncooperative" multi-agent (i.e., each order is to be "optimized" separately) planning problem in which activities must be selected, sequenced, and assigned resources and time of execution. Resource contention is high, hence closely coupling decisions. Search is combinatorially explosive; for example, 85 orders moving through eight operations without alternatives, with a single machine substitution for each and no machine idle time has over 10^{880} possible schedules. Many of which may be discarded given knowledge of shop *constraints*. At the core of ISIS is an approach to automatic scheduling that provides a framework for incorporating the full range of real world constraints that typically influence the decisions made by human schedulers. This results in an ability to generate detailed schedules for production that accurately reflect the current status of the shop floor, and distinguishes ISIS from traditional scheduling systems based on more restrictive management science models. ISIS is capable of incrementally scheduling orders as they are received by the shop as well as reactively rescheduling orders in response to unexpected events (e.g. machine breakdowns) that might occur.

The construction of job shop schedules is a complex constraint-directed activity influenced by such diverse factors as due date requirements, cost restrictions, production levels, machine capabilities and substitutability, alternative production processes, order characteristics, resource requirements, and resource availability. The problem is a prime candidate for application of AI technology, as human schedulers are overburdened by its complexity and existing computer-based approaches provide little more than a high level predictive capability. It also raises some interesting research issues. Given the conflicting nature of the domain's constraints, the problem differs from typical constraint satisfaction problems. One cannot rely solely on propagation techniques to arrive at an acceptable solution. Rather, constraints must be selectively *relaxed* in which case the problem solving strategy becomes one of finding a solution that best satisfies the constraints. This implies that constraints must serve to discriminate among alternative hypotheses as well as to restrict the number of hypotheses generated. Thus, the design of ISIS has focused on

- constructing a knowledge representation that captures the requisite knowledge of the job shop environment and its constraints to support constraint-directed search, and
- developing a search architecture capable of exploiting this constraint knowledge to effectively control the combinatorics of the underlying search space.

ISIS constructs schedules by performing a hierarchical, constraint-directed search in the space of alternative schedules. The search is divided into four levels: order selection, capacity analysis,

resource analysis, resource assignment. Each level is composed of three phases: a pre-search analysis phase which constructs the problem, a search phase which solves the problem, and a post-search analysis phase which determines the acceptability of the solution. In each phase, ISIS uses constraints to bound, guide, and analyze the search.

Level 1 is responsible for selecting the next unscheduled order to be added to the existing shop schedule. Its selection is made according to a prioritization algorithm that considers order type and requested due dates. The selected order is passed to level 2 for scheduling.

Level 2 performs a dynamic programming analysis of the plant based on current capacity constraints. It determines the earliest start time and latest finish time for each operation of the selected order, as bounded by the order's start and due date. The times generated at this level are codified as operation time bound constraints which serve to influence the search at the next level.

Level 3 selects a particular routing for the order and assigns reservation time bounds to the resources required to produce it. Pre-search analysis begins with an examination of the order's constraints, resulting in the determination of the scheduling direction (either forward from the start date or backward from the due date), the creation of any missing constraints (e.g. due dates, work-in-process), and the selection of the set of search operators (e.g., choose next operation, choose machine for the operation) which will generate the search space. A beam search is then performed using the selected set of search operators. The search space to be explored is composed of states which represent partial schedules. The application of operators to states results in the creation of new states which further specify the partial schedules under development. Depending on the results of pre-search analysis, the search proceeds either forward or backward through the set of allowable routings for the order. An operator that generates states representing alternative operations initiates the search; in which case it generates alternative initial (or final) operations.

Once a state specifying an operation has been generated, other operators extend the search by creating new states which bind a machine and/or execution time to the operation. A variety of alternatives exist for each type of operator. For example, two operators have been tested for choosing the execution time of an operation. The "eager reserver" operator chooses the earliest possible reservation for the operation's required resources, and the "wait and see" operator tentatively reserves as much time as available, leaving the final decision to level 4. This enables the adjustment of reservations in order to reduce work-in-process time. Alternative resources (e.g. tools, materials, etc.) are generated by other operators. Each state in the search space is rated by the set of constraints found (resolved) to be relevant to the state and its ancestors. This set is determined by

collecting the constraints attached to each object (e.g. machine, tool, order, etc.) specified by the state and applying resolution mechanisms. Each constraint assigns a utility between -1 and 1 to a state; -1 signifies that the state is not admissible, 0 signifies indifference, 1 maximal support. The rating of a state with multiple constraints is the mean of the utilities assigned by the constituent constraints, each weighted by the the importance of the assigning constraint.

Once a set of candidate schedules have been generated, a rule-based post search analysis examines the candidates to determine if one is acceptable (a function of the ratings assigned to the schedules during the search). A set of schedules may be found to be unacceptable due to the lack of satisfaction of the more important constraints. If no acceptable schedules are found, diagnosis is performed. First, the schedules are examined to determine a type of scheduling error and the appropriate repair. Intra-level repair may result in the re-instantiation of the level's search. Pre-analysis is performed again to alter the set of operators and constraints for rescheduling the order. Inter-level repair is initiated if diagnosis determines that the poor solutions were caused by constraint satisfaction decisions made at another level. Inter-level diagnosis can be performed by analyzing the interaction relations linking constraints. A poor constraint decision at a higher level can be determined by the utilities of constraints affected by it at a lower level. That is, a poorly satisfied constraint at a level 3 may be linked to a well satisfied constraint at level 2. Only by going back to level 2 and choosing another value for the constraint at that level, which is less optimal, can the level 3 constraint's utility be increased (e.g., adding more shifts at level 2, which is a costly decision, will increase throughput and reduce work in process time at level 3).

Level 3 outputs reservation time bounds for each resource required for the operations in the chosen schedule. Level 4 then establishes actual reservations for the resources required by the selected operations which minimize the work-in-process time.

In addition to incrementally scheduling orders for production as they are received by the shop, the ISIS search architecture can be exploited in a reactive manner. As unexpected events (e.g., machine breakdowns) cause disruptions in the existing shop schedule, ISIS needs only to reschedule the affected orders. This results in a minimal amount of change, and provides continuity in the shop schedules generated over time.

The application to which ISIS has been applied is the production of steam turbine blades. The plant has over 30,000 part numbers, 200 machines and over 800 orders active at any time. In addition, the knowledge base contains information about operations to produce each part number (about 10 operations per part), tooling, resources, and constraints. A separate schema in SRL, a knowledge

representation language, is required for each piece of information.

The ISIS knowledge base must support parallel access by plant staff other than the schedulers. These include managers, tooling personnel, resource personnel, marketing and sales. These accesses are made via separate terminals on the same or distributed computers. Accesses are both read and write and are addressed by schema name (i.e., unique record key). Hence a DBMS which support multi-read/multi-write access is required if more than one process is to be supported. The need for an extensive query language is somewhat reduced because of similar facilities provided in SRL directly.

2.3. Callisto: Project Management

A major portion of the product development cycle is consumed in the performance and management of activities. For example, in high technology industries such as the computer industry, thousands of activities are required to be performed in the design and prototype build of a new product. Poor performance or poor management of an activity can result in critical delays. If product development time is to be reduced, then better management and technical support should be provided to each of the activities.

The Callisto project [SFG85] [SRM86] examines the extension and application of artificial intelligence techniques to the domain of large project management. Managing large projects entails many tasks, including:

Activity Management: Plan generation, scheduling, monitoring and control of activities. This involves selecting activities, assigning resources to accomplish these activities, chronicling or recording the status of activities as they are performed, diagnosing deviations from the plan and repairing the plans.

Product Management: maintaining a current description of the product (which is usually the outcome of a project), and determining the effects of changes to its definition (e.g., engineering change orders).

Resource Management: acquisition, storage, and assignment of the many resources required to support a project.

A close observation of project tasks shows that errors and inefficiencies increase as the size of the project grows. The successful performance of project tasks are hindered by:

Complexity: due to the number and degree of interactions among activities (e.g., resources, decisions, etc.).

Uncertainty: of direction, due to the unknown state of other activities and the environment.

Change: in activities to be performed and products to be produced, requiring project flexibility and adaptability.

While CPM and PERT techniques provide critical path and scheduling capabilities, the bulk of the

ISIS
JOB-SHOP SCHEDULING

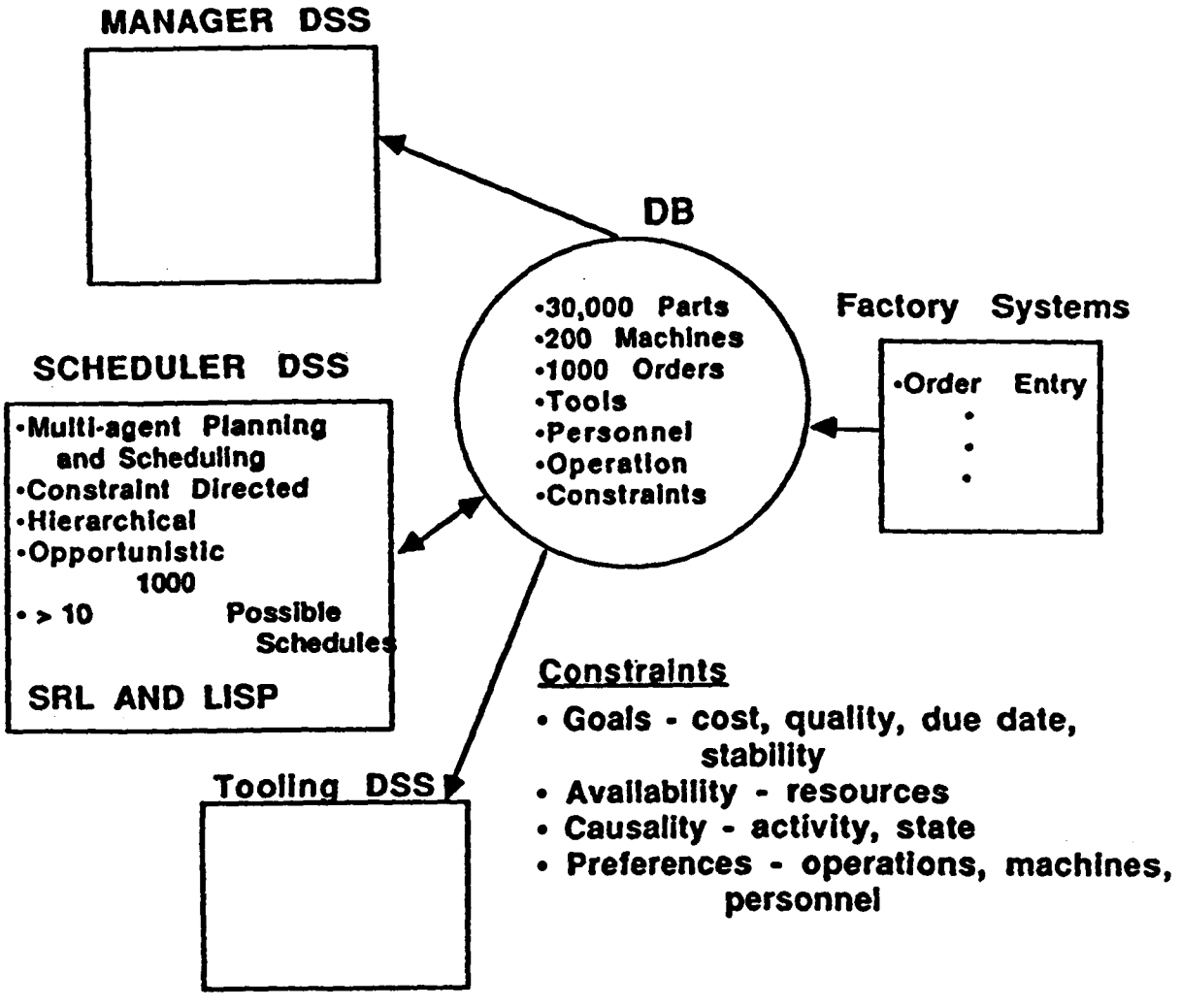


Figure 2-2: ISIS Process Architecture

tasks are performed manually.

Callisto provides decision support and decision making facilities in each of the above tasks. The ability to extend the capabilities found in classical approaches is due to Callisto's project model. Starting with the the SRL knowledge representation language, a set of conceptual primitives including time, causality, object descriptions, possession, organizational authority and responsibility are used to define the concept of activities, product and the project organization. The language is further extended by the inclusion of a constraint language which represents the constraints among activities. The modeling language provides Callisto with the ability to model both products and activities in enough detail that inferential processing may be performed.

Callisto's decision support and decision making capabilities include:

- interactive generation of plans,
- interactive change order management for products,
- multi-level scheduling of activities,
- chronicling of activity status,
- rule-based analysis and maintenance of activities,
- automatic generation of graphic displays of project models, and
- communication of project information among project members.

These functions are constructed from a combination of five problem-solving architectures:

Object programming: Each project concept is described in terms of its information contents, as well as procedures for a variety of operations (e.g., creation, deletion, display). Thus, manipulation of these objects is achieved by sending specific messages to these objects.

Robject-based: Robjects, or responsibility objects, are program modules capable of communicating with other modules, reasoning to generate a procedure for solving their problems and having mechanisms for generating, filtering and archiving messages. They form an organization structure within Callisto for handling a task given by the user.

Event/Agenda-Based: Callisto robjects interpret a user's process, represented as a network of activities and states, by setting up and maintaining an agenda of goals and monitored events. The agenda facilitates or inhibits the execution of certain actions depending on the goals. This processing facility is used in the scheduling and chronicling activities.

Rule-Based Programming: SRL-OPS, a production system language built on top of SRL, is used to implement managerial heuristics for project management. SRL-OPS can monitor and act on arbitrary conditions in project environment. The rule-based programming is used to evaluate activity structures and for the specification of status reports.

Logic Programming: HSRL, a Horne clause theorem prover, is used as a question answering mechanism for perusal and reporting. HSRL represents assertions and theorems within SRL.

Current research focuses on a distributed Callisto system. Each member of a project has a "mini-Callisto" to aid in managing his or her task. Each mini-Callisto is able to communicate with other mini-Callisto's to collectively manage the entire project. The project management tasks supported by the mini-Callisto network are:

- **Communication** of which phase the project is in, requests for participation, and commitment to a phase.
- **Definition** of activity networks for others. These networks define, at different levels of abstraction, the activities to be performed by each project member.
- **Distribution** of activity network to others.
- Record of **Negotiation** of the activities to be performed.
- **Monitoring** of employee performance by receiving status of milestones, and perusing activities.
- **Alteration** and extensions to previously defined activity networks.

The knowledge base for distributed Callisto does not reside in a central location. Instead, portions of it are distributed among the project members. Communication is performed via message sending. Parallel access to a single knowledge is not provided. Access is mediated by the owning process.

3. Analysis

This section identifies issues of common interest to the data base and artificial intelligence communities using the paradigm put forth by Newell [NEWE81]. The key idea is that an intelligent system can be viewed at a variety of levels; two of the levels identified by Newell are the symbol level and the knowledge level:

- ***Symbol Level***: the concern at this level is with what processing is required to bring relevant knowledge to bear in solving a problem in real time and real space. Thus the issues at this level are focussed around knowledge structures and mechanisms for accessing and maintaining these structures.

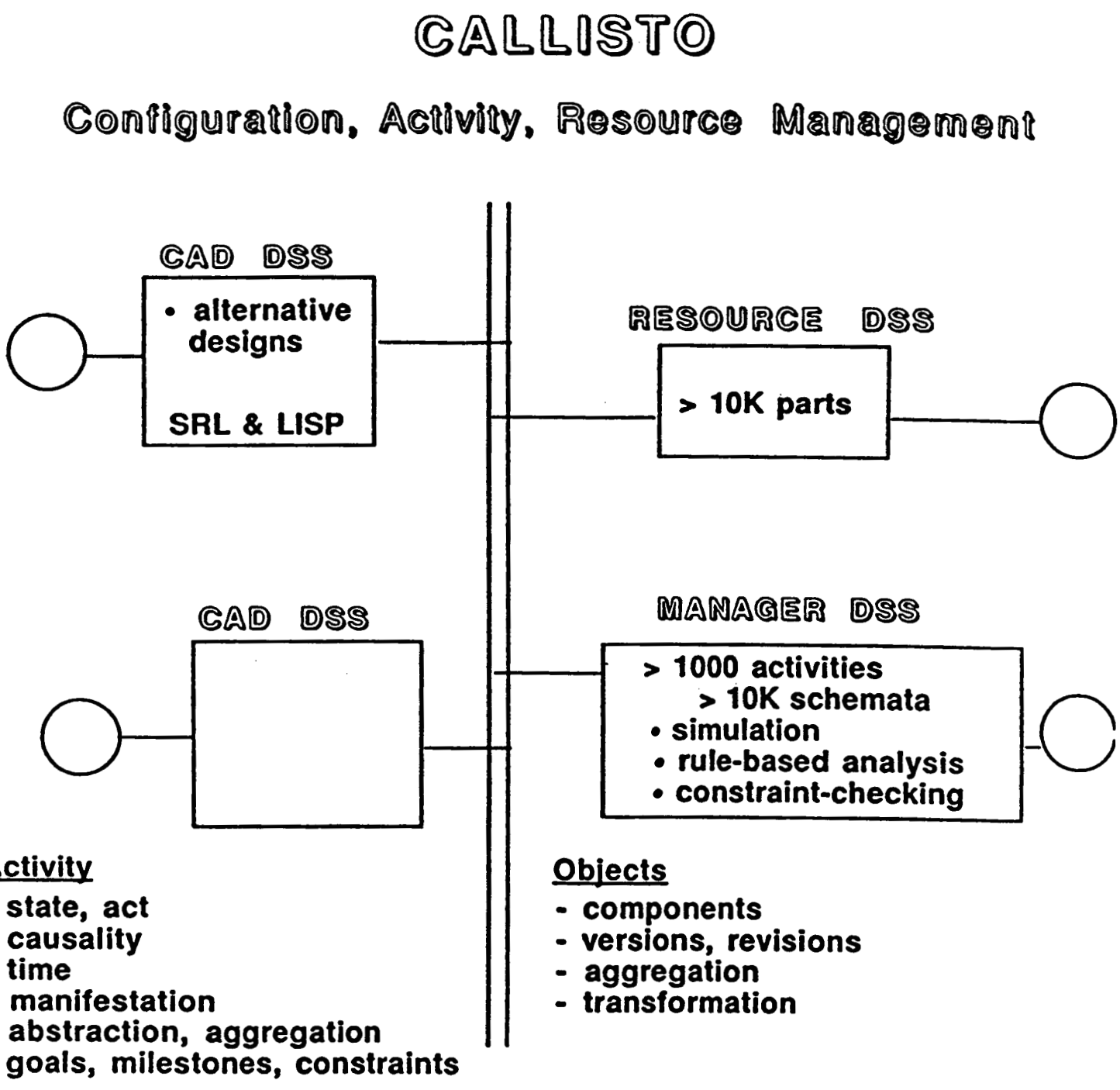


Figure 2-3: Callisto Distributed Architecture

- ***Knowledge Level***: the concern at this level is with what knowledge is required to solve a problem, rather than with how that knowledge can be effectively used. Thus here, the issues are focussed around the content of the knowledge base, rather than its structure

We share the views of Brachman and Levesque [BL84] that at the knowledge level, a DB is a limited form of knowledge base; its limits are determined by the functionality it can provide an agent. But if we examine the role DBs actually play in the world, it is clear that this knowledge level view is incomplete. Intelligent problem-solving tends to be a group effort involving more than one agent. A variety of issues in group problem-solving situations that are not considered at the knowledge level can only be resolved at a higher level, the organization level.

Organization Level: the concern at this level is with how agents, each with limited knowledge, can cooperate to solve problems.

Most DB research has focussed on providing capability at the organizational level and the symbol level. At the symbol level, the primary DB concern has been with time and space efficiencies. At the organization level research has focused on the support of access to a single DB my multiple agents (i.e., processes). Most artificial intelligence research has focussed on providing capability at the knowledge level and the symbol level. At the symbol level, the primary AI concern has been with flexibility and power. During the past several years, there has been a growing interest within the artificial intelligence community in issues at the organizational level and with issues of efficiency at the symbol level. Within the section we first point to some of the areas of DB research at the symbol level that AI researchers should attend to and then discuss the intersection of interests between the two communities at the organization level. The next section discusses ways that various results of data base research could be realized in the knowledge representation language, SRL.

3.1. Issues at the Symbol Level

Traditionally, efficiency has not been a criterion by which to measure the effectiveness of a system at the knowledge level. In fact, the knowledge level ignores the issue of how knowledge is to be structured by relegating it to the symbol level. Nevertheless, efficiency, particularly time efficiency, is a major problem in knowledge-based systems. There are two parts to the efficiency issue: efficiency of search and efficiency of data management.

3.1.1. Efficiency of Search

Consider the case of a frame or semantic network-based knowledge representation language, such as SRL [FWA84] or KL-ONE [BRAC77]. These representations implement a form of default reasoning by means of inheritance. The time to access a unit of information is dependent upon the amount of search that is to be performed in the network. Linear speed-ups in computer power do not match the

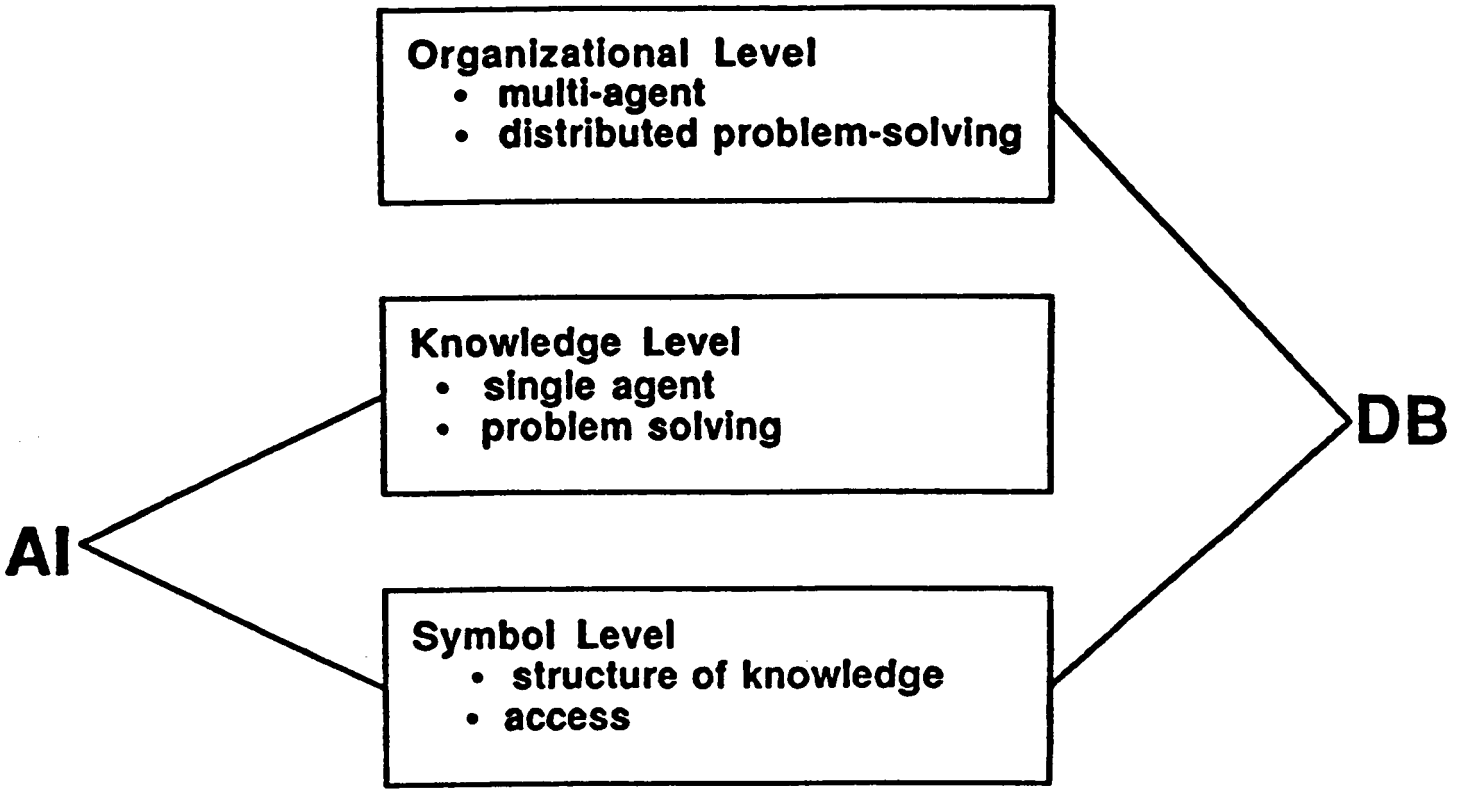


Figure 3-1: AI and DB Levels

possibly exponential growth in the search space of some knowledge networks.

Similarily, rule-based systems, in which the condition portion of rule is to be matched against data in working memory, may perform a great deal of search before a match is found. Systems such as R1, which has over 4000 rules and typically deals at any given time with 500 objects each of which may have up to 125 attributes [BM84], have begun to push the limits of conventional architectures. To date, symbol level solutions have been investigated. The RETE algorithm [FORG82] is one technique for compiling the patterns (i.e., conditions) of rules into a network in order to reduce the time to test whether a rule's condition is true. But this technique has serious shortcomings when the number of objects that need to be attended to at any given time (ie, the number of objects in working memory) is

large.

3.1.2. Efficiency of Data Management

The issue of how to manage large volumes of data has been largely ignored by AI. The approach traditionally taken is to increase the address space and store all data in virtual memory, letting the operating system worry about the rest.[1] With the application of AI to problems such as factory management, the need to store and access large volumes of data outside of virtual memory is critical. A portion of DB research has been concerned with managing the efficiency of access over a variety of storage devices.

The RETE algorithm, for example, assumes that the entire knowledge base fits in memory[2]. The issue of how to match patterns against large amounts of knowledge stored on a variety of storage devices is an issue that needs to be addressed. The problem can be broken into two parts: static and dynamic pattern matching. To the extent that a pattern is static, it is predefined and available at "compile" time for matching optimization. However, to the extent the pattern is dynamic, it is not defined until run time and is not available for compilation. DB research in indexing methods may play a significant role in solving this problem.

Inheritance and the more general relation-based search also raise data management issues. The cost of such searches increases dramatically when much of the knowledge is not contained in main memory but in a DB. If such operations are to be supported, then we must reduce the time to swap data in so that it appears invisible. Better DB fetching must also be sought. DB systems available today do not appear to provide a solution; access times to records are quite high relative to memory access times. Solutions which include parallelism, multiple levels of caching, and intelligent pre-fetch need to be investigated.

3.2. Issues at the Organization Level

Some AI researchers have already had to address issues at the organization level. For example, Ptrans [MCDE83], ISIS, and Callisto share information with other agents (or processes), and Hearsay-II's [EHLR80] knowledge sources share knowledge via a blackboard. Consequently, issues of security and synchronization as well as issues of incompleteness and inconsistency arise. At the knowledge level, one views a knowledge base as being "owned" by a single process. The problems

---

[1] Currently available lisp machines take this approach and do not offer DB support.

[2] The effect of the RETE algorithm, which works in main memory, is similar to materialized views in DBs which are supported in secondary memory.

of security and synchronization are removed by restricting access via messages. It is then up to the process to decide what information is to be seen, when, and by whom. But distributed problem solving systems require a variety of methods of sharing knowledge (e.g., blackboards) and thus require that the problems of security and synchronization be faced. At the symbol level, the problems of incompleteness and inconsistency are taken as a given. At the knowledge level, the problems require providing the system with the knowledge of how to qualify its judgements. But distributed problem solving systems are collections of agents the sum of whose knowledge may be more complete than that of any individual agent. Moreover, the nature of the inconsistencies in the sum of the knowledge may allow judgements of relative reliability. Thus at the organization level, the task of developing a system requires providing the system with the knowledge of how to extend its own knowledge by interacting appropriately with other agents.

3.2.1. Security and Synchronization

Blackboard mechanisms such as found in Hearsay-II provide a means by which a group of agents may jointly peruse and modify a common pool of information. Obviously, issues of synchronization occur. In addition, depending upon the role of an agent, limitations may be placed on what information may be seen and/or modified. DB security and synchronization techniques such as transaction models and optimistic concurrency control can be readily applied to shared knowledge structures such as blackboards or production system working memories. While it is at the organization level that the issue arises, it is at the symbol level that the solution will have to be implemented.

The problem of security appears in distributed systems which lack global data structures. If each agent possesses its own knowledge base and information flows only via message passing, how is an agent to determine whether to honor requests for information from other agents? The problem of synchronization also appears when agents cooperate to solve problems. How must an agent manage the synchronization of knowledge and actions of multiple agents when cooperatively working on a problem?

3.2.2. Incompleteness and Inconsistency

In a situation when multiple agents are problem-solving in parallel, each agent may generate new knowledge. To maintain full access to the generated knowledge, each agent may have to broadcast all it knows or make its knowledge available in a global knowledge base. The former may quickly result in saturation of the broadcast medium, while the latter may saturate the storage device and become an access bottleneck. Alternatively, an agent may not want all of the knowledge it generates available to the entire organization. In any of these cases, it is clear that one agent's model of the

"world" may rapidly become incomplete. Techniques are required to decide:

- what knowledge to store locally,
- what knowledge to communicate,
- to whom to communicate this knowledge,
- how to restrict access by other agents, and
- where to look for knowledge if you don't have it.

In the case when knowledge is shared among multiple agents and stored locally, an agent may extend or alter the knowledge in a manner which makes it inconsistent with another agent's understanding of the same knowledge. For example, an agent may share a description of a person with other agents, but during its problem-solving it may alter its description (e.g., guessing of the person's age) so that it is inconsistent. At some time during problem-solving, another agent may request that person's description from other agents and receive differing descriptions. How is this inconsistency to be resolved? It is not enough to maintain separate belief models; this only maintains a separation.

4. SRL

This section presents one knowledge representation and problem-solving system, SRL[3], and examines how it implements solutions to some of the above problems.

4.1. Language Overview

SRL is a frame-based language with the "schema" as its primitive. A schema is a symbolic representation of a single concept. Its definition is the summation of its slots and values. Slots are used to represent attributive, structural and relational information about a concept. A schema is composed of a schema name (printed in the bold font), a set of slots (printed in small caps) and the slot's values (lisp printing conventions are observed). Values can be any lisp expression and reference schemata when they are strings. When printed, a schema is always enclosed by double braces with the schema name appearing at the top. The **h1-spec** schema (figure 4-1) contains six slots, each of which contains a value.

---

[3]Development of SRL began in 1977. Its contents evolved during its application to industrial problems in the Intelligent Systems Laboratory. Since 1985, many of the concepts investigated in SRL have been implemented in Knowledge Craft, a knowledge engineering system available from Carnegie Group Inc, Pittsburgh PA.

```
{{ h1-spec
      IS-A: "engineering-activity"
      SUB-ACTIVITY-OF: "develop-board-h1"
      INITIAL-ACTIVITY-OF: "develop-board-h1"
      ENABLED-BY: "TRUE"
      CAUSE: "h1-spec-complete"
      DESCRIPTION: "Develop specifications for the cpu board" }}
```

Figure 4-1: h1-spec Schema

Many ideas found in other representation systems have been incorporated into SRL including meta-information, demons, restrictions on legal slot value and a context facility.

Meta-information may be associated with schemata, their slots, and values in the slots. It is represented by another schema, called a meta-schema, that is attached to the schema, slot, or value. Representing meta-information as schemata provides a uniform approach to representation. The user is provided with access functions for retrieving meta-schemata. Once retrieved, they are manipulated just as any other schema. The meta-information is printed in italics beneath schema, slot or value to which it is attached:

```
{{ h1-spec
      Creator: "mark fox"
      To-Create:: schemac

      IS-A: "engineering-activity"
      SUB-ACTIVITY-OF: "develop-board-h1"
          range: (type "instance" "activity")
      INITIAL-ACTIVITY-OF: "develop-board-h1"
      ENABLED-BY: "TRUE"
      CAUSE: "h1-spec-complete"
      DESCRIPTION: "Develop specifications for the cpu board" }}
```

Figure 4-2: h1-spec Schema

Any slot may have *facets* associated with it. Four facets are defined in SRL: DEMON, DOMAIN, RANGE, and CARDINALITY. The DEMON facet allows lisp procedures to be associated with a slot. The execution of demons is keyed to particular SRL access functions, such as filling or retrieving the value of a slot. RANGE and DOMAIN facets are used to restrict the values that may fill a slot and the schemata in which a slot may be placed, respectively. The CARDINALITY is used to restrict the number of values that a slot may contain. Values for each facet may be inherited from slots in other schemata.

As in other representation languages, a standard set of relations are provided to the user to form taxonomic and part hierarchies. Slots and values may be inherited automatically between schemata along these relations. One of the novel representational ideas introduced by SRL is *user-defined inheritance relations* [FOX79]. In most other knowledge representation systems, several relations for inheriting slots and values are defined as part of the representation (eg. AKO, is-a, virtual-copy). In contrast, SRL offers a facility by which users can define their own inheritance relations, allowing only slots and values of the user's choice to be inherited. In addition, slot structures can be elaborated between schemata, and slots and their values mapped arbitrarily between schemata, as need demands. Inheritance relations are represented by additional slots in a schema. A *dependency mechanism* is integrated into the inheritance facility that notes, as meta-information, the source of inherited slots and values. Here again, the user can define the dependency relations that are put into place.

Another novel feature provided by SRL is a means of *controlling the search* performed by the inheritance process. Any query of the model may optionally use a *path* to restrict which relations may be traversed while searching for a suitable value to inherit. Paths may also be used to specify the *transitivity* properties of relations. For example, a PART-OF hierarchy for describing a car might represent the **battery** as PART-OF the **electrical system**, and the **electrical system** is PART-OF the **car**. The implicit notion that the **battery** is PART-OF the **car** (ie. that PART-OF is transitive with itself) is represented using paths.

Contexts in SRL act as virtual copies of DBs in which schemata are stored. In the copy, schemata can be created, modified and destroyed without altering the original context. Contexts are structured as trees where each context may inherit the schemata present in its parent context. Hence, only schemata that are used in a context need be explicitly represented there. This avoids copying schemata that will never be used in the context. The context provides for version management and alternate worlds reasoning with SRL models.

Error handling is also schema-based. An instance of the **error** schema is created to describe each error encountered by the system. **error-spec** schemata may be defined that specify how to recover from each kind of error.

In order to support large applications, a simple DB system is integrated into SRL. The DB system provides two key access (i.e., context and schema name) to varying length records each containing a single schema. Concurrent access is supported only for a single writer and multiple readers. Schemata are stored in a DB until they are accessed, at which time they are brought into lisp. A

cache of the most recently accessed schemata are kept in lisp for quick access. When the cache becomes too large, schemata are swapped back to the DB using recency algorithm.

4.2. Extensions to the Language

SRL serves as the core of a knowledge engineering environment called Islisp [ISL84]. It offers a number of inference tools that operate on schemata: HSRL, PSRL, OSRL, ESRL, and KBS. HSRL [AW83] takes HCPRVR, [CHES80] a *logic program interpreter*, and alters it to use SRL models as its axioms. The system combines the modus ponens inference of logic programming systems with the representation power of SRL. In addition, the inheritance mechanism provides default reasoning, not available in logic programming environments. Similarly, PSRL is a *production rule interpreter* that operates on SRL models [RYCH84]. Production rules and their parts are represented by schemata. A subset of PSRL provides the form and execution pattern of OPS5 rules [FORG81]. OSRL provides a schema-based object programming. ESRL provides an event mechanism which enables the user to schedule events to occur either in a simulated or normal operating mode. KBS, a knowledge-based simulation system [RF82] uses ESRL to perform discrete simulations of systems modeled in SRL. Simulation objects are represented as schemata. An object's associated events and behaviors are represented as slots and values in the schema. An object's event behavior may be inherited along relations which link it to other schemata.

In addition to inference tools, system building tools are provided. RETINAS [GREE83] is a schema-based *window system*. Schemata for windows, displays, and canvases are instantiated to build an interface. Default specifications for windows, etc., may be inherited from the prototype schemata. KBCI [ISL84] is a schema-based *command system*. Again, the **command** schema is instantiated to create commands. A command interface is defined by a collection of command schemata organized in a SUB-COMMAND-OF hierarchy. CPAK [ISL84] is a 2D graphics package based on the CORE definition. A business graphics facility is provided on top of CPAK.

4.3. Analysis

The SRL system has served as both a production inference engine and an testbed for ideas to solve the problems raised in sections 2 and 3. Hence, we have had to worry about solving efficiency problems while investigating solutions to the more difficult incompleteness and inconsistency problems. In the following, we re-examine the problems raised earlier. Our "solutions" to these problems raise at least as many questions as they answer.

4.3.1. Efficiency of Data Management

Early on in the development of SRL, it became apparent that efficient access to large volumes of data would have to be provided to multiple agents. Applications such as job-shop scheduling (i.e., ISIS) and project management (i.e., Callisto) require very large DBs to be maintained, of which only a small portion is ever used by any agent at any given time. We searched for a DB system available under UNIX which had the following attributes:

- very fast two key retrieval (schema name and context) of varying length records, and
- supporting multiple readers and writers.

The search failed, so a DB system was created in house which provided b-tree access to varying length records while supporting multiple readers and a single writer. A re-design of SRL's internal structure was performed to integrate better the DB system with the knowledge representation. A layered approach was taken:

1. **Implementation Layer**: This is the core layer at which the structure of schemata are defined in terms of of nodes, links and metalinks. It is at this layer that indexing of static patterns is performed and maintained.

2. **Context Layer.** The definition of contexts and binding of schema/context pairs to schema structures is performed here.

3. **Database Layer.** Access to the DB is performed at this layer whenever a schema is not contained in virtual memory. Schema are cached in main memory on demand and are swapped out according to frequency of usage.

4. **Context Inheritance.** Inheritance of schemata between contexts is performed at this layer.

5. **Schema Inheritance.** Inheritance of slots and values is performed at this level.

6. **Distributed Knowledge Bases.** Retrieval from other separate knowledge bases is performed at this level.

The DB system has provided a significant extension to SRL's knowledge base. Large volumes of data can be maintained reliably without wasting virtual memory. Nevertheless, the cost of a DB retrieval is significantly more than a simple virtual memory access. Better methods for storing and retrieving large numbers of schemata are still required. DB machines may hold promise in this area.

4.3.2. Efficiency of Search

A version of SRL was constructed in which rule conditions were compiled into a RETE-like network. The network was indexed directly into schemata at the implementation layer. No indexing of schemata in the DB was performed. In the case of dynamic patterns, brute force search was performed and was slow. The problem of how to index patterns has not been fully resolved in a

general sense. With inheritance being a basic capability of a representation, indexing techniques have to be aware of changes not only within a schema but in schemata from which it may inherit slots and values. This is achievable in representations where inheritance is performed along classification relations (e.g., is-a, ako, virtual-copy) and slots and values are propagated to descendents upon creation. But for more complex inheritance such as found in SRL, the problem is much more difficult; slots and values are propagated on demand and inheritance is user definable and context sensitive. Either more dynamic pattern matching techniques are required, or restrictions will have to be placed on the functionality of the rule-base.

This still does not address the problem of whether to index schemata both in memory and in the DB, or only those in memory. An argument for the latter is that memory contains only those schemata which have been changed and require patterns to be checked. The problem here comes with the introduction of new rules: should the rule be indexed only in memory, or across the entire DB? The latter is a truly expensive operation for large DBs. We suspect a solution to this problem lies in parallel hardware specialized for knowledge base applications.

4.3.3. Security and Synchronization

The Callisto project provided us a context in which to investigate the issues of security and synchronization. We were concerned with the construction of a distributed project management system where each project member had his own version of Callisto to manage his own activities and those of the people for whom he was responsible (if any). Communication of activities to be performed and their status among these Callisto systems was required.

The DB system provided for single writer and multiple reader access with a semaphore for synchronization. In the case of multiple agents, each with its own non-shareable DB, each agent could send messages to each other requesting information. The basic information access functions of SRL were extended to include the name of the agent from which to retrieve the information. This resulted in a message being sent to the appropriate agent. The agent receiving the message then determined if the sender should have access to the information and sent the appropriate reply. While this mechanism was sufficient for security purposes -- an agent managed access to its own information -- it was insufficient with respect to synchronization.

The problem of synchronization arose at the application level. The process of developing project activity networks is iterative, requiring many agents to review, critique and modify them. Keeping track of the current stage of the process and activity network versions is a significant task. Much of the work we have performed in the area is nascent and is focusing on the development of protocols,

multi-agent review, critiquing, negotiation, and modification[4]. This issue is truely an organization level problem requiring further development of the concepts of communication, cooperation and negotiation.

4.3.4. Incompleteness and Inconsistency

The problem of incomplete and inconsistent knowledge arose also in the distributed version of Callisto. If during an agent's problem-solving a schema was referenced which was not contained in virtual memory nor in its DB, an attempt was made to communicate with other agents to determine the whereabouts of the schema in question. Three approaches were taken to guide the sending of messages:

1. A schema can be referred to only via another schema (i.e., the value of a slot). Since each schema contains meta-information describing the agent with whom it originated, the origin of the referring schema was used as the initial agent to communicate with.

2. Each organization contains a profile describing what *type* of schemata various agents in the organization contain. An agent looking for a schema can determine the corresponding agent by checking type description -- but only if they possess the type information to begin with.

3. Lastly, each agent possesses a list of "acquaintances". As a last ditch effort, it could send a message to each of them.

These three approaches provide a reasonably powerful means of finding a schema, but does not guarantee its retrieval. Much work remains at the organization level to solve this problem satisfactorily.

The concept of *agent responsibility* was proposed as a means of aiding in inconsistency reduction. For each schema an agent was assigned the responsibility of maintaining that schema's consistency. All problems concerning a schema's inconsistency would be resolved by the agent responsible. To support the agent's analysis, a record of transformations and communications of schemata was kept at the meta-level. If an inconsistency arose, it would be possible to trace the problem back to the source. Giving an agent the responsibility to resolve inconsistencies is only half a solution; how the inconsistencies are resolved is the other half, and that half requires further research.

---

[4] To appear as the thesis of Arvind Sathi.

5. Conclusion

The integration of Artificial Intelligence and Database technologies is a problem whose "time has come". As AI systems are extended to more complex problems requiring large volumes of information and knowledge, the need for DB support becomes apparent. The three examples described in this paper demonstrate both the ease and difficulty of this integration; R1, ISIS and Callisto all interface either loosely, via transactions, or tightly, via direct integration with the knowledge representation system. Though each demonstrates a successful integration of the technologies, issues of theoretical and practical importance arise.

First, a tighter integration of knowledge and data representations are required in order to allow pattern matching and search to be performed more efficiently. Without a solution to these problems, a natural complexity barrier may arise which will restrict the size and complexity of intelligent systems.

Secondly, problems arise in the design of distributed systems. In particular, issues of incompleteness and inconsistency bring into question the efficacy of current representational theories. In particular, the theory of the knowledge and symbol levels needs to be extended to include a third level, the organization level, whose focus is on the multi-agent distributed problem-solving. A sound theory of the organization will ultimately lead to solutions in distributed problem-solving.

References

[AW83] Allen, B.P., and Wright, J.M., (1983),"Integrating Logic Programs and Schemata." *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany.

[BM84] Bachant, J. and J. McDermott, (1984). "R1 revisited: four years in the trenches". The AI Magazine, Vol. 5, No. 3.

[BRAC77] Brachman R.J., (1977), "A Structural Paradigm for Representing Knowledge," (Ph.D. Thesis), Harvard University, May 1977.

[BL84] Brachman R.J. and H.J. Levesque, (1984) "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level", Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island, SC.

[CHES80] Chester, D., (1980), "HCPRVR: an Interpreter for Logic Programs," *Proceedings of the National Conference on Artificial Intelligence*.

[EHLR80] Erman L.D., F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, (1980), "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty", *Computing Surveys*, Vol. 12, No. 2, June 1980, pp. 213-253.

[FORG81] Forgy, C.L., (1981) "OPS5 User's Manual", Technical Report, Department of Computer Science, Carnegie-Mellon University.

[FORG82] Forgy, C.L., (1982) "Rete: a fast algorithm for the many pattern/many object pattern match problem", Artificial Intelligence, Vol 19, No 1.

[FOX79] Fox M.S., (1979), "On Inheritance in Knowledge Representation", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 282-284, Tokyo Japan.

[FS84] Fox M., and S. Smith, (1984), "ISIS: A Knowledge-Based System for Factory Scheduling", *International Journal of Expert Systems*, Vol. 1, No. 1.

[FWA84] Fox M.S., J.M. Wright, and D. Adam, (1984), "Experiences with SRL: An Analysis of a Frame-based Knowledge Representation", *Proceedings of the First International Workshop on Expert Database Systems*, Kiawah Island, SC.

[GREE83] Greenberg M., (1983), "RETINAS User's Manual", Internal report, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA.

[ISL84] ISL, (1984), "Intelligent Systems Laboratory Software Systems Manual", Internal report, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA.

[MCDE82] McDermott, J. (1982), "R1: A Rule-based Configurer of Computer Systems", Artificial Intelligence, Vol. 19, No. 1.

[MCDE83] McDermott, J. (1983), "Extracting Knowledge from Expert Systems", Proceedings of IJCAI-83, Karlsruhe, West Germany.

[NEWE81] Newell A., (1981), "The Knowledge Level", The AI Magazine, Vol. 2, No. 2.

[RF82] Reddy Y.V. and M.S. Fox, (1982), "KBS: An Artificial Intelligence Approach to Flexible Simulation", CMU-RI-TR-82-1, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA.

[RYCH84] Rychener M., (1984), "PSRL User's Manual", Technical Report, Robotics Institute, Carnegie-Mellon University.

[SFG85] Sathi A., M.S. Fox, M. Greenberg, (1985), "Representation of Activity Knowledge for Project Management", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 5, September, 1985, pp. 531- 552. Also appeared as Technical Report CMU-RI-TR-85-17, Robotics Institute, Carnegie-Mellon University.

[SRM86] Sathi, A., T.E. Morton, and S. Roth, (1986), "Callisto: An Intelligent Project Management System", *AI Magazine*, to appear.