November 1986

Report No. STAN-CS-87-1147
Also numbered KSL-86-38

A Layered Environment for Reasoning about Action

by

Barbara Hayes-Roth, Alan Garvey, M. Vaughan Johnson Jr.,
Micheal Hewett

Department of Computer Science

Stanford University
Stanford, CA 94305

Knowledge Systems Laboratory
Report No. KSL 86-38

November 1986

A Layered Environment for Reasoning about Action

by

Barbara Hayes-Roth, Alan Garvey, M. Vaughan Johnson Jr., and Micheal Hewett

KNOWLEDGE SYSTEMS LABORATORY
Computer Science Department
Stanford University
Stanford, California 94305

The research was supported by the following grants: NIH Grant #RR-00785; NIH Grant #RR-00711; NSF Grant #DMB84-2038; NASA/Ames Grant #NCC 2-274; Goeing Grant #W266875; DARPA Contract #N00039-83-C-0136. We thank the following individuals for their contributions. Jeff Harvey, Bob Schulman, and Craig Cornelius worked on BB1. Russ Altman translated the PROTEAN knowledge sources from their original implementation in BB1 to their current implementation in BB1-ACCORD. Bill Pardee and Paul Rosenbloom made useful suggestions on presentation. Frederick Hayes-Roth provided constructive criticism at several stages of the work. We especially thank Bruce Buchanan and Ed Feigenbaum for sponsoring the work within the Knowledge Systems Laboratory.

Table of Contents

Abstract

An intelligent system reasons about--controls, explains, **learns** about--its actions, thereby improving its efforts to achieve goals and function in its environment **In** order to perform effectively, a system must have knowledge of the actions it can perform, the events and states that can occur, and the relationships among instances of those actions, events, and states. We represent such knowledge in a hierarchy of knowledge abstractions and impose uniform standards of knowledge content and representation on modules within each hierarchical level. We refer to the evolving set of such modules as the **BB*** environment. To illustrate, we describe selected elements of **BB*:** (a) the foundational **BB1** architecture: (b) the ACCORD framework for solving arrangement problems by means of an assembly method: (c) two applications of **BB1-ACCORD,** the PROTEAN system for modeling protein structures and the **SIGHTPLAN** system for designing construction-site layouts: and (d) two hypothetical ***multi-faceted systems*** that integrate ACCORD, PROTEAN, and **SIGHTPLAN** with other possible **BB*** frameworks and **applications.**

1. Overview: Four Themes

"Human intelligence depends essentially on the fact that we can represent in language facts &bout our situation, our goals, and the effects of the various actions we can perform." John McCarthy [35]

"In the knowledge is the power." Edward A. Feigenbaum [14]

"The fact, then, that many complex systems have a nearly decomposable, hierarchic structure is a major facilitating factor enabling us to understand, to describe, and even to 'see' such systems and their parts." Herbert A. Simon [48]

We be&in with a premise: ***An intelligent** system **reasons about its actions.*** Of course, we do **not** mean to suggest that a system should engage in extended contemplation of every one of its computational and physical actions, but rather= (a) that it ***can*** reason about many of its actions: **(b)** that it does reason about them much of the time; and (c) that its reasoning improves its "efforts to achieve goals and otherwise function in its environment.

A system might reason about its actions in various ways and with various consequences (see Figure la) . For example, a system might ***control*** its actions: decide which actions to perform at particular points in time. Control reasoning can affect the resources the system **consumes** in pursuing a goat. the side effects it produces, and the probability of achieving its goal **[8,** 9, 13, 17, 23, 26, **27].** As a second example, a system might ***explain*** its actions: describe the ways in which the actions it intends to perform or has performed serve its goals. Explanation typically serves social functions, such as teaching another individual how to perform a **task** or persuading another individual that one is performing the **task** competently **[5,** 6, 21, 22). As a third example, a system might ***learn*** about its actions: modify its **ability** or inclination to perform particular actions in appropriate circumstances. Learning **enables** the system to expand and improve its capabilities **[24,** 32, 33, 35, 38, 39, **45].** While a system could perform many other important types of reasoning about its actions, we focus on control, explanation, and learning.

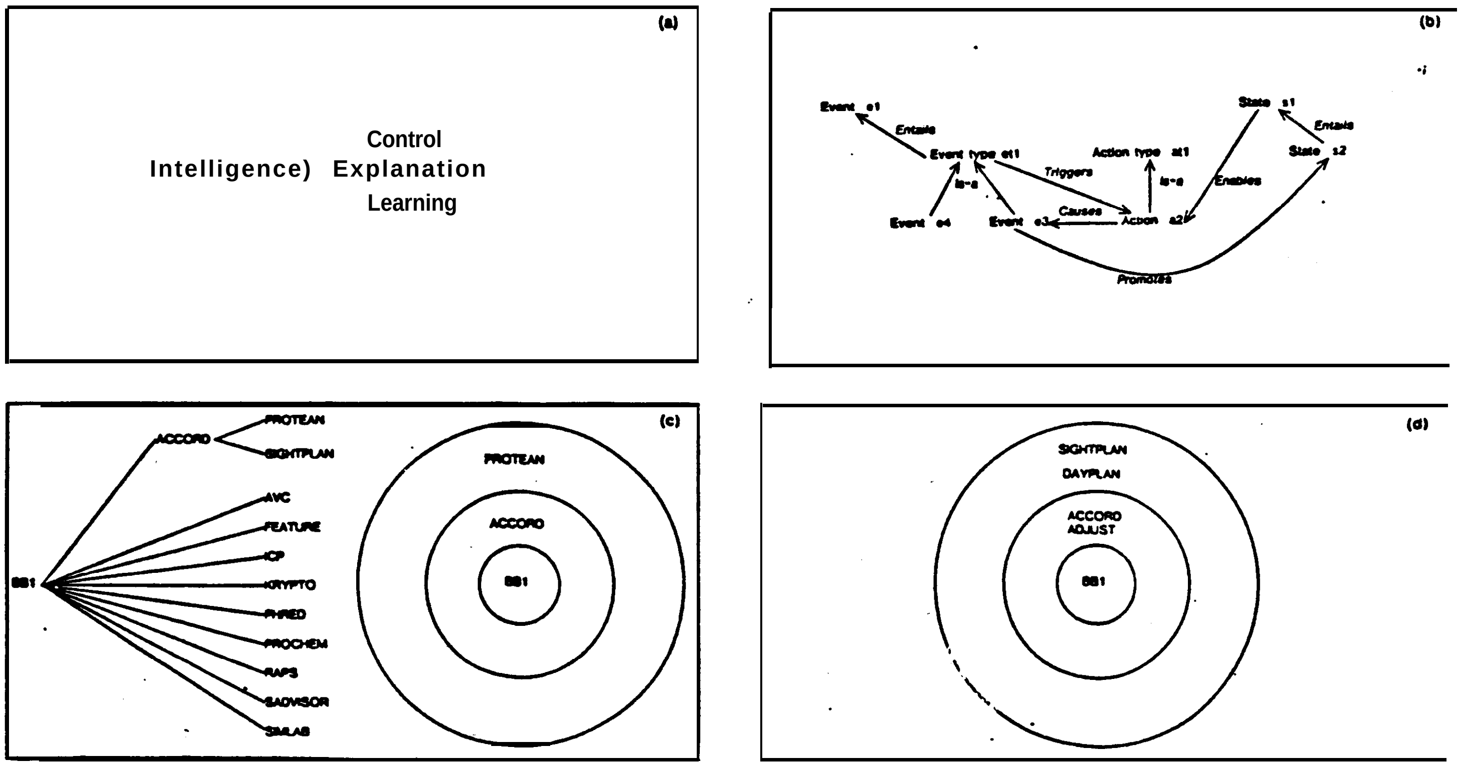


Figure L Four **Themes.** *(a) An intelligent* system *reasons about its actions. The* **BB1** architecture provides knowledge structures and a basic mechanism for control. explanation, and **laming (b) To perform** effectively, a ***system must*** *hare knowledge* ***about its*** *actions.* Frameworks explicitly represent knowledge about task-specific actions, **events, and states** and the relationships among them. *(c) Knowledge is* ***represented*** In *an* ***abstraction*** *hierarchy. The* **BB*** environment comprises an evolving body of **knowledge:** the **BB1** architecture, task-specific frameworks, such as ACCORD, and domain-specific applications, such as PROTEAN (see Table 1). Conversely. an application system **layers** application-specific knowledge on a framework, which **layers** task-specific knowledge on the **BB1** architecture. (d) *Knowledge modules within a* ***level satisfy*** *uniform rrandards of knowledge* ***content*** *and representation. As* a consequence. **BB* achieves** open systems integration: Independently constructed modules can be fully integrated in implementation and reasoning.

Given the premise above, we put forth a hypothesis: In order ***to perform effectively, an intelligent system must have knowledge of its actions.*** It must have knowledge of the actions it can perform, of the events and states that can occur, and of the relationships among particular instances of these actions, events, and states. For example, it must know: the actions that are relevant to its current task; the enabling conditions required by particular actions; the cost, reliability, and side effects of particular actions; the internal and external events and states whose occurrences contribute to or hinder performance of its task; the power of particular actions to bring about particular events and states: and the power of external forces to bringabout particular events and states.

In our work, we formulate explicit, interpretable representations of these and other kinds of knowledge (see Figure lb) as a foundation for intelligent behavior. Thus, we define "knowledge" broadly, as "that which is known."[2] In fact, most computational objects in our systems (all except the basic architectural cycle, low-level data-retrieval functions, and user interface) appear as elements of a well-structured, modular, declarative knowledge base. As such, they are amenable to knowledge-level operations, such as acquistion, modification, verification, deduction, induction, instantiation, and comparison. Moreover, we can incrementally improve almost any aspect of a system's' behavior by extending the depth or extent of its knowledge. We have begun to construct an expanding edifice of such knowledge for a variety of ***problem classes, problem-solving methods,*** and ***subject-matter domains.***

In constructing this edifice, we emphasize a design principle: We ***represent knowledge in an abstraction hierarchy.*** Although "true" knowledge abstractions probably lie on a continuum, we currently focus on three particular levels--architecture, framework, and application.

At the most general level, we define an ***architecture*** to comprise: (a) the set of basic knowledge structures used to represent all actions, events, states, and facts in a system; and (b) a mechanism for instantiating, choosing, and executing actions. Architectural knowledge is independent of problem class, problem-solving method, and subject-matter domain. For example, the ***blackboard control architecture*** [23], which is implemented as the BB1 system discussed below, supports applications as varied as protein-structure analysis [4, 25, 29], process planning [41], and autonomous vehicle control [43]. In addition, BB1 provides specific knowledge structures and a powerful mechanism to support intelligent. control, explanation, learning.

[2]The American Heritage Dictionary of the English Language, 1981, "Knowledge," **definition # 3.**

At the intermediate level, we define ***a framework*** as the set of knowledge structures used to represent actions, events, states, and facts involved in performing a particular ***task.*** That is, a framework comprises the knowledge structures involved in solving a particular class of problems with a particular method, but independent of subject-matter domain. For example, ***the arrangement-assembly framework,*** which is implemented as the ACCORD knowledge base discussed below, embodies the knowledge used to solve ***arrangement problems*** by means of an ***assembly method.*** However, the knowledge in ACCORD applies to arrangement-assembly tasks in such varied subject-matter domains as protein-structure analysis, construction-site layout, and travel planning.

At the most specific level, we define an ***application*** as the set of knowledge structures that instantiate particular actions, events, states, and facts to solve a particular class of problems by means of a particular method in a particular subject-matter domain. For example, the PROTEAN system [4, 25, 29] embodies the knowledge used to determine the three-dimensional structures of proteins--that is, to solve arrangement problems in the domain of protein chemistry by means of the assembly method.

As illustrated in Figure lc (see also Table 1), BB1, ACCORD, and PROTEAN are elements of a knowledge abstraction hierarchy. BB1 can accommodate a variety of modular frameworks, one of which is ACCORD. Similarly, ACCORD (and each other framework) can accommodate a range of modular applications, one of which is PROTEAN. (As Figure lc shows, many current applications are implemented directly in BB1.) We refer to the evolving set of such modules *as the BB* environment.*

Table 1. Some Current Elements of BB*

Architecture	Description
BB1[23]	Blackboard-based problem solving architecture
Framework	**Description**
ACCORD	Solves arrangement problems using the assembly method
Application	**Description**
AVC[43]	Plans missions for automomous vehicles
FEATURE[3]	Explores protein structures for interesting features
ICP[34]	Dynamically plans curricula for an Intelligent tutoring system
KRYPTO[28]	Solves constraint-satisfaction problems
PHRED[41]	Plans the construction process for aircraft components
PROCHEM[11]	Models protein structure based on theoretical constraints
PROTEAN[4,25,29]	Assembles protein structure based on empirical constraints
RAPS[30]	Diagnoses electro-mechanical systems
SADVISOR[10]	Advises on space station safety
SIGHTPLAN[50]	Designs construction site layouts
SIMLAB[40]	Schedules personnel. hardware and software for flight simulation

Conversely, a given application system composes modules from the BB* environment in several layers of implementation (see Figure lc). For example, PROTEAN's knowledge about constructing proteins instantiates and configures a number of ACCORD's more general knowledge structures for assembling arrangements. Similarly, ACCORD's knowledge structures instantiate and configure a number of BB1's still more general knowledge structures about problem-solving, control, explanation, and learning. When PROTEAN goes to work on a problem, its actions are interpreted through these several layers of implementation.

In adapting this widely accepted software engineering principle-generally referred to as ***modular and layered design*** [18, 19, 50]--to intelligent systems, we achieve several advantages. First, each abstraction level offers certain representational and computational services to higher levels, while shielding them from the details of implementation. Second, we can understand complex systems in terms of their simpler modular components. Third, we can investigate and test alternative implementations of modules at one level independently of the modules at other levels. Fourth, we can eliminate levels from applications that do not require their services. Fifth, we can achieve additive and, in some cases, multiplicative improvements in efficiency across levels [44]. Finally, we can apply general knowledge modules in an appropriate variety of contexts and configure selected lower-level knowledge modules for a variety of specific purposes.

We impose one additional constraint on our knowledge abstraction hierarchy: ***Modules within a level must meet uniform standards of knowledge content and representation.*** Accordingly, we adopt a single architecture, BB1. Although BB1 accommodates multiple frameworks, each of them must provide the same core categories of knowledge within a specified representation scheme. Similarly, each application must provide another set of core knowledge categories within another specified representation scheme.

This constraint offers several related advantages. First, we can define new application systems by configuring and augmenting existing knowledge modules within a level. Second, we can identify and eliminate redundancy in the contents of independently acquired modules within an application system. Third, we can organize modules in any appropriate organizational-scheme. In particular, we can organize them in a conventional "pipeline," such that a succession of modules receive, process, and pass on information. Alternatively, we can organize them to operate more intimately: operating simultaneously, sharing intermediate results, and affecting one another's behavior. In fact, a system can reason about how to select and organize modules to solve new problems. Fourth, we can superimpose generic capabilities for control, explanation, and learning upon the designated configurations of modules. In sum, uniformity of content and representation within a level allows us to achieve the conventional capabilitiy of open systems ***interconnection*** [53] and to strive toward a more ambitious capability that we will call ***open systems integration.*** It raises the possibility of incrementally increasing the quantity and variety of knowledge within an application system, while preserving a well-structured foundation and a coherent face for the system as a whole (see Figure Id).

Our objectives in this work are two-fold, First, we wish to develop a rich and varied family of reusable modules for building intelligent systems. System builders should be able to build new systems by configuring appropriate subsets of these modules in appropriate organizational schemes. Where new modules are needed, system builders should be able to introduce them into the existing family and integrate them into new systems with ease. The resulting systems should be well-structured, perspicuous, modifiable, and extensible. Second, we wish to develop a theory of intelligent systems. The theory must provide: (a) a great range of problem-solving skills, including the ability to solve a variety of problem classes with a variety of problem-solving methods in a variety of subject-matter domains; (b) the ability to apply any available knowledge to improve problem-solving performance: and (c) the ability to reason about--control, explain, and learn about--action. We believe that our approach to developing the BB* environment enables us to progress toward both objectives.

The remainder of this paper develops and substantiates the four themes introduced above and displayed in Figure 1 as follows. Section 2 briefly reviews the BB1 blackboard control architecture and its capabilities for control, explanation, and learning. Section 3 defines the arrangement-assembly task, using PROTEAN as an illustration. Section 4 presents the arrangement-assembly framework, "and its implementation as the ACCORD knowledge base. Section 5 describes the BB1 *framework-interpreter,* which allows BB1 to accommodate any framework that meets the standards of knowledge content and representation illustrated by ACCORD. Section 6 describes the layered architecture of PROTEAN and illustrates control, explanation, and learning. within a BB* application system. Section 7 discusses knowledge engineering within the BB* environment. It describes the design and implementation of another arrangement-assembly system (the SIGHTPLAN system [51] for designing construction-site layouts) and examines the applicability of BB1-ACCORD to arrangement-assembly tasks in other domains. Section 8 introduces a new class of *multi-faceted* systems to illustrate BB*'s capability for open systems integration. Section 9 discusses the current state of the BB* environment and our plans for extending it. Section 10 highlights the major results of the paper.

2. BB1: An Architecture for Control, Explanation, and Learning

2.1 Overview of BB1

BB1 provides a uniform blackboard architecture for systems that reason about their own actions as well as about particular problems and solutions. In a BB1 system, functionally independent ***knowledge sources*** cooperate to solve problems by recording and modifying solution elements in a global data structure called the ***blackboard.*** A system may have three classes of knowledge ***sources. Domain knowledge sources*** solve domain problems on a ***domain blackboard*** and send and receive messages along input/output channels. ***Control knowledge sources*** construct control plans for the system's own behavior on a ***control blackboard. Learning knowledge sources*** modify knowledge sources and facts in the system's ***knowledge base.*** All knowledge sources operate simultaneously and, when triggered; compete for scheduling priority. BB1 also provides an explanation capability by which a system shows how its actions fit into its control plan. Figure 2 illustrates the BB1 execution cycle.

Since we have discussed BB1's knowledge structures and procedures in detail elsewhere [23], we do not repeat that material here. Instead, we briefly characterize BB1's capabilities for 'control, explanation, and learning.

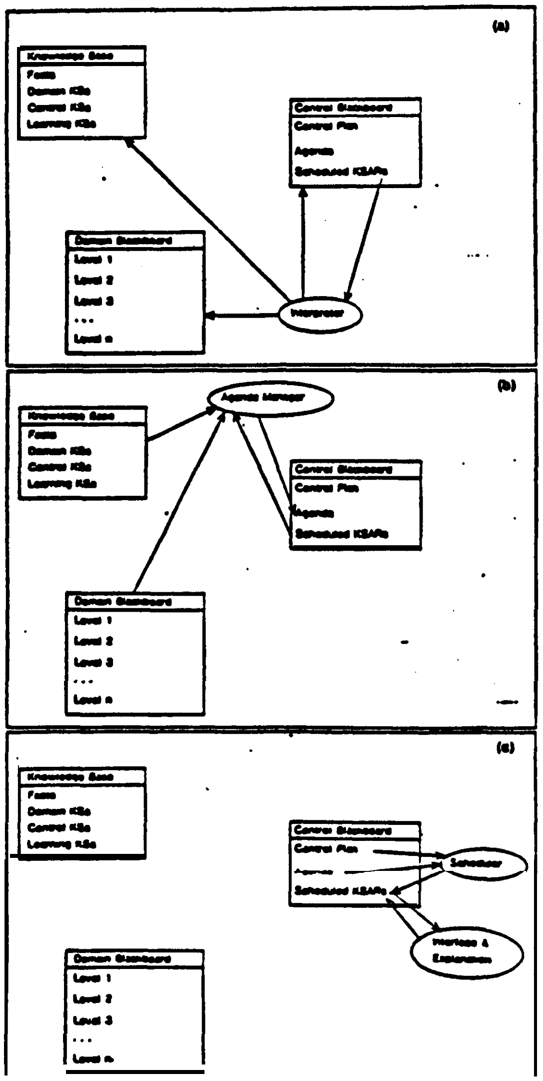


Figure 2. The **BB1** Blackboard **Control Architecture.** **The BB1** execution cycle comprises **three** steps (a) **The *interpreter* executes** the action **Of** one knowledge source. Depending upon whether ***the knowledge source is a domain. control. or learning knowledge source, its action*** changes the contents of the domain or control blackboard or the knowledge base. (b) The blackboard changes satisfy the conditions of ***other*** domain, control, and learning knowledge **sources.** ***The agenda-manager*** ad& corresponding ***KSARs (knowledge source activation records) to the agenda.*** (c) The ***scheduler*** rates each **KSAR** on the agenda against the current **control** plan and, using ***a scheduling rule*** that is **recorded** on the control blackboard. chooses one **KSAR** to execute **its** action. Unless it has been instructed to operate autonomously, the scheduler also invites the user to request **an** explanation **for** the chosen action. **to request** any of several other kinds of informarion. or **to override** the scheduler's chosen action with another **one.**

2.2 Control Reasoning in BB1

BB1 provides a framework in which control knowledge sources incrementally construct an explicit control plan, for the system's own actions, on the control blackboard. Decisions at high levels of abstraction prescribe general classes of actions to be performed during relatively long problem-solving time intervals, while decisions at low levels prescribe more specific classes of actions to be performed during relatively short time intervals. Thus, BB1 supports a kind of hierarchical planning [16, 15, 46, 37] with several important differences.

First, hierarchical planning systems typically refine selected plans to sequences of specific actions to be performed on specified sequences of problem-solving cycles. By contrast, a BB1 system can refine selected plans to any desired level of specificity. For example, a system might refine its plan to a sequence of action classes, where each class is characterized by a set of desirable attribute-value relations. It would perform the "best" actions in each class during an open-ended problem-solving time interval that begins when a specified control state occurs and terminates when a specified solution state is achieved

Second, hierarchical planning systems typically formulate complete plans prior to beginning plan execution. By contrast, a BB1 system can--and generally does--construct its plan incrementally during plan execution, taking account of the results of previously executed actions in its reasoning about subsequent plan elements. For example, a system ordinarily would not generate its second planned action class until after it had achieved the goal of the first action class. It might use solution elements established by actions in the first class to determine some of the desirable attribute-value relations in the second class.

Third, hierarchical planning. systems typically formulate a single, integrated plan for the problem at hand. By contrast, a BB1 system can formulate multiple plans, of idiosyncratic hierarchical depths, for overlapping aspects of the problem and pursue them simultaneously. For example, a system might adopt and begin pursuing a comprehensive plan for the entire problem at hand. At some point during its problem solving, the system might notice an infrequent, but significant intermediate solution state. It might formulate a local plan that specifically addresses that solution state and pursue it concurrently with its larger plan.

Fourth, since BB1 generates its control plan incrementally and explicitly represents the evolving control plan on the control blackboard, a system can interrupt, depart from, modify, discard, or resume construction and execution of a plan in response to the dynamic situation. For example, a system could begin implementing a comprehensive strategy for the problem at hand, but subsequently determine that it had chosen a suboptimal initial value for a *key* strategic parameter. **A** control knowledge source triggered by this observation could "back up" the system's control reasoning, add a new heuristic to exclude' the originally chosen value, and then allow the system to resume its problem solving activities in accordance with the modified control plan.

Fifth, in addition to the top-down inference method underlying **skeletal** planning, a **BB1** system can incorporate a variety **of** other inference methods, such **as:** (a) bottom-up methods that hypothesize the desirability of pending actions not explicitly favored by the current control plan: (b) goal-directed methods that plan actions whose results would trigger actions **favored** by the **current** control plan: and (c) opportunistic methods that plan actions. whose **results** would improve **a** targetted aspect *of* the **current** solution.

Finally, a **BB1** system integrates reasoning about **control** of all domain and control actions within a uniform blackboard architecture. Thus, for example, **a** system might record and concurrently apply heuristics favoring control actions over domain actions along with its strategic heuristics favoring particular kinds of domain actions.

2.3 Explanation in **BB1**

23.1 Overview of Explanation

BB1's explicit representation of a system's control plan provides a database for use in explaining a system's actions. Drawing upon this information, a system can explain what makes particular actions feasible and how alternative actions serve its current control plan. It also can explain the internal structure and rationale for its control plan.

BB1 currently provides a graphics-based, menu-driven explanation capability. Different menu options allow the user to request explanations that highlight different aspects of the current control plan. For example, the option focal context explains an action's immediate superordinate in the control plan and its preceding siblings. By contrast, the option complete picture explains the entire control plan and all previously performed actions leading up **to** the decision to perform an action. These and other explanation options are described in more detail in **[47].**

2.4 Learning in BB1

BB1 structures the data needed to learn new control strategies. Learning knowledge sources can observe relationships between KSARs, the events that trigger them, and the events that they produce. They can observe similarities and differences among competing KSARs and determine how those KSARs rate against the current control plan. They can exploit BB1 data structures to program new control knowledge sources.

For example, a generic learning knowledge source called MARCK [24] learns a new control heuristic whenever a domain expert corrects an application system's scheduling decision. MARCK hypothesizes that the expert is using a control heuristic that distinguishes the action he or she wishes to perform from the one the application system scheduled. MARCK compares the two actions, identifies the key difference between them, and formulates a control heuristic favoring the attribute preferred by the domain expert MARCK immediate posts the new heuristic on the control blackboard, but also programs a new control knowledge to post that heuristic in future problem-solving episodes.

We are working on another set of learning knowledge sources called WATCH [20]. These knowledge sources observe a domain expert scheduling a system's problem-solving actions and recursively abstract a hierarchy of' control heuristics that capture sequential regularities in the expert's scheduling decisions. Then they automatically program new control knowledge sources that post and expand the hierarchy top-down during subsequent problem-solving episodes.

3. The Arrangement-Assembly Task

As discussed in section 1, a framework, such as ACCORD, is more specific than an architecture, such as BB1, because it defines the actions, events, states, and facts involved in solving a particular class of problems by means of a particular method. However, a framework remains independent of subject-matter domain. In this section, we discuss an illustrative task-- *the arrangement-assembly task* and illustrate it with PROTEAN's protein-modeling task In section 4, we discuss the framework we have developed for the arrangement-assembly task and its implementation as the ACCORD knowledge base.

3.1 Arrangement Problems .

We define a ***problem class*** by its characteristic inputs and outputs. ***Arrangement problems*** provide these inputs: a set of symbolic objects, a context, and a set of constraints. They require as output: one or more arrangement(s) of the objects in the context such that' each arrangement satisfies the constraints. Arrangement problems arise in a variety of domains, such as furniture arrangement, page layout, travel planning, and task scheduling. For illustration purposes, we focus on an arrangement problem attacked by the PROTEAN system.[3]

PROTEAN must identify the three-dimensional conformations of proteins. Its input data specify a test protein's ***primary*** and ***secondary*** structures (see Figure 3) and the atomic architecture of each individual ***amino acid*** (see Figure 4). Its input data also specify a number of constraints (see Table 2). For example, there may be about SO-60 ***NOEs (Nuclear Ovcrhauser Effects),*** each of which indicates that two particular atoms in the protein are within 3-10 angstroms of one another. There may be evidence that certain atoms are accessible to solvent, indicating that they lie near the molecular surface of the protein. There may be information about the overall size, shape, and density of the protein molecule.

[3]The PROTEAN project is directed by Bruce Buchanan and Oleg Jardetzky. The research team includes: Olivier Lichtarge, Barbara Hayes-Roth. Bruce Duncan, Russ Altman, Jim Brinkley, Craig Cornelius. Alan Garvey, and John Brugge.

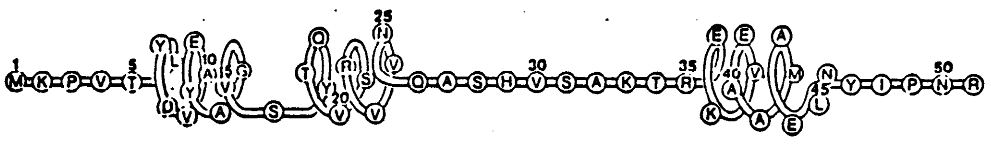


Figure 3. Primary and Secondary Structure of the Lac-Repressor Headpiece The lac-repressor's *primary structure* is a unique sequence of 51 amino acids, each of which is one of the 20 unique amino acids. Its *secondary structure* includes three *alpha-helices*, each of which is defined by a series of repeated angular turns in the protein's backbone. Interspersed among its helices, the k-repressor headpiece has random *coils.* segments of the primary structure that show no identifiable regularity.

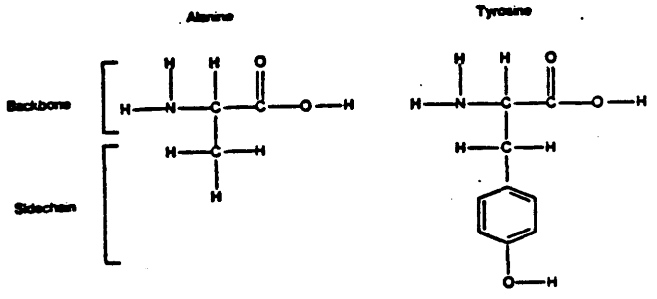


Figure 4. Two Amino Acids: Alanine and Tyrosine. As these examples illustrate, each amino acid has a common part, at which it bonds to neighboring amino acids to form the backbone of a protein, and a unique sidechain that distinguirhea it from other amino acids.

Table 2. Some of the Constraints Available to PROTEAN

Primary structure
Atomic structure of individual amino acids
Van der Waals' radii of individual atoms
Peptide bond geometry
Secondary structure
Architectures of alpha-helices and beta-sheets
Molecular size
Molecular shape
Molecular density
NOE measurements
Surface data

Based on these input data, PROTEAN must identify the test protein's *tertiary structure*--the folding of its primary and secondary structures in three-dimensional space (see Figure 5). Because the problem is underconstrained, there may be many conformations that satisfy the available constraints. PROTEAN must identify the entire family of such conformations. Moreover, since proteins are known to be mobile in solution, PROTEAN must characterize potential mobility in the conformations it identifies.

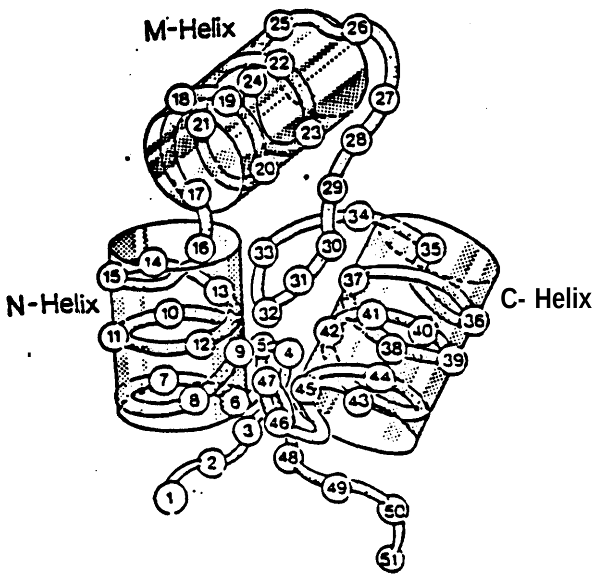


Figure 5. The Tertiary Structure of the Lac-Repressor Headpiece. The lac-repressor's *tertiary structure* defines the folding of its primary and secondary structures in three-dimensional space to pack all component structures into a globular molecule.

3.2 The Assembly Method

We define a ***problem-solving method*** in terms of the knowledge a problem solver uses and the operations it performs in order to solve a particular problem. In principle, a problem solver could use any of several different methods to solve an arrangement problem (see Table 3). In practice, however, the problem solver may not have the knowledge necessary to apply a given method. For example, PROTEAN cannot apply the ***selection, refinement, modification,*** or ***generation*** methods because it does not have knowledge of alternative protein structures, a prototypical protein structure, almost-correct protein structures, or an algorithm for generating complete protein structures. In the absence of such knowledge, a problem solver must ***construct*** hypothetical arrangements. The ***assembly method*** is one method for constructing arrangements. Unlike the other methods in Table 3, the assembly method can be applied to any arrangement problem.

Table 3. Methods for solving Arrangement Problems.

1. Select an arrangement that satlsfles the constraints from a pre-enumerated set of alternatives.
   Requires Knowledge of: Alternative arrangements.
   Example: A travel agent selects one of several tour "packages" that includes all of the destinations requested by a client.

2 Refine a prototypical arrangement so as to satisfy the constraints.
   Requires Knowledge of: A prototypical arrangement.
   Example: An architect refines a prototypical U-shaped kitchen design to include the special appliances requested by a client.

3. Modify an almost-correct arrangement to satisfy the constraints.
   Requires Knowledge of: Almost-correct arrangements.
   Example: A tool designer modifies an existing tool to fit a new machine.

4. Generate a complete arrangement that satisfies the constraints.
   Requires Knowledge of: A procedure for generating complete arrangements.
   Example: A psychologist uses a multi-dimensronal-scaling algorithm to generate a spatlal model of subjects' similarity ratings of related concepts.

5. Construct an arrangement that satisfies the constraints.
   Requires Knowledge of: A method for constructing arrangements.
   Example: A person solves a jigsaw puzzle by placing pieces one at a time.

The basic assembly operation applies one or more constraints to determine where in the specified context a particular object can lie given: (a) its current hypothesized position; (b) its constraints with other objects or with contextual ‘features; and (c) the current hypothesized position of those other objects or features. In performing this operation, the problem solver must exploit some application-specific procedure for generating legal positions. For example, PROTEAN currently uses a generate-and-test procedure [3], sampling space at some level of resolution and identifying all locations in which a structure satisfies a given set of constraints. Figure 6 illustrates PROTEAN’s application of constraints.

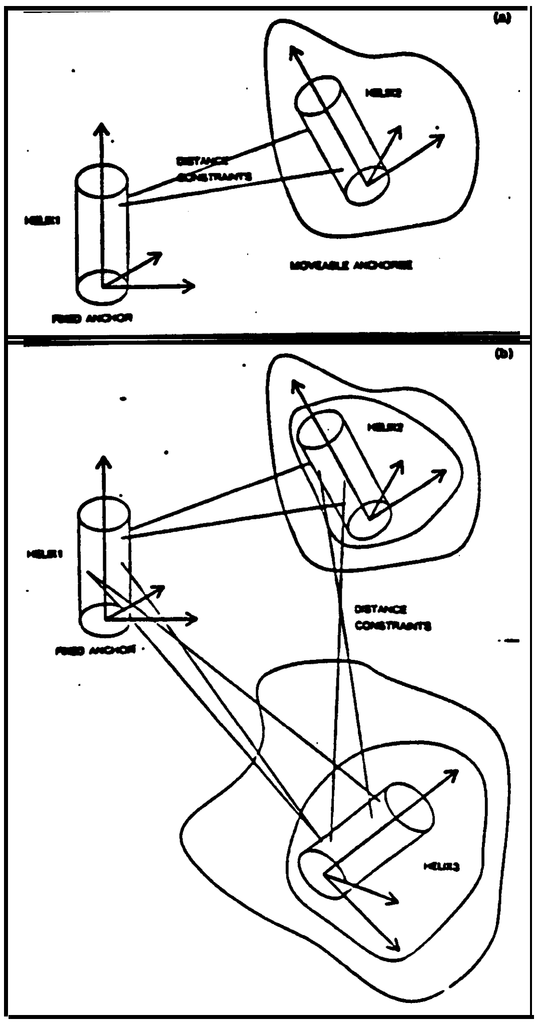


Figure 6. Constraint Application in PROTEAN. (a) PROTEAN **assumes** a fixed position for **helix1** and ***anchors*** **helix2**, determining that helix2 can lie in any **location** within the outlined region and still satisfy its constraints with **helix1. (b)** PROTEAN ***yokes*** helix2 and **helix3, pruning** the locations previously identified for **these helices** to include only those that satisfy **constraints** between them.

The problem solver can perform its positioning operations in the context of one or more partial arrangements (see Figure 7). Each partial arrangement includes a subset of the objects and constraints specified in the problem. The problem solver designates one object in a partial arrangement to occupy a fixed location and positions all other included objects relative to it. Eventually, the problem solver combines two or more partial arrangements to form a complete arrangement.

The problem solver may assemble partial arrangements at different levels of abstraction, where objects at each level aggregate sets of constituent objects at the next lower level (see Figure 8). The problem solver can use the positions of abstract objects to restrict the number of possible locations for their constituent objects. Conversely, it can use the positions of constituent objects to restrict the locations hypothesized for their superordinate objects.

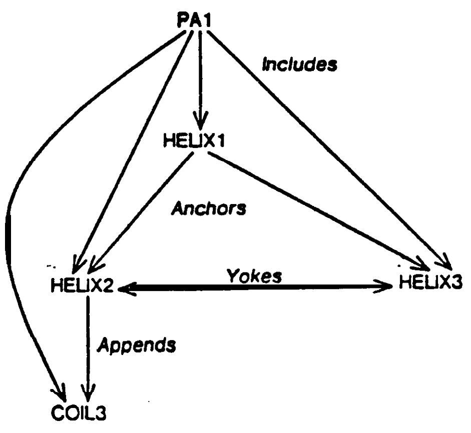


Figure 7. A Partial Solution for the Lac-repressor Headpiece. Pal ***includes*** **helix1, helix2, helix3,** and **coil3. Helix1,** which has been defined as the anchor of pal, ***anchors*** helix2 and **helix3.** Helix2 ***appends*** **coil3,** which has no **constraints** with the anchor. Helix2 and helix3 yoke one another with the constraints between them.

Because the assembly method searches a combinatoric space of possible arrangements, the problem solver must control its search intelligently. It must reason about: how to group objects in partial arrangements: which object should define the local coordinate space of each partial arrangement; when to position particular objects with particular constraints; when to work at particular levels of abstraction: and when to combine partial arrangements. This reasoning must incorporate general computationat principles, such as: defining the local coordinate space about an object that has many constraints to many other objects; focusing on objects that already have been restricted to relatively specific positions: and preferring constraints that maximally restrict an object's position. It must also incorporate domain-specific knowledge. For example, PROTEAN's reasoning incorporates biochemistry knowledge such as: defining the space of potentially useful constraints: and characterizing the constraining power of different constraints. .

Similarly, an intelligent problem solver should be able to explain its assembly actions and learn new assembly strategies from experience.

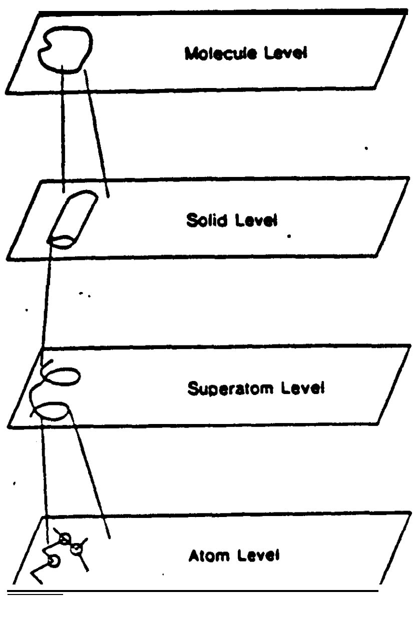


Figure 8. PROTEAN's Levels of Reasoning. At the *molecule* level. PROTEAN reasons about the size, shape, and density of the protein molecule. At the *solid level,* it reasons about the relative positions of the test protein's secondary structures, represented as geometric solids. At the *superatom* level, it reasons about the positions of each amino acid's constituent peptide unit and sidechain. At the *atom* level, it reasons about the positions of individual atoms.

4. ACCORD: Knowledge about Assembling Arrangements

The ACCORD knowledge base provides an explicit, interpretable representation of the knowledge required, to assemble arrangements and to control, explain, and learn about arrangement-assembly actions. The elements of ACCORD include: (a) a conceptual network that organizes all arrangement-assembly knowledge; (b) a type hierarchy of domain entities; (c) a type hierarchy of arrangement roles; (d) type hierarchies of assembly actions, events, and states: (e) networks of characteristic relations among assembly actions, events, and states; (f) linguistic templates for instantiating assembiy actions, events, and states: (g) the partial matches among these templates; and (h) translations of arrangement-assembly templates into corresponding templates in a lower-level language. The following sections describe these elements and, where necessary, illustrate them with the domain knowledge of PROTEAN.

4.1 The Conceptual Network

We represent all of the knowledge in ACCORD within a conceptual' network [49].

The network distinguishes three kinds of concepts (see Figure 9): types, individuals, and instances. Concept types intensionally define the generic concepts of a **task** by means of ***is-a*** links. These include ***domain entities*** (e.g., helix is-a secondary-structure), ***arrangement roles*** (e.g., anchor is-a arrangement-role), and ***assembly actions, events,*** and ***states*** (e.g., position is-a assembly-action). Concept ***individuals exemplify*** particular concept types (e.g., helix1 the first helix in the primary sequence of the lac-repressor headpiece. exemplifies helix). Concept ***instances instantiate*** individuals in particular contexts (e.g., helixl-1, instantiates helixl in the context of partial arrangement pal). Concept instances also ***play*** particular roles in those contexts (e.g., helixl-1 plays anchor).

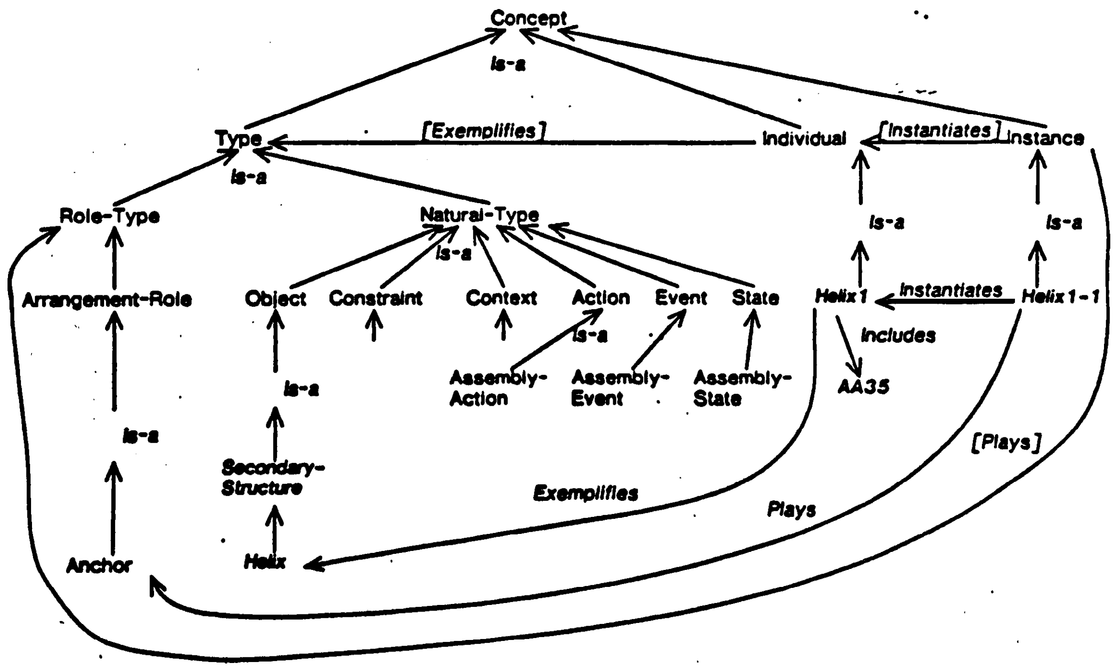


Figure 9. Schematic **Overview of ACCORD's Conceptual Network. Concept *types* intensionally** define generic **concepts** by **means** of ***is-a*** links. These include natural **types (e.g., helix, assembly-action) and role typa (e.g., anchor). Concept *individuals*** (e.g. (helixl) ● **xrmpl/fi particular concept types.** Concept ***instances* (e.g., helix1-1) *instantiate*** particular **individuals** to play particular roles in particular **contexts.** Concepts attributes can **have static or procedural values. Both** attributes and link relations are inheritable. Bracketed links in this and other **figures** indicate legal l inks **(e.g.,a** concept individual may exemplify **a** concept type), while unbracketed **links** indicate **actual** links (e.g., **the** individua! helix1 actually does exemplify the type helix). PROTEAN-specific concepts in this figure appear in bold type.

Concepts may have other links. For example, one concept may ***include*** several constituents. In addition, all links in the network have corresponding inverse links: ***can-be-a, is-exemplified-by, is-instantiated-by, is-played-by, is-included-by.*** Finally, implicit ***$link*** relations hold between concepts related by chains of specific component links. A ***$is-a*** relation holds between any two concepts related by a chain of instantiates, exemplified, and is-a links. For example, we may infer that:

```
        Helix1-1 $is-a secondary-structure.
```

because:

```
     Helix1-1 instantiates helixl.
        Helix1 exemplifies helix.
        Helix is-a Secondary-Structure.
```

A ***$includes*** relation holds between concepts related by a chain of instantiates, exemplifies, is-a, and' includes links. For example, we may infer that:

```
        Hel ix1-1 $includes Amino-Acid35.
```

because:

```
        Helix1-1 instantiates helixl.
        Helix1 includes Amino-Acid35.
```

A ***$plays*** link holds between concepts related by a chain of exemplified-by, instantiated-by, and plays links. For example, we may infer that:

```
        Helix Splays anchor..
```

because:

```
        Helix is-exemplified-by helixl.
        Helix1 is-instantiated-by helix1-1.
        Hel ix1-1 plays anchor.
```

These and all other ***$<link>*** relations 'have corresponding inverse relations that hold between corresponding chains of inverse component relations. For example, we may infer that:

```
        Anchor $is-played-by helix.
```

because:

```
        Anchor is-played-by helix1-1
        helix1-1 instantiates helixl.
        helix1 exemplifies helix.
```

Any concept in the network may specify particular attributes, along with static or procedural values. For example, PROTEAN's concept network includes the facts that: helix has an attribute called ***shape.*** whose value is cylinder: and secondary-structure has an attribute called ***length.*** whose value is determined by a procedure called Number-of-AA that counts the number of amino-acids included by the secondary-structure. Like relations among concepts, *these* attributes are inheritable. For example, helixl-l's shape is cylinder and its length is determined by the procedure Number-of-AA.

One class of attributes warrants special mention. ***Modifiers*** are attributes whose procedural attachments evaluate the applicability of the named descriptors to any given concept individuals or instances. For example, PROTEAN's modifier *tong* is an attribute of the concept type secondary-structure. Its value, which is computed by the procedure called How-Long-Is, is a function of the number of amino-acids included by a particular secondary-structure (that is, by. a particular alpha-helix, beta-sheet, or random-coil). All such procedures return numerical values scaled O-100, where 0 signifies minimal applicability of the modifier and 100 signifies maximal applicability. However, a framework can distinguish two different procedural definitions for each modifier.

Threshold procedures evaluate concepts in an all-or-none fashion. For example, PROTEAN might refer to a "long helix," meaning "a helix that has at least 15 amino acids." An individual helix, say helixl, either matches this description or it does not Therefore, the threshold procedure attached to the attribute long returns a value of 100 for any helix that includes more than 15 amino acids and a value of 0 for any helix that includes fewer than 15 amino acids. In general, threshold procedures return a value of 100 or 0, depending upon whether or not the modified concept exceeds a designated threshold on a designated attribute.

. ***Scale procedures*** evaluate concepts in a graded fashion. For example, PROTEAN might refer to a "long helix," meaning "a helix that includes at least 15 amino acids is better than one that includes 10-14 amino acids, which is better than one that includes fewer than 10 amino acids." An individual helix, say helixl, matches this description to some degree. Therefore, the scale procedure attached to the attribute long returns a value of 100 for any helix that includes more than 15 amino acids, a value of 50 for any that includes 10-14 amino acids, and a value of 0 for any helix that includes fewer than 10 amino acids. In general, scale modifiers return values somewhere in the range O-100, depending upon the degree to which the modified concept exhibits a designated attribute.

Threshold or scale procedures may be specified within an expression by extending the modifier name with "-T" or "-S." However, as discussed below, BB1 knows in which circumstances each type of procedure typically applies. If no extension appears in a modifer, it uses the appropriate procedure.

4.2 Types of Domain Entities

A framework provides skeletal branches of the natural-type hierarchy in which to define relevant domain entities.

For the arrangement-assembly task, ACCORD provides skeletal branches for. the ***objects*** to be arranged, the ***context*** in which the objects must be arranged, and the ***constraints*** that must be satisfied within the arrangement. Particular constraints may ***involve*** particular objects and constraints. Figure 10 illustrates how PROTEAN instantiates these skeletal branches with biochemistry entities. In addition. PROTEAN specifies the characteristic attributes of and relations among entities. For example, it specifies that alpha-helix, beta-sheet and random coil have the attribute ***shape*, with** the values cylinder, prism, and sphere, respectively.

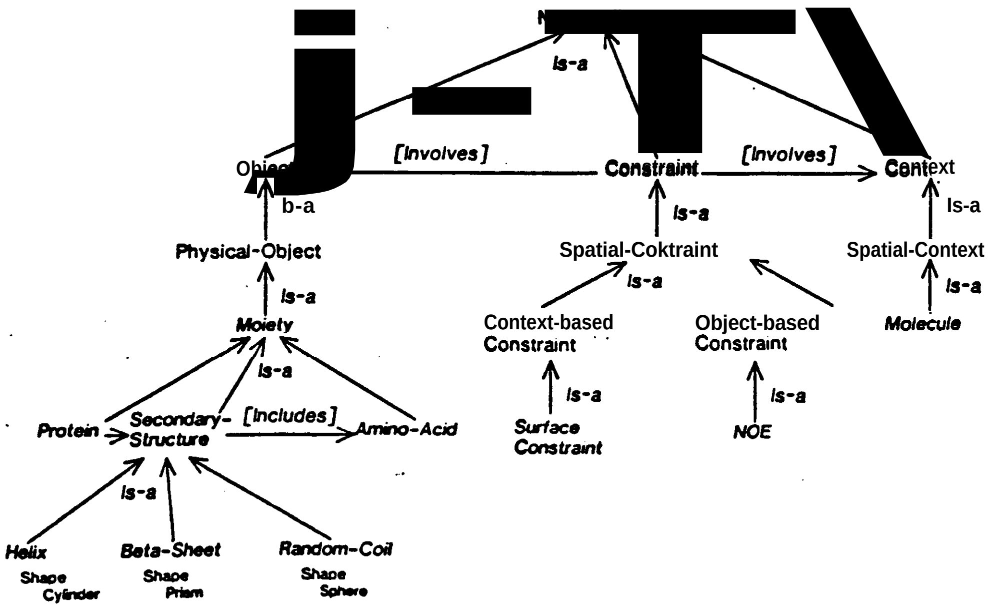


Figure 10. ACCORD's Skeletal Branches for Objects. Contexts. and Constraints. PROTEAN-specific entitia appear in bold type. Individual constraints can involve particular objects or contexts.

4.3 Role Types

A framework defines the roles that problem entities can play in hypothetical solutions.

ACCORD defines the arrangement roles illustrated in Figure 11. An ***arrangement*** is a potential complete solution to an arrangement problem, that comprises one or more partial-arrangements that, together, comprise a criteria1 subset of its objects, constraints, and context. A ***partial-arrangement*** **is a partial** solution to a problem, that comprises a **non-criterial** subset of **its** objects, constraints, and context An ***included-object*** is one of the objects from the problem that has been selected for inclusion in a partial-arrangement Included-object has three subordinate subtypes. An ***anchor*** is an included-object that has been assigned a fixed location **to** define ***the local context*** of a partial arrangement An ***anchoree*** is an included-objects that **has** at least one constraint with the anchor. An ***appendage*** is an included-objects that has at least' one constraint with at least one **anchoree.**[4]

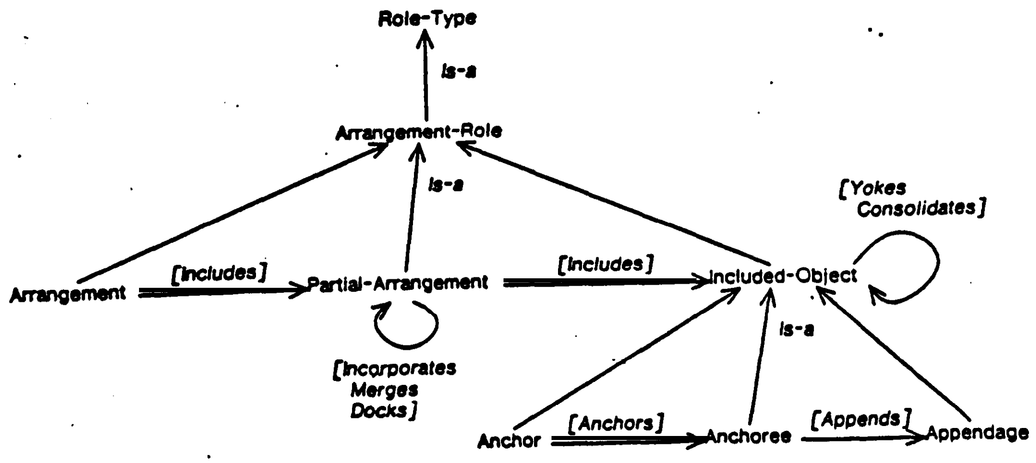


Figure 11. ACCORD's Arrangement-Role Types. An ***arrangement*** *is a* complete solution to **an arrangement** problem **and** may include one or more partial **arrangements.** A **partial-arrangement** is **a partial** solution that includes a subset of the objects. **constraints,** and **contextual regions specified** in **the** problem. **Particular partial-arrangements** an ***incorporate, merge,*** *or dock* **with one** another. ***Included-objects*** an *serve* **as** *anchors*, ***anchorees,*** *or* ***appendages*** within a **partial-arrangement.** *An* anchor an ***anchor*** an snchoree. *An* snchoree an ***append*** **an appendage.** In **addition,** included-objects an ***yoke*** or ***consolidate*** with **one** mother.

[4]**We have not** yet found it **necessary** to **elaborate** similar role types for **constraints** and **contexts,** but we may do so in the future.

Figure 11 also illustrates characteristic relations among solution elements that play particular roles. An arrangement ***includes*** partial-arrangements, which, in turn, ***include*** Included-objects. Anchors ***anchor*** anchorees. Anchorees may ***append*** appendages. Two included-objects may ***yoke*** one another. Three or more included-objects may ***consolidate*** with one another. A partial-arrangement may incorporate, merge, or dock another one.

Finally, ACCORD specifies a number of characteristic attributes and default values for solution elements that play particular roles (not shown in Figure 11). For example, included-object has a ***locations*** attribute, whose default value is Nil, that specifies its legal locations in its partial-arrangement context, given the constraints that have been applied at any point in time. Included-object also has an attribute named ***secure*** whose value is a procedure for rating (O-100) the degree to which an included-object's current locations have been restricted.

4.4 Types of Actions, Events, and States

A framework defines task-specific action, event, and state types as homologous variations on an underlying network of root verbs.

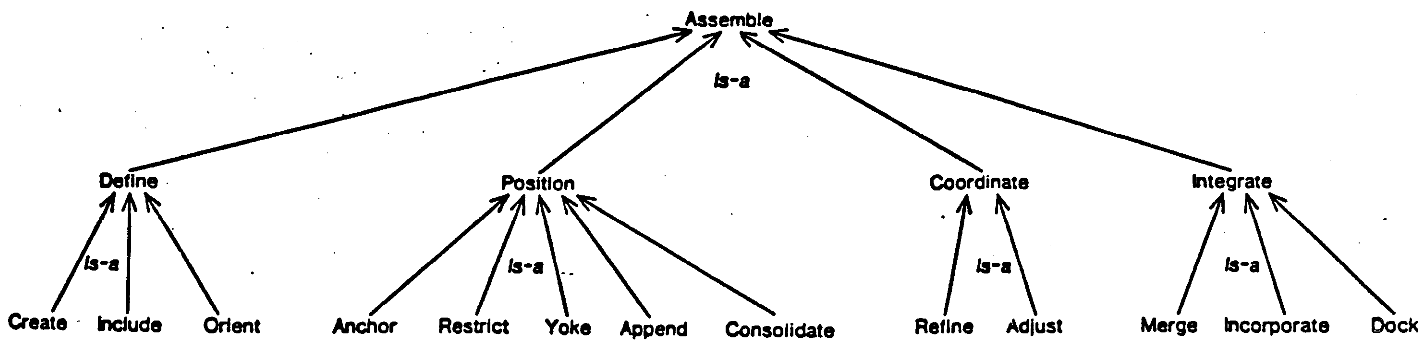


Figure 12 ACCORD's Type Hierarchy of Arrangement-Assembly .Root Verbs. *Assemble* has four subtypes. ***Defining*** a partial arrangement involves ***creating a*** partial arrangement, ***including*** objects in it and ***orienting*** the partial arrangement about a selected anchor. ***Positioning*** objects within a partial smngement may involve, for example, ***anchoring*** and object to the anchor or ***yoking*** two previously positioned anchorees. ***Coordinating*** partial arrangemenu may involve ***refining*** their subordinates at lower levels of abstraction or ***adjusting*** their superordinate at higher levels of abstraction. ***Integrating*** partial arrangements may involve ***merging*** those that have a common anchor. ***Incorporating*** one partial arrangement into another one that shares a common object. or ***docking*** those that include objects that constrain one another.

ACCORD defines the type hierarchy of root verbs shown in Figure 12. The top-level verb, ***assemble,*** means: solve an arrangement problem by means of the assembly method. Assemble has four subtypes. ***Define*** means: construct a partial arrangement that includes particular objects in particular roles. ***Position*** means: identify the locations in which particular objects can lie within a particular partial arrangement while satisfying particular constraints. ***Coordinate*** means= identify the locations in which particular objects can lie within a partial arrangement white satisfying their part-whole relations with previously positioned superordinate or subordinate objects. ***Integrate*** means: combine two partial arrangements to form ‘a single, larger partial arangement Each of the four verb subtypes-define, position, coordinate, and integrate--has two or more subordinate subtypes, as described below.

Define has three sub-types. ***Create*** means: record a blackboard objects representing a new partial arrangement. ***Include*** **means:** create instances of particular objects or constraints’within a particular partial arrangement. ***Orient*** means: declare that a particular objects in a partial arrangement is the anchor and assign the roles anchoree and appendage to other included objects **depending** upon whether or not they have constraints with the anchor.

Position has five subtypes. ***Anchor*** means: identify the locations in which an anchoree . satisfies particular constraints with the anchor. ***Append*** means: identify the locations in which an appendage satisfies constraints with an anchoree or appendage that has already been positioned. ***Yoke*** **means:** prune the locations for two included-objects that have already been positioned so that they include only locations in which the two objects satisfy constraints with one another. ***Restrict*** means: prune the locations identified for an anchoree or appendage to include only those that satisfy additional constraints. ***Consolidate*** means: prune the locations for three or more objects to include only those that satisfy all constraints among the objects simultaneously.

Coordinate has two subtypes. ***Refine*** means: identify locations for a previously positioned objects’s constituent objects so as to satisfy their part-whole relationship. ***Adjust*** means: identify an objects’s ‘locations **to** satisfy its part-whole relationship **with** previously positioned constituent objects.

integrate has three subtypes. Merge means: combine two partial arrangements that have the same anchor. ***Incorporate*** means: combine two partial arrangements that include anchorees or appendages. ***Dock*** means: combine two partial arrangements that have no common objects, but include objects that constrain one another.

ACCORD also specifies entailments of these root verbs (see Figure 13). For example, the anchor verb entails the ***generate*** verb, which means: generate a family for an included-object. Similarly, the position verb entails the ***apply*** verb, which means: apply a constraint to an included-object within a partial arrangement. An implicit ***$entails*** relation holds between two concepts related by any chain of is-a, exemplifies, instantiates, and entails links. For example, we may infer **that:** .

```
Anchor $entails apply.
```

because:

```
Anchor Is-a Position.
. Position entails apply.
```

ACCORD distinguishes homologous type hierarchies for actions, events, and states by different verb **tenses:** Do-verb signifies an action. ***Did-verb*** signifies an event ***Is-verbed*** signifies a state. As illustrated in Figure 13, all relations and attributes in the root verb hierarchy reappear in the action, event, and state type hierarchies.

ACCORD also recognizes implicit states reflecting the existing properties of particular concepts (e.g., Has helix2 shape cylinder) and the relationships between them (e.g., Exemplifies helix1 helix). As a consequence, the number of recognizable state types in an application system greatly exceeds the number of action and event types defined in ACCORD. For reasons of efficiency, ACCORD does not explicitly enumerate all such states, but only those that have important relationships (e.g., is-caused-by, is-entailed-by). to actions, events, or states in the type hierarchy. Nonetheless, it supports verification and assessment of all explicit and implicit states in the conceptual network.

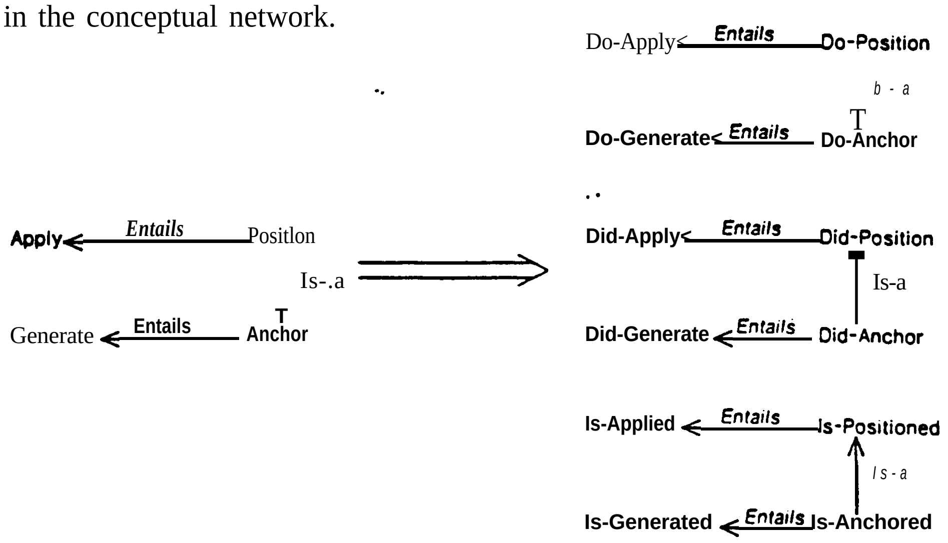


Figure 13. Homologous Action. Event. and State Subnetworks. The root verb hierarchy underlies homologous action, event, and **state type hierarchies.** distinguished by verb tense. Do-<verb> signifier an action. ***Did-<verb> signifies*** an **event.** ***Is-<verbed>*** signifia a **state.** Implicit **$<links>** indicate. for example. that do-anchor actions **$entail** do-apply **actions.**

4.5 Relations among Actions, Events, and States

A framework specifies legal relations among different types of actions, events, and states [1, 36]. Events of a particular type can *trigger* actions of a particular type, that is, indicate that the actions are potentially feasible. States of a particular type can ***enable*** triggered actions of a particular type, that is, render the triggered actions feasible. Actions of a particular type can ***cause*** events of a particular type. Finally, events of a particular type can ***promote*** states **of** a particular type. Figure 14 illustrates some of the legal relations specified in the ACCORD knowledge base.

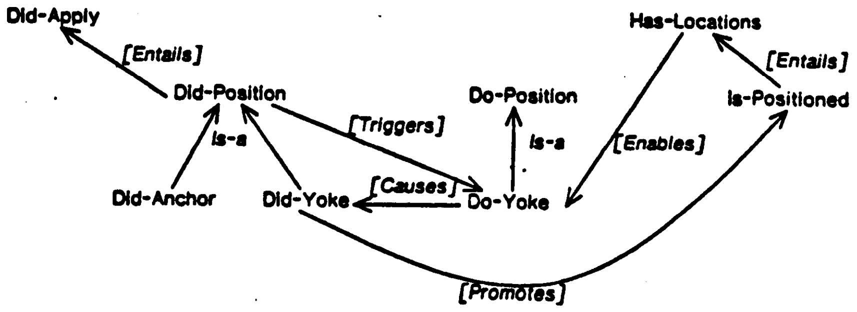


Figure 14. Some **Legal** Relations among Actions, **Events, and States.** Did-position events ***trigger*** do-yoke actions. which must he ● ***nubled*** by has-locations states. When executed. **do-**yoke actions cause **did-yoke events,** which *promote* is-positioned states Implicit **$<link>** relations indicate. **for example,** that did-anchor **events *$trigger*** do-yoke actions and **that** do-yoke actions ***$cause*** did-apply events.

An implicit ***$<links>*** form of each of these relations:

```
        A [$<links>] B
```

holds for any two concepts, a and b, whenever:

```
        a $is-a A or a $entails A.
               and
        B $is-a b or B $entails b.
```

. For example, we may infer from Figure 14 that:

```
        Did-anchor [$triggers] do-yoke.
```

because:

```
        Did-anchor is-a did-position.
        Did-position [triggers] do-yoke.
```

Similarly, we may infer that:

```
        Do-yoke [$causes] did-apply.
```

because:

```
        Do-yoke [causes] did-yoke.
        Did-yoke $entails did-apply.  .
```

Note that legal relations such as those specified in Figure 14 may not actually hold among all individual actions, events, and states of the specified types. For example, a did-position event **can** trigger a do-yoke action. But an individual did-position event may require additional attributes (discussed below) in order to trigger an individual do-yoke action.

4.6 Linguistic Templates for' Actions, Events, and States

A framework provides linguistic templates for all root verbs and their entailments. Each template comprises a ***verb keyword,*** followed by a specified sequence of ***formal parameters,*** interspersed with optional conjunctions and prepositions (noise words). Particular actions, events, or states are represented **as** ***patterns*** that instantiate the formal parameters of particular templates with particular concept types, individuals, or instances. In addition, each keyword and formal parameter value in a pattern may be preceded by any number of modifiers and followed by a local variable. name in parentheses.

Table 4 shows ACCORD's templates for the arrangement-assembly root verbs. (For brevity, we omit ACCORD's templates for entailed verbs.) For example, the anchor template is':

```
    Anchor anchoree to anchor in pa with constraint.
```

Here, the keyword, anchor, is followed by the sequence of formal parameters: anchoree, anchor, pa, constraint, with some parameters preceded by the declared noise words: to, in, with.

A system instantiates these templates with domain-specific entities to form particular action, event, and state patterns. For example, PROTEAN might instantiate the anchor template as this action pattern:

Do-anchor helix2-1 to helix1-1 in pa1 with NOE1.

PROTEAN could represent a larger class of actions with this pattern:

Do-anchor helix to helixl-1 in pal with constraints.

It could represent a restricted class of actions by inserting modifiers before some parameter values, as in this pattern:

Quickly do-anchor long helix to helix1-1 in pal with strong constraints.

PROTEAN could instantiate event and state patterns in a similar fashion by substituting the appropriate did-verb or is-verbed forms of the root verbs,

Table 4. Templates for Arrangement-Assembly Root Verbs.

- Assemble pa
  - Define pa
    - Create pa at level
    - Include object in pa
    - Orient pa about included-object
  - Position object in pa with constraints
    - Anchor anchoree to anchor In pa with constraints
    - Restrict included-object in pa with constraints
    - Yoke included-object to included-object in pa with constraints
    - Append 'appendage to included-object in pa with constraints
    - Consolidate included-obiects in pa with constraints
  - Integrate pa with pa
    - Merge pa with pa
    - Incorporate pa into pa via included-object
    - Dock pa to pa with constraints
  - Coordinate pa at level and level
    - Refine sub-object of object in pa from level to level
    - Adjust object for sub-object in Pa

4.7 Partial Matches Among Templates

A framework defines the **potential partial matches** among action, event, and state patterns by identifying corresponding parameters in their underlying templates. (These correspondences need not be one-to-one.) Two patterns match to the degree that the values of their corresponding parameters match. For example, Figure 15 identifies corresponding parameters in the assemble, position, and anchor template.

Consider the position and anchor templates. By definition, the two keywords; position and anchor, correspond. In this context, the formal parameters included-object and anchoree correspond because they both represent objects that the actions position. The two formal parameters called pa correspond because they both represent the partial arrangement in which the actions occur. The two formal parameters called constraints correspond because they both represent constraints that the actions apply. The anchor template's formal parameter called anchor does not correspond to anything in the position template because the position template does not specify an object that lies at the center of the designated local coordinate system.

Given this knowledge, a system can assess the degree to which two patterns match by assessing the matches between their formal parameter values. For example, PROTEAN can assess the degree to which the pattern:

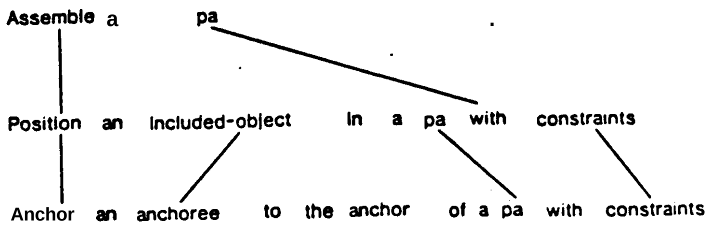
```
Anchor helix2-1 to helix1-1 in pal with NOE1.
```

matches the pattern:

```
Position long helix in pal with strong constraint.
```

by assessing the matches of:

```
anchor against position:
helix2-1 against long helix:
pal against pal:
NOE1 against strong constraint.
```



Assemble a pa
Position an included-object in a pa with constraints
Anchor an anchoree to the anchor of a pa with constraints

Figure 15. Partial Matches between Assemble. Position. and Anchor Templates. Partial matches identify semantically corresponding formal parameters in all pairs of templates. In theses examples: Assemble, position. and anchor all represent verb keywords. Included-object and anchoree represent objects being positioned. All Parameters called pa refer to the partial arrangement. Parameters called constraints represent constraints to be applied.

4.8 Template Translations

Since a framework such as ACCORD must be applied in the context of a computational architecture, it provides the'knowledge necessary to translate certain framework templates into semantically equivalent templates in the language of the chosen architecture. In our work, we use the BB1 blackboard control architecture (see section 2 above) and ACCORD provides knowledge for translating arrangement-assembly templates into BB1 templates. So far, we have found it necessary to provide such knowledge for terminal action patterns and for all state patterns. In both cases, translation knowledge comprises the parameterized framework templates and the semantically equivalent parameterized BB1 templates, with corresponding parameters of the same names. Thus, BB1 can translate patterns between representations by means of a variable-substitution procedure discussed below.

For example, Figure 16 shows the BB1 template for the do-anchor action. As this example illustrates, each BB1 action template is a parameterized program of rules that evaluate lisp expressions, set local variables, and modify objects on the blackboard or in the knowledge base. (Note that all application-specific routines for constraint satisfaction are inserted indirectly through calls to ACCORD's generic *CSS-<extension>* functions.) Both do-anchor templates refer to the parameters: anchoree, anchor, pa, and constraints.

```
ACCORD Template: Anchor Anchoree to Anchor in PA with cons-

BB1 Template:
        ((1 (T)
                ((EXECUTE ($Set Constraints (CONSTRAINTS-IN Constraints)))
                 (EXECUTE ($Set CSS-Anchor-Results (CDR (CSS-ANCHOR Anchoree
                        Anchor PA Constraints))))
                 (PROPOSE changetype MODIFY object Anchoree attributes
                        CSS-ANCHOR-RESULTS))))

PROTEAN CSS-ANCHOR Function:
        (PROG (AbTable PObiect PAnchor PConstraints Sample-Vector Oescriptlon
                CalcLocAns DescribeAns)
            (SETQ AbTable (CSS-GENERATE-TABLE-NAME Object Anchor
                Constraints PA 'Anchor)
            (SETQ PObiect ($SHORT-NAME (SOSJECT Object 'Instantlater)))
            (SET0 PAnchor (SSHORT-NAME (SOSJECT Anchor 'Instantlater)))
            (SETQ PConstraints ($SHORT-NAME Constraints))
            (SETQ Sample-Vector. '(2 2 2 30 30 30))
            (SETQ Description (LIST 'Anchor PObiect 'to PAnchor))
            (SET0 CalcLocAns (GS-CALCULATE-LOCATIONS AbTable NIL PAnchor
                PObiect PConstraints NIL Description Sample-Vector NIL))
            (IF (NULL (CAR CalcLocAns))
                THEN (RETURN CalcLocAns))
            (SETQ DescribeAns (GS-OESCRIBE-LOCATIONS AbTable PAnchor
                PObiect PConstraints 'GS-CALCULTATE-LOCATIONS
                (DATE) Description))
            (RETURN (CDR DescribeAns)))
```

Figure 16. ACCORD and BB1 Templates for the Do-Anchor Action. Both templates refer to the same parameters, which can be instantiated to define specific action Patterns. The ACCORD template is essentially a macro for the more complex underlying BB1 program of rules. Note that all application-specific routines for constraint satisfaction are inserted indirectly through calls to ACCORD's generic CSS-<extension> functions.

Figure 17 shows the BB1 template **for the is-anchored state.** As **this** example illustrates, each BB1 state template is a parameterized program of blackboard access functions. Both is-anchored templates refer to the parameters: anchoree, anchor, pa, constraints.

In addition to these explicitly stored state translations, BB1 automatically translates any has-attribute state pattern instantiating the prototypical framework template:

Has object attribute value

into the equivalent prototypical BB1 template:

(Equal ($Value object attribute) value).

ACCORD Template: Is-Anchored Anchoree to Anchor in PA with Constraints.

BB1 Template:

```
((EO ($OBJECT Anchoree 'Anchored-by) Anchor)
 (FMEMB Anchor ($OBJECTS PA Includes))
 ($OBJECT· Anchoree 'Located-by)
 (EQ (SVALUE ($OBJECT Anchoree 'Located-by) 'Constraint-Set-Used)
    Constraints))
```

Figure 17. ACCORD and BB1 Templates for the Is-Anchored **State.** Both templates refer to the same **parameters,** which can be instantiated **to** define specific **state patterns The ACCORD template is essentially a macro for the more complex underlying BB1 program of access functions.**

5. The BB1 Framework-Interpreter

To support the application of frameworks, we have extended the BB1 architecture with a ***framework-interpreter:*** a collection of procedures for ***parsing*** patterns, ***matching*** patterns, ***quantifying*** the match between two patterns, ***generating*** an ordered list of quantified instantiations of a pattern, and ***translating*** framework patterns into BB1 patterns. The BB1 framework-interpreter applies to <u>any</u> <u>user-specified framework</u> defined with the BB1 knowledge structures illustrated above for ACCORD. In addition, BB1 can accommodate heterogeneous systems, applying the new procedures to framework knowledge structures and its standard procedures to BB1 knowledge structures. Section 6 below shows how BB1 uses the framework-interpreter during problem solving.

5.0.1 Parsing Patterns

The BB1 ***parser*** converts patterns from their English form to a parsed form for use by the matcher, quantifier, generator, and translator. The parser first removes noise words (conjunctions and prepositions) from a pattern. It then works left to right, using recognized verb keywords and the sequence of parameters in their associated templates to identify the pattern's constituent phrases. The parser produces a list of simple lists, each of which contains a single parameter value and the modifiers that precede it in the pattern. For example, the parser would parse the pattern:

```
          Quickly do-anchor long helix to helix1-1 in pal
                    with strong constraint.
```

as the list:

```
((d&anchor  Quickly)
 (helix long)
 (hel ixl-1)
 (pa1)
 (constraint  strong))
```

Other interpretation procedures access particular parameter phrases according to their sequential positions in the templates and parsed lists.

5.0.2 Matching Patterns

The BB1 ***matcher*** assesses whether a ***test pattern*** matches a ***target pattern.*** For each corresponding parameter in the two patteins, the matcher declares a match whenever the test pattern value has a $is-a, $entails, or $plays relation with the target pattern value. A ***perfect match*** is one in which the matcher declares a match for all parameters (verbs and nouns) in the target pattern. However, the matcher uses the partial-match knowledge described above to assess the partial match between any two patterns, regardless of the number of corresponding parameters between them. Figure 18a illustrates a perfect match between two PROTEAN action patterns.

5.03 Quantifing a Match

The BB1 ***quantifier*** records a numerical assessment of the match between each parameter value in a test pattern and: (a) its corresponding parameter value in a target pattern; and (b) each modifier of the corresponding parameter value in the target pattern. It records 0 for each non-matching parameter value and 100 for each matching parameter value. For non-matching parameters, the quantifier also records 0 for each modifier of the parameter value in the target pattern. For matching parameter values, it records for each modifier a number between 0 and 100, which it obtains from the attribute named by the modifier. A ***perfect quantified match*** is one in which the test pattern receives a value of 100 for all parameters in the target pattern and their associated modifiers. Again, however, the quantifier numerically assesses the degree of match between any two patterns regardless of the number of corresponding parameters. Figure 18b illustrates a quantified match between two PROTEAN action patterns.

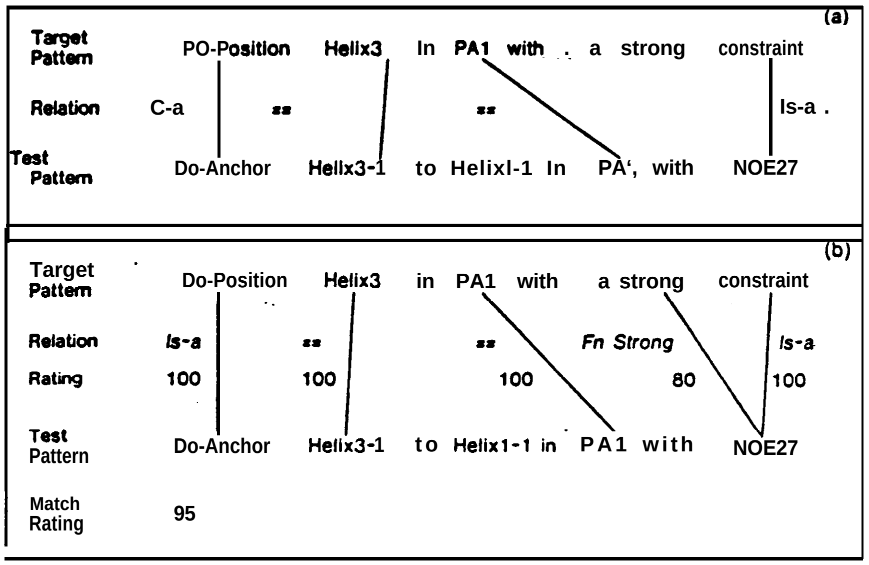

(a)
Target Pattern
PO-Position Helix3 In PA1 with a strong constraint
Relation
C-a
=
=
Is-a .
Test Pattern
Do-Anchor Helix3-1 to Helixl-1 In PA', with NOE27
(b)
Target Pattern
Do-Position Helix3 in PA1 with a strong constraint
Relation
Is-a
=
=
Fn Strong
Is-a
Rating
100
100
100
80
100
Test Pattern
Do-Anchor Helix3-1 to Helix1-1 in PA1 with NOE27
Match Rating
95

Figure 18. Matching Two Action Patterns. (a) The tat pattern produces a *perfect* march to the target pattern because: Do-anchor is-a do-position action. Helix3-1 is helix3-1. Pal is pal. NOE1 is-a constraint (b) The *match rating*, 95, combines component ratings for each puameter and modifier in the target pattern. proportionate to their weights. In this case, the perfect match entails ratings of 100 for each parameter and NOE27 rates 80 against the modifer. strong.

As discussed above, modifiers may specify threshold or scale procedures with the extensions **"-T"** or **"-S"** to the modifier name. However, **BB1** knows in which circumstances threshold and **scale** procedures typically apply and uses the appropriate one if no extension appears in the **named** modifier. For example, **BB1** uses threshold procedures to quantify matches underlying its all-or-none triggering decisions and scale procedures to quantify matches underlying its **graded ratings** of pending **KSARs.**

5.0.4 Generating an Ordered Lii of Quantified Matches

The **BB1** generator generates **all** ***(or*** a specified number of) values for a **designated** parameter that legally instantiate a set of patterns or ***phrases. The*** generator first follows links in the concept networlc to find values that match parameter values and associated threshold modifiers and relations specified in **the** input patterns. It then rates each value 'against associated scale modifiers in the input patterns. **It** returns all values and their ratings, "best **first."** For example, Figure 19 **illustrates generation of** all long **helices** that are positioned in some partial arrangement

```
Generate X such that:
            b-r x Long Helix
            Plays X Included-Objects
            Is-Positioned X

b-a X (Long) Helix
            -> ($ALL-OBJECTS Helix 'Can-be-a)
               = (Helix1 Helix2 Helix3
                  Helix1-1 Helix2-1 Helix3-1)

Plays X Included-Object
            -> (Hellxl-1 Helix2-1 Helix3-1)

Is-Positioned X
            -> (Helix 1-1 Helix2-1 Helix3-1)

b-a X Long Helix
            -> ((Helix1-1 (90)) (Helix3-1 (70)) (Helix2-1 (40)))
```

Figure 19. Generation of Parameter **Values.** This set of expressions ***generates*** **all** long helixes **that** are **positioned** in some partial arrangement. bat first. First the generator generates all legal values for X to instantiate the state, Is-a helix. Then it prunes this set to include **only** . legal valua of X to instantiate the state, Plays X included-object Then it **prunes** the reduced set to include only legal **values** of X to instantiate the state. Is-positioned **X.** Finally, it orders the **remaining set** according to the rating of each value in the **phrase, Long X.**

5.0.5 Translating Between Framework and BB1 patterns

The BB1 *translator* uses a variable-substitution procedure to translate framework and BB1 patterns into one another. For example, Figure 20 illustrates the translation of an ACCORD pattern for the do-anchor action into the semantically equivalent BB1 action pattern.

```
ACCORD Template: Anchor Anchoree to Anchor in PA with Constraints.

BB1 Template:
        ((1 (T)
                ((EXECUTE ($Set Constraints (CONSTRAINTS-IN Constraints*)))
                 (EXECUTE ($Set CSS-Anchor-Results (CDR (CSS-ANCHOR Anchoree
                        Anchor PA Constraints))))
                 (PROPOSE changetype MODIFY object Anchoree attributes
                        CSS-ANCHOR-RESULTS))))

                        [CSS-ANCHOR . . . ]

ACCORD Pattern: Do-Anchor Helix2-1 to Helix1-1 In PA1 with CSet1.

BB1 Pattern:
        ((1 (T)
                ((EXECUTE ($Set Constraints (CONSTRAINTS-IN CSet1)))
                 (EXECUTE ($Set CSS-Anchor-Results (CDR (CSS-ANCHOR Helix2-1
                        Helix1-1 PA1 CSet1)))))
                 (PROPOSE changetype MODIFY object Helix2-1 attributes
                        CSS-ANCHOR-RESULTS)

                        [CSS-ANCHOR . . . ]
```

Figure 20. Translation of Action Patterns. The translator substitutes the parameter values in the ACCORD pattern for the corresponding parameters in the BB1 template.

6. Reasoning within a BB* Application System

6.1 The Layered Architecture of a BB* System

Application systems built within the BB* environment have layered architectures: application-specific knowledge is layered on the task-specific knowledge of an appropriate framework, which' is layered in turn on the architectural knowledge in BB1. For example, PROTEAN layers PROTEAN-specific knowledge on the arrangement-assembly knowledge of ACCORD, which is layered on the architectural knowledge of BB1.

Application-specific knowledge typically extends the task-specific framework knowledge in four areas. First, the application instantiates skeletal branches of the concept network to define domain entities. For example, PROTEAN extends ACCORD's type hierarchy to define biochemical objects (i.e., protein structures) and constraints (e.g., NOEs) and to identify the individual objects and constraints involved in particular proteins (e.g., helix1 in the lacrepressor headpiece). Section 3 above illustrates these extensions to ACCORD's concept network. Second, the application specifies knowledge sources that instantiate the framework's action templates as feasible actions during problem solving. For example, PROTEAN's knowledge sources, which are discussed below, instantiate ACCORD's assembly action templates. Third, the application provides special-purpose programs required to execute feasible actions. For example, PROTEAN provides a geometric constraint-satisfaction system [3], which is implemented in C and run remotely over a network, for use in executing instantiated assembly actions. Finally, the application specifies control knowledge sources that instantiate the framework's templates as strategic plans to guide the system's actions during problem solving. For example, PROTEAN's control knowledge sources, which. are discussed below, instantiate ACCORD's templates as strategic plans for assembling proteins.

6.2 Domain Reasoning in a BB* System

6.2.1 Domain Knowledge Sources

Like a standard BB1 application system, a BB* system uses domain knowledge sources to solve problems. These knowledge sources monitor the events and states that occur during problem solving. When criteria1 events and states occur, they instantiate feasible problem-solving actions (recorded as KSARs on BB1's agenda), which compete for scheduling priority, Unlike a standard BB1 system, a BB* system can express actions. events, and states of interest as instantiated framework patterns. For example, Figure 21 shows how PROTEAN's knowledge **source Yoke-Structures** instantiates particular assembly actions, events, and states. As discussed in the following sections, a **BB*** system can use the **BB1** framework-interpreter to perform all associated computations.

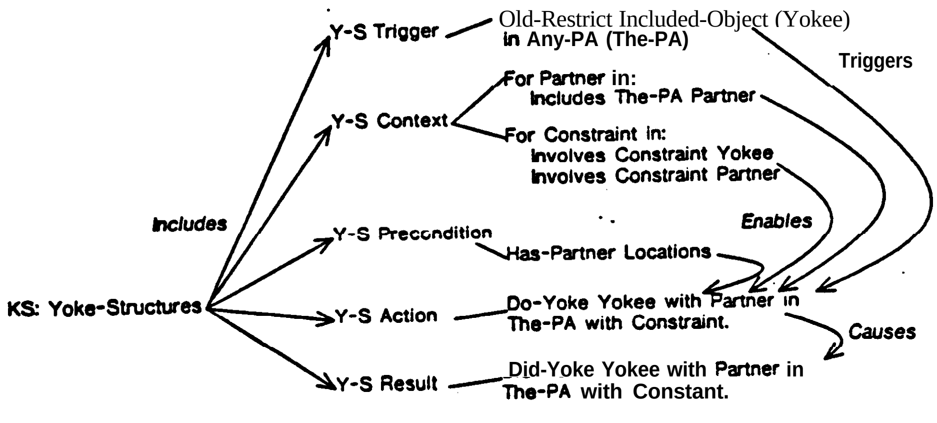

Y-S Trigger
Old-Restrict Included-Object (Yokee) In Any-PA (The-PA)
Triggers
Y-S Context
For Partner in: Includes The-PA Partner
For Constraint in: Involves Constraint Yokee Involves Constraint Partner
Includes
Enables
Y-S Precondition
Has-Partner Locations
KS: Yoke-Structures
Y-S Action
Do-Yoke Yokee with Partner in The-PA with Constraint.
Causes
Y-S Result
Did-Yoke Yokee with Partner in The-PA with Constant.

Figure 2L A Domain Knowledge Source in **BB1-ACCORD.** Each attribute of the **knowledge** source **Yoke-Structures** is represented **as action. event,** or state patterns. with appropriate links **among** them.

6.2.2 Triggering

BB1 triggers a knowledge source whenever it assesses a perfect quantified match of a new blackboard event against the knowledge source trigger patterns. At the same time, it binds the value of each parameter in the trigger patterns to the specified local variable name (or, if none is specified, to an internally generated name).

For example, Yoke-Structures's trigger comprises one did-restrict event pattern:

```
Did-restrict included-object (yokee) in any-pa (the-pa).
```

BB1 would trigger Yoke-Structures for this blackboard event:

```
Did-anchor helix2-1 to helix1-1 in pal with NOE1.
```

because:

```
Did-anchor $entails did-restrict.
Helix2-1 Splays included-object.
Pal Sis-a pa.
```

In this case, BB1 would bind two local variables: yoket to helix24 and the-pa to pal.

6.2.3 Context Generation

A knowledge-source context comprises a nested set of expressions of the form:

```
'For <variable> In <state patterns>.
```

For each such expression, BB1 generates and identifies as a context each unique combination of variable-value pairs that match the pattern. If several such expressions are nested, BB1 applies this procedure recursively. It generates a KSAR for each identified context and places all generated KSARs on the agenda

For example, Yoke-Structures's context comprises two expressions:

```
For partner in:
  Includes the-pa partner.
  Not Is yokee partner.
For constraint in:
  Involves constraint yokee.
  Involves constraint partner.
```

Let us continue the example begun above. Based on the first expression, BB1 generates alternative values of the context variable, partner: all objects that are included by pal (the-pa), excluding helix24 (yokee). Supposing that pal includes one such object, BB1 generates one value of partner: helix3-1. Based on the second expression, BB1 generates for each value of partner alternative values cf the context variable, constraint: all constraints that involve helix3-1 (partner) and helix2-1 (yokee). Supposing that two such constraints exist, BB1 generates two values for constraint: NOE6 and NOE8. Finally, BB1 generates a unique context representing each combination of context-variable values and generates a separate KSAR for each context, for example KSAR 50 in Figure 22.

6.2.4 Precondition Checking

A knowledge-source precondition comprises any number of state patterns that must match information on the blackboard or in the knowledge base before the KSAR can execute its action. For each KSAR, BB1 translates and evaluates each precondition pattern, performing specified variable bindings along the way. If all preconditions evaluate to true, BB1 places the KSAR on the agenda of executable actions where it competes for scheduling priority. If any do not evaluate to true, BB1 places the KSAR on the agenda of triggered actions and rechecks unsatisfied preconditions on each cycle until all are true.

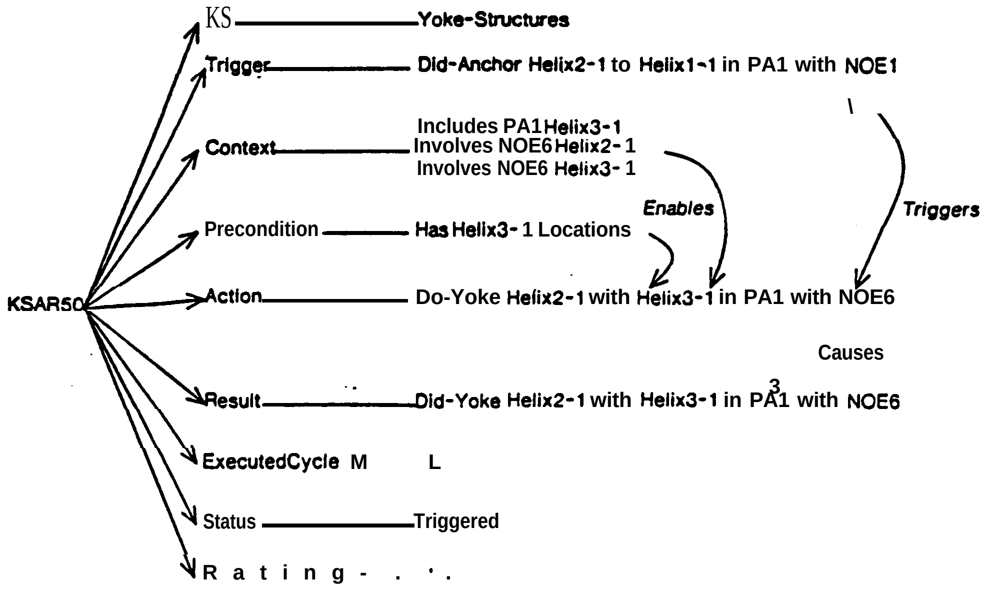


Figure 22 A Domain KSAR in BB1-ACCORD. Each attribute of this Yoke-Structures KSAR is represented as action, event or state patterns, with appropriate links among them. Each of these patterns instantiates the corresponding pattern in the Yoke-Structures knowledge source. For example, KSAR50's trigger event. *Did-anchor helix2-1 to helix1-1 in pa1 with NOE1* matches Yoke-Structures's trigger event, *Did-restrict included-object (yokee) in any-pa (the-pa)* because did-anchor entails did-restrict, helix2-1 plays included-object and pa1 plays pa Similarly, KSAR50's action, *Do-yoke helix2-1 with helix3-1 in pa1 with NOE6* instantiates Yoke-Structures's action, *Do-yoke yokee with partner in the-pa with constraint* because helix2-1 is the bound value of yokee, helix3-1 is the bound value of partner. pa1 is the bound value of the-pa, and NOE6 is the bound value of constraint

For example, Yoke-Structures's precondition:

```
Has partner locations.
```

specifies that Yoke-Structures can execute its action only when the previously identified partner has an attribute named locations whose value is not nil. For KSARSO above, BB1 translates this pattern into the BB1 pattern:

```
($Value helix34 locations)
```

and evaluates it. If it evaluates to true, BB1 determines that KSAR1 is executable.

6.2.5 Action Execution

A knowledge-source action is a terminal action pattern whose parameters are bound within a KSAR during the triggering, context-matching, and precondition-evaluation procedures described above. When BB1 decides to execute a particular KSAR, it translates the action pattern into the equivalent BB1 action and sends it to BB1's low-level action interpreter.

For example, KSARSO specifies the action pattern:

```
Do-yoke helix2-1 with helix3-1 In pal with csetl.
```

BB1 translates this pattern into the equivalent BB1 **action** pattern (see in Figure 20) and sends it to the low-level action interpreter for. execution.

6.2.6 Event Generation

A knowledge source result is a terminal event pattern that corresponds to the knowledge source action pattern. Within a KSAR, corresponding parameters in the action pattern and result pattern have identical values. When BB1 executes the action of the KSAR, it generates the event pattern and records it on its internal event *list* for use during knowledge source triggering,

For example, in executing KSARSO, BB1 generates the event pattern:

```
Did-yoke helix2-1 with helix3-1 In pal with csetl.
```

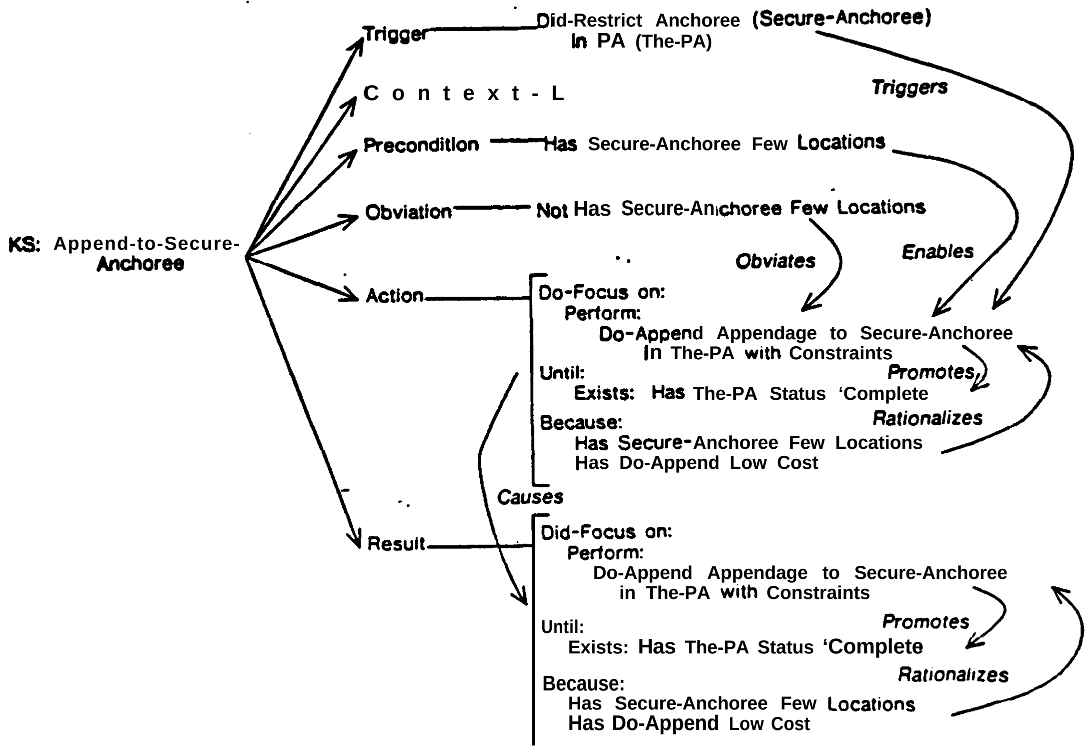
6.2.7 Advantages of Domain Reasoning in BB*

BB* offers three important advantages for domain reasoning. First, it provides a superior--concise, perspicuous, uniform, modular, interpretable--representation for knowledge sources, events, and KSARs. Second, it permits provides powerful framework-interpreter procedures for all operations performed on these knowledge structures. Third, its layered approach reveals the distinctions among domain-specific, task-specific, and task-independent knowledge.

6.3 Control Reasoning in a BB* System

63.1 Control Knowledge Sources

Like a standard BB1 application system, a BB* system uses control knowledge sources to generate strategic plans for its own actions in real time. These knowledge sources monitor the events and states that occur during problem solving. When criteria1 events and states occur, they instantiate feasible actions for extending or mddifying the -current control plan. These actions (recorded as KSARs on BB1's agenda) compete with one another and with instantiated domain actions for scheduling priority. Unlike a standard BB1 system, however, a BB* system can express actions, events, and states of interest as instantiated framework templates. Similarly, it can use the BB1 framework-interpreter to perform all associated computations.



KS: Append-to-Secure-Anchoree
Trigger
Did-Restrict Anchoree (Secure-Anchoree) In PA (The-PA)
Context-L
Triggers
Precondition
Has Secure-Anchoree Few Locations
Obviation
Not Has Secure-Anchoree Few Locations
Obviates
Enables
Action
Do-Focus on:
Perform:
Do-Append Appendage to Secure-Anchoree In The-PA with Constraints
Until:
Promotes
Exists: Has The-PA Status 'Complete
Because:
Rationalizes
Has Secure-Anchoree Few Locations
Has Do-Append Low Cost
Causes
Result
Did-Focus on:
Perform:
Do-Append Appendage to Secure-Anchoree in The-PA with Constraints
Until:
Promotes
Exists: Has The-PA Status 'Complete
Because:
Rationalizes
Has Secure-Anchoree Few Locations
Has Do-Append Low Cost

Figure 23. A Control Knowledge Source in BB1-ACCORD. Each attribute of the knowledge source Append-to-Secure-Anchoree is represented as action. event. or state patterns, with appropriate links among them. Similarly. Append-to-Secure-Anchoree's action and result are control actions and events (*do-focus-on* and *did-focus-on*) whose parameters (*prescription, goal,* and *rationale*) are represented as action, event, and state patterns, with appropiate links among them.

For example, Figure 23 shows the control knowledge source: Append-to-Secure-Anchorees. BB1 applies its framework-interpreter to control knowledge sources exactly as it does for domain knowledge sources. For example, the event:

Did-anchor helix2-1 to helixl-1 in pal with NOE1.

in which helix2-1 was restricted to a criterially small number of locations would produce KSAR51, shown in Figure 24. When executed, the KSAR would record a decision with the specified attributes at the focus level of the control blackboard.

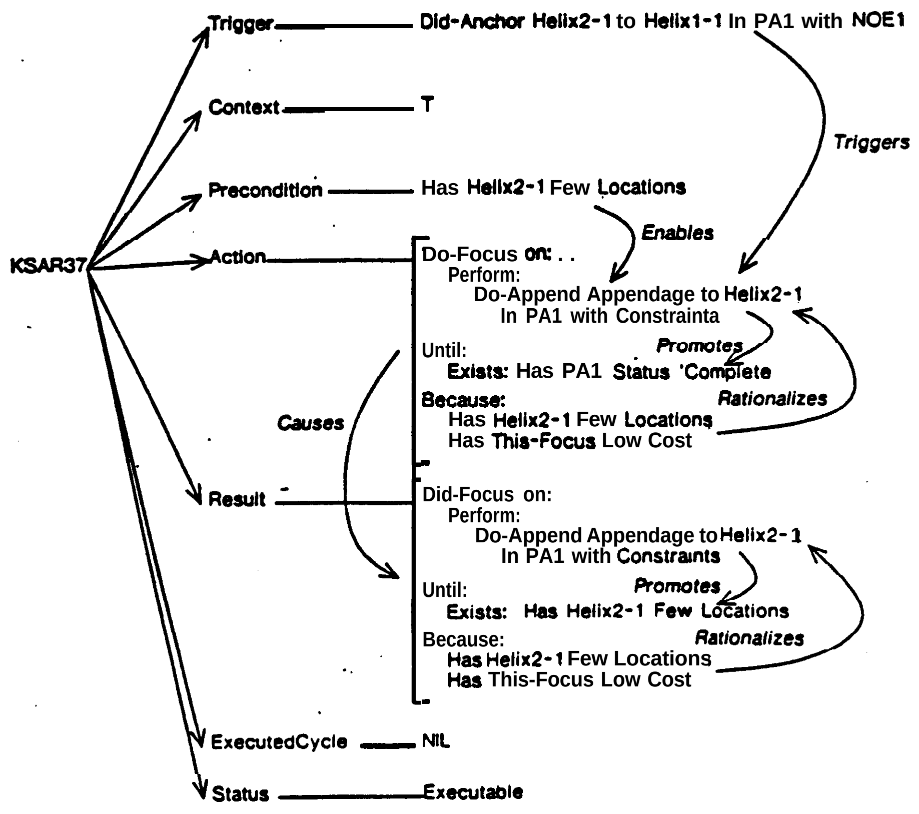


Figure *24.* A Control KSAR in BB1-ACCORD. This Append-to-Secure-Anchoree KSAR is represented as action. event, or state patterns. with appropriate links among them. Each of these patterns matches or instantiates the corresponding pattern in the Append-to-Secure-Anchoree knowledge source. For example, KSAR37's trigger event, *Did-anchor helix2-1 to helixl-1 in pal wirh* NOEI matches Append-to-Secure-Anchoree's trigger event. *Did-restrictanchoree (secure-anchoree) in pa (the-pa)* because did-anchor entails did-restrict, helix2-1 plays anchoree and pal plays pa. In addition,. KSAR37's action and result are control actions and events whose parameters instantiate the corresponding patterns in the Append-to-Secure-Anchorees knowledge source (see Figure 23). For example, *Do-append uppenduge to hellx2-1* in *pal wirh constraints* instantiates *Do-append uppenduge to secure-unchortt in the-pa with constraints* because helix2-1 is the value bound to secure-anchoree.

6.3.2 Control Plans

Like a standard BB1 application system, a BB* system constructs explicit control plans at multiple levels of abstraction. High-level strategy decisions prescribe sequences of subordinate decisions, each of which typically encompasses a shorter problem-solving time interval than its superordinate. All branches of a control plan terminate in *focus* decisions, which the BB1 scheduler uses to rate pending KSARs. Unlike a standard BB1 system, a BB* system can represent control decisions as instantiated action, event, and state templates.

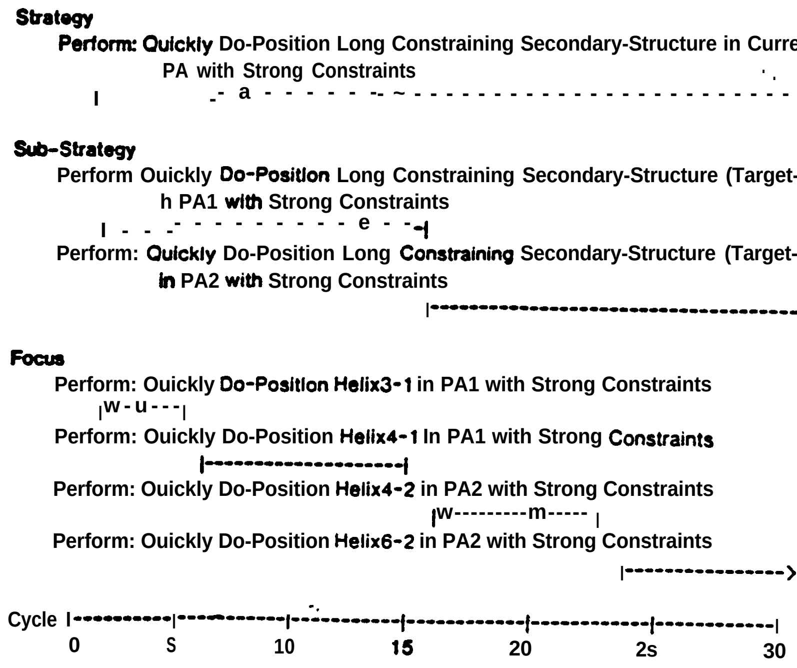

Strategy
Perform: Quickly Do-Position Long Constraining Secondary-Structure in Current-Best PA with Strong Constraints
Sub-Strategy
Perform Ouickly Do-Position Long Constraining Secondary-Structure (Target-Object) h PA1 with Strong Constraints
Perform: Quickly Do-Position Long Constraining Secondary-Structure (Target-Object) in PA2 with Strong Constraints
Focus
Perform: Ouickly Do-Position Helix3-1 in PA1 with Strong Constraints
Perform: Ouickly Do-Position Helix4-1 In PA1 with Strong Constraints
Perform: Ouickly Do-Position Helix4-2 in PA2 with Strong Constraints
Perform: Ouickly Do-Position Helix6-2 in PA2 with Strong Constraints
Cycle
0
S
10
15
20
2s
30

Figure 25. Excerpt from a PROTEAN Control Plan in BB1-ACCORD. ACCORD clearly articulates the hierarchical relationships betweer control decisions: each higher-level decision summarizes and prescribes a sequence of subordinate decisions to obtain during its constituent time intervals. In this example, the generic control knowledge source, Refine-Parameters, generates the excerpted plan automatically. Starting with the top-level strategy, it substitutes the values pal and then pa2 for the phrase, current-best pa. to generate the sequence of two sub-strategies. For each sub-strategy, it similarly substitutes values best first for the phrase, long constraining secondary-structure. to generate a sequence of focus decisions. ACCORD provides concise and perspicuous representations of the goals and rationales of all control decisions.

For example, Figure 25 shows an excerpt from a PROTEAN control plan. Each higher-level decision in this plan clearly summarizes and prescribes its subordinate decisions. For example, the first sub-strategy decision:

```
Perform:
  Quickly do-position long constraining secondary-structure
    (target-object) in pal with strong constraints.
```

summarizes and prescribes its subordinate focus decisions:

```
Perform:
  Quickly do-position helix3-1 in pal
    with strong constraints.

Perform:
  Quickly do-position helix4-1 in pal
    with strong constraints.
```

because **helix3-1** is the longest, most constraining secondary-structure in partial arrangement pal and **helix4-1** is the runner-up. Similarly, although it does not appear in Figure 25, the goal of the sub-strategy decision:

```
Has taraet-object few locations.
```

summarizes and prescribes **the** goal of its subordinate focus decisions:

```
Has helix3-1 few locations.

Has helix4-1 few locations.
```

Notice also that each control decision in Figure 25 captures the meaning of a set of interacting control heuristics, while preserving their individual modularity. For example, the first focus decision in Figure 25:

```
Perform:
  Quickly do-position helix3-1 in pal with strong constraint.
```

captures these heuristics:

```
Prefer KSARs that execute do-position actions.
Prefer KSARs that execute actions in this priority order:
  do-anchor > do-yoke > do-restrict > do-consolidate > do-append.
Prefer KSARs that operate on helix3-1.
Prefer KSARs that operate in the context of pal.
Prefer KSARs that apply constraints.
Prefer KSARs that apply strong constraints.
```

Similarly, although the goal of this decision:

```
Has helix3-1 few locations.
```

represents **a** single **BB1** access function, the goals of other decisions can capture the meaning of any program of access functions.

Finally, the knowledge in a framework permits control decisions to specify desirable actions in terms of the actions themselves, the events that trigger them, the states that enable them, the events they cause, or the states they promote. Table 5 shows examples of these other kinds of prescriptions. Similarly, the goal of a control decision can specify desirable conditions in terms of any state of the knowledge base or *any* blackboard. Table 6 shows examples of different kinds of goals.

Table 5. Examples of ACCORD Prescriptions

1. Perform an action In a particular class of actions.
   Perform: Do-Position Long Helix In PA1 with Strong Constraint.

2. Perform an action that was triggered by a particular class of events.
   Respond-to-Events-thatz Did-Restrict Well-Restricted Anchoree in PA1 with Constraint.

3. Perform an action that was enabled by a particular class of states.
   Respond-to-States-in-which: Has Anchoree Few Locations.

4. Perform an action that causes a particular class of events.
   Cause: Did-Restrict Helix2-1 in PA1 with Constraint.

5. Perform an actlon that promotes a particular class of itates.
   Promote: Is-Positioned Helix2-1 in PA1 with Strong Constraint.

Table 6. Examples of ACCORD Goals.

1. Achieve a state In which a particular class of events has occurred.
   Until:
   Dld-Restrict Helix2-1 in PA1 with Constraint.

2. Achieve a state in which a particular class of actions is executable.
   Until:
   Can Perform:
   Do-Append Helix2-3 to Helix in PA1 with Constraint.

3. Achieve a state In which a particular class of actions has been executed.
   Did Perform:
   Do-Append Helix2-3 to Helix In PA1 with Constraint.

6.33 Constructing Control Plans

Like a standard **BB1** application system, a **BB*** system can combine various inference methods (e.g., top-down refinement, goal-directed reasoning, opportunistic focus) in its efforts to construct effective control plans. Unlike a standard application system, however, a **BB*** system can exploit the knowledge and expressive power of frameworks. Let us briefly consider two **examples** of generic control knowledge sources available within the **BB*** environment.

One generic control knowledge source, Refine-Parameters, incrementally refines a strategy decision as a sequence of subordinate decisions by replacing specified parameter phrases with legal values, best first. The strategy decision must specify which parameters to replace. For example, the strategy decision in Figure 25 might specify the parameters: pa and target-object. Given this specification, Refine-Parameters generates the first sub-strategy by replacing the phrase, current-best pa, with its highest-rated legal value, pal. It generates that sub-strategy's first subordinate focus decision by replacing the phrase, long constraining secondary-structure, with its highest-rated legal value, **helix3-1.** When PROTEAN has performed actions that satisfy the focus decision's goal. Refine-Parameters generates the sub-strategy's second subordinate focus decision by replacing the phrase, long constraining **secondary-structure,** with its second highest-rated legal value, **helix4-1.** It continues to generate the entire plan shown in Figure **25** in a similar fashion.

Depending on **how** many parameters a strategy specifies, Refine-Parameters can refine a strategy to an arbitrary level of detail. If a strategy specifies all of its parameters, each focus decision will specify the currently most desirable <u>individual</u> action. However, if a strategy specifies a subset of its parameters, as illustrated in the example above, each focus decision will specify the currently most desirable <u>class</u> of actions.

A second generic control knowledge source, Enable-Priority-Action, posts focus decisions favoring actions whose results would trigger strategically desirable actions. For example, suppose that, at some point during the first focus interval in Figure 25, there are no feasible actions that match the focus. That is, there are no feasible actions that match the prescription:

```
        Perform: Quickly do-position helix3-1 in pal
                     with strong constraints.
```

Suppose also, however, that the focus goal has not yet been satified. *In* this kind of situation, Enable-Priority-Action -examines the **concept** network to determine what types of actions would match the focus (e.g., anchor, yoke, and restrict actions). It then posts focus decisions favoring actions that might trigger those action types or satisfy their preconditions. For example, in this case it might post this focus decision:

```
        Promote: Did-position helix4-1 in pal with constraints.
```
to satisfy the precondition of an action for yoking helix3-1 with helix4-1.

6.3.4 Rating Feasible Actions

Like a standard BB1 application system, a BB* system rates alternative feasible actions (KSARs) against all operative focus decisions. However, given the expressive language of a framework, a BB* system can rate KSARs with powerful pattern-matching operations. It rates each parameter value in a KSAR against each corresponding parameter value and modifier in a focus decision. It combines these component ratings according to some integration function (either one specified in that particular focus or a default function) to produce a rating against the entire focus decision. Figure 18 above shows an example in which the KSAR action:

```
        Do-anchor helix3-1 to helix1-1 in pal with NOE27.
```
is rated against the focus decision:

```
        Perform:
          Do-position helix3-1 in pal with strong constraint.
```

63.5 Advantages of Control Reasoning in BB*

The BB* environment offers several important advantages for control reasoning. It provides a superior--concise, perspicuous, uniform, modular, interpretable--representation for control knowledge sources, KSARs, events, and decisions. It provides powerful framework-interpreter procedures for all operations performed on these knowledge structures. It empowers a variety of generic inference methods for control reasoning. Most importantly, it explicitly and unambiguously articulates task-specific control parameters and the relationships among them. Thus, it enforces a semantically correct mapping between the attributes of feasible actions and the control plans against which they are rated.

6.4 Explanation in a BB* System

Like a standard BB1 application system, a BB* system constructs explanations out of information in its evolving control plan. However, a BB* system's explanations can exploit the expressive framework language used to represent its feasible actions and its control plan. For example, Figure 26 shows how PROTEAN explains its decision to execute KSAR55 based on the control plan in Figure 25.

```
  I recommend KSAR55:

  Anchor Helix6-2 to Helix3-2 in PA2 with NOE7.

> Why?

  I recommend KSARSS:

  Anchor Helix6-2 to Helix3-2 in PA2 with NOE7.

  because I am trying to:

  F7: Ouickiy Do-Position Helix6-2 in PA2 with Strong Constraints.

> Why Not73

        NOE7 Is Strong (R60).
        NOE7 is a Constraint (Rl 00).

> Why F7?

  I am attempting to:

  F7: Ouickiy Do-Position Helix6-2 in PA2 with Strong Constraints.

  because I am attempting to:

  S3-2: Quickly Do-Position Long Constraining Secondary-Structure
        in PA2 with Strong Constraints.

> why Helix6-2?

        Helix 6-2 la Long (80).
        Heilx 6-2 is Constraining (75).
        Helix 6-2 is a Secondary-Structure (100).

> Why S3-2?

  I am trying to:

  S3-2: Ouickiy Do-Position Long Constraining Secondary-Structure
        In PA2 with Strong Constrvnt.

  because I am trying to:

  S3: Quickly Do-Position Long Constraining Secondary-Structure
        in Current-Best PA with Strong Constraint.
```

Figure 26. Explanation of the Decision to Perform KSAR55 in BB1-ACCORD.

As this example illustrates, the BB* environment offers several advantages for explanation. Its framework-based language of explanation articulates task-specific control parameters, provides a'structured account of the organization of individual heuristics within a control decision, and communicates the mapping between control heuristics and the KSAR attributes to which they are applied. In addition, a single representation of control decisions serves both as a machine-interpretable object of control reasoning and as a human-interpretable object for use in explaining that reasoning. As a result, we can argue that a BB* system explains its behavior in terms of its own understanding of that behavior.

6.5 Learning in a BB* System

A system's ability to team depends upon several **factors,** including the following: (a) the power of the system's learning procedures: (b) the quality of the data to which it applies those procedures: and (c) the depth and organization of the system's knowledge about relevant concepts. Because BB* offers improvements on each of these factors, it can improve both the efficiency and the accuracy of learning.

Let us consider efforts by the learning knowledge source MARCK to learn the PROTEAN heuristic, Prefer-Anchoring-over-Yoking (see Figure 27). At this point, PROTEAN is operating under the focus:

```
Perform: Do-position long rigid constraining secondary-structure
            in pal with strong constraint.
```

Given this focus, PROTEAN chooses to perform the action of KSAR56:

```
KSAR56: Do-yoke helix6-2 with helix4-2 in pa2 with NOE9.
```

However, the domain expert prefers the action of KSAR55:

```
KSAR55: Do-anchor helix6-2 to helix3-2 in pa2 with NOE8.
```

The domain expert's action triggers MARCK, which tries to identify the **key** difference between the two KSARs and automatically program a corresponding control heuristic.

BB* substantially limits the number of differences between the two KSARs that MARCK must consider. Given the standard BB1 representation, MARCK must search for differences on all KSAR attributes with the same name. Then it must ask the domain expert to choose the key attribute from among all whose values differ. Given BB*'s framework representation, MARCK can focus on corresponding parameters in corresponding patterns in the two KSARs. In this case, there are only four corresponding parameters in the two action patterns and only two of them have different **values: action-keyword and constraint.**

```
  I recommend:
       KSAR56: Yoke Helix6-2 with Helix4-2 In PA2 with NOE9.

> Override with KSARSS.

  I am performing:
       KSARSS: Anchor Helix6-2 to Helix3-2 in PA2 with NOE8.


  I wouid like to understand why you prefer KSARSS over KSAR56:

       KSAR55: Anchor Helix6-2 to Helix3-2 in PA2 with NOES.
       KSAR56: Yoke Helix6-2 with Helix4-2 In PA2 with NOES.

  b It because you prefer:
       (i) Action = Do-Anchor over Action = Do-Yoke?
       (b) Constraint = NOE9 over Constraint = NOE8?

> a

  Do you prefer Action = Do-Anchor over Action = Do-Yoke
       because you wish to:
       Quickly Do-Position Helix6-2 in PA2 with Strong Constraint?

> Yes

  h general, do you prefer to:
       Quickly Do-Position Long Constraining Secondary-Structure
             in Current-Best PA with Strong Constraint?

> Yes

  I am modifying my current plan.
  Shall I modify the knowledge source: Build-PAs?

>  Yes  ,

  I am modifying the knowledge source: Build-PAs.
```

Figure 27. MARCK Learns to Prefer Anchoring Actions over Yoking Actions in the Context of BB1-ACCORD.

BB* also prevents MARCK from making specious comparisons or overlooking valid comparisons. Working with the standard BB1 representation, different or undisciplined system builders often give the same name to unrelated attributes in different knowledge sources or different names to semantically equivalent attributes. In the first case, MARCK must pursue differences in the values of unrelated attributes' as though they were meaningful. In the second case, it will fail to notice differences in the values of semantically equivalent attributes. Finally, if several attributes happen to exhibit a common difference in values, MARCK must ask the domain expert which is the key attribute. Since the domain expert is not a programming expert and ordinarily would not appreciate the actual differences between two attributes having the same values, he or she may choose the wrong one. By contrast, BB*'s use of frameworks focuses MARCK's and the domain expert's attention on key task-specific control parameters by enforcing consistent and semantically valid. naming conventions and explicitly identifying corresponding parameters. As a consequence, MARCK pursues all and only meaningful differences.

BB* also enhances MARCK's ability to identify the heuristic function underlying a domain expert's preference for one value of a parameter over another. MARCK can inspect the knowledge base to determine whether any known modifiers favor the expert's preferred value over the system's preferred value. For example, in PROTEAN quickly is a defined modifier for position, which is the superordinate of anchor and yoke. In Figure 27, MARCK determines that the modifier, quickly, favors anchor over yoke, hypothesizes that this is the key difference between the two KSARs, and asks the domain expert for confirmation. If the modifier, quickly, were not already defined, MARCK would search for the key attribute of the identified parameter and for an appropriate. canonical function, automatically program a new heuristic function, and record it in the knowledge base as the definition of a new modifier for the concept, position.

BB* enables MARCK to introduce a new heuristic at the appropriate level of the control plan. Thus, once MARCK identifies quickly as the key modifier, it can search the control plan for the highest superordinate of its current focus that specifies position or one of its subordinates in as the action keyword. With confirmation from the domain expert, MARCK inserts the new modifier at that level of the plan. If the expert objected, MARCK could work down the plan searching for the appropriate level at which to insert the new modifier.

Finally, BB* obviates MARCK's use of its Lisp-English translator since all of the objects on which it operates are already expressed in the stylized English of frameworks. Thus, MARCK completes its learning simply by inserting the new modifier before the corresponding parameter in its focus decision on the blackboard and in the control knowledge sources that generate that decision.

```
Quickly do-position long rigid constraining
        secondary-structure in pal with
        strong constraint.
```

These advantages apply to other learning procedures as well. For example, we have been working on a set of knowledge sources called WATCH to form inductive generalizations of sequences of executed actions. For example, suppose a domain expert executes the following sequence of actions:

```
Anchor Helix2-1 to Helix1-1 in PA1 with NOE15.
Anchor Helix3-1 to Helix1-1 in PA1 with NOE19.
```

The WATCH knowledge sources can consult the ACCORD conceptual network to determine that:

```
Helix24 Sis-a Helix.
Helix3-1 Sis-a Helix.
Long Helix24 = 90.
Long Helix3-1 = 70.
NOE15 Sis-a NOE.
NOE16 Sis-a NOE.
```

Based on this information, they can hypothesize that the domain expert's current focus is to:

```
Perform:
        Anchor Long Helix to Helix1-1 in PA1 with NOE.
```

In principle, any BB1 system could provide the data required for inductive generalization. In practice, however, such learning ordinarily is not feasible for systems implemented directly in BB1 knowledge structures. Given the unrestricted number of KSAR attributes, the space of possible generalizations is intractably large. Moreover, given an undisciplined approach to attribute naming, the learning data are liable to be extremely noisy. They may support specious generalizations, while entirely concealing valid generalizations. By contrast, a BB* system can exploit a framework such as ACCORD, vastly reducing the space of possible inductions and guaranteeing that generalizations are internally consistent, unambiguous, and semantically valid.

7. Knowledge Engineering within the BB* Environment

The BB* environment facilitates the design and implementation of new applications by providing a general architecture for problem solving and reusable task-specific frameworks. To illustrate this potential, we discuss our experience in building a prototype of the SIGHTPLAN system [51] for designing construction-site layouts within BB1-ACCORD. We then consider the space of domains in which arrangement problems occur and BB1-ACCORD's applicability in different regions of that space.

7.1 Building SIGHTPLAN: A New Application of BBl-ACCORD

7.1.1 SIGHTPLAN's Problem

SIGHTPLAN must arrange pieces of construction equipment (e.g., cranes and trailers) and construction areas (e.g., access roads and lay-down areas) in a two-dimensional construction site to satisfy a variety of constraints. Part-whole relations exist among some of these objects (e.g., the employee-facilities include some trailors and a rest area). Part-whole relations also exist among sub-regions of the construction site (e.g., the building-zone includes the building-site and all of its borders). Available constraints include object-based constraints (e.g., the rest area must be within a short distance of the- trailers) and context-based constraints (e.g., the access road must intersect the perimeter of the construction site on two sides). Since construction projects proceed in identifiable stages, the layout design must include sub-layouts for different stages. Further, there are transitional constraints between the stages (e.g., the crane must move from the northwest corner of the building site to the southeast corner of the building site between stages 1 and 2). (See [52] for a more detailed description of the problem of designing construction-site layouts.)

Despite the obvious dissimilarities between proteins and construction sites, the problem of designing a construction site closely resembles the problem of modeling the construction of a protein. In both cases, the problem-solver must arrange physical objects in a spatial context to satisfy constraints. It must accommodate a variety of constraints, including part-whole relations. objects-based constraints, and context-based constraints. It must design multiple-component solutions for different time intervals and provide legal transitions from each component solution to its successor. In short, both problems are arrangement problems.

On the other hand, SIGHTPLAN's problem is substantially less complex than PROTEAN's problem. STGHTPLAN must deal with tens or hundreds of objects, while PROTEAN must deal with hundreds or thousands of objects. SIGHTPLAN must arrange objects in a two dimensional space, while PROTEAN must arrange objects in a three-dimensional space. SIGHTPLAN must design layouts that incorporate fewer than ten discrete states, while PROTEAN must construct proteins that move through a continuous family of conformations. SIGHTPLAN knows in advance how many stages it must consider and which objects and constraints belong in each state, while PROTEAN must identify protein states and their constituent objects and constraints as part of its reasoning process. SIGHTPLAN must design a small number of satisfactory. site layouts, while PROTEAN must construct the entire family of legal protein structures.

Because of the similarities between SIGHTPLAN's problem and PROTEAN's problem, SIGHTPLAN's principal designers, Iris Tommelein and Ray Levitt, decided to develop it within BB1-ACCORD and we collaborated with them on a prototype system. The following sections discuss how the availability of BB1-ACCORD affected the design and implementation of different aspects of the SIGHTPLAN prototype.

7.13 Choosing a Method

As discussed above, a problem-solving system could, in principle, solve an arrangement problem by any of several different methods. Enumerating and characterizing alternative methods and then choosing and operationalizing an appropriate method for a particular application are time-consuming processes that can determine the success or failure of a system-building effort. For example, it took approximately one person-year of effort to consider alternative methods for PROTEAN and to operationalize a subset of the elements of the chosen assembly method. The PROTEAN staff readily implemented the chosen method within BB1, which was, itself, the product of several person-years of effort.

The very existence of a relevant architecture or framework can facilitate this process by suggesting a candidate method in a clearly operational form. If the architecture or framework already has been applied in other domains, information about thdse applications can facilitate evaluation of the method for the new application. Thus, Tommelein and Levitt quickly recognized the appropriateness of BB1 for SIGHTPLAN. They spent approximately one additional person-month evaluating and deciding to use ACCORD.

7.13 Basic Knowledge Acquisition

Knowledge acquisition requires a conceptual analysis of the knowledge required by an application and a technical analysis of appropriate knowledge representation structures. For example, knowledge acquisition for PROTEAN began with unstructured discussions with domain experts to discover the important domain concepts. The initial **PROTEAN** knowledge base was an unprincipled collection of Lisp functions and data structures, converted to its current declarative form during **a** reimplementation phase. All stages of knowledge acquisition required close collaboration between domain experts and knowledge engineers.

A **framework** can facilitate knowledge acquisition by capturing the conceptual analysis common to a class of applications, identifying appropriate knowledge representation structures, and providing **a** software environment in which to build the **new** knowledge base. For example, **ACCORD** requires domain-specific extensions **of its** conceptual network branches representing objects, contexts, and constraints and specification of low-level functions for anchoring, yoking, appending, etc. Thus, knowledge acquisition for SIGHTPLAN began directly with the **introduction** of particular objects, contexts, and constraints into ACCORD's skeletal concept **network and investigation of alternative approaches to building low-level functions. In addition, domain experts were able to do much of the knowledge acquisition, with modest** amounts of assistance from a knowledge engineer. Of course, since the framework provides much of the actual code necessary to represent the knowledge, there is a substantial reduction in the number of lines of new code generated during knowledge acquisition.

7.1.4 Domain Knowledge Sources

A framework's action hierarchy guides. the design of domain knowledge sources. Basically, the system builder should consider designing one or more knowledge sources to instantiate each terminal action type. The hierarchical classification of action types provides a nice organization of the knowledge sources and the sequence in which to develop them. Further, the knowledge sources developed for previous applications can provide valuable prototypes for new applications.

Without the benefit of ACCORD, the first version of PROTEAN had knowledge sources for anchoring and yoking, which it used to position structures within one complete arrangement. After studying the performance of this system, it **became** apparent that PROTEAN needed a knowledge source for appending and only much later did it become apparent that PROTEAN needed knowledge sources for defining partial arrangements. (PROTEAN still does not have knowledge sources for integrating partial arrangements and coordinating them at multiple levels of abstraction.) Each knowledge source, especially the early ones, required a significant design effort and each successive one had to be coordinated with those developed so far. Since we did not anticipate all contexts in which knowledge sources might interact, we repeatedly modified previously implemented knowledge sources to disambiguate the relationships among them.

By contrast, SIGHTPLAN's current domain knowledge sources are close translations of PROTEAN's domain knowledge sources and were implemented in a matter of days. Although we anticipate that STGHTPLAN and PROTEAN eventually will have many distinct knowledge sources, we expect the translated knowledge sources to endure as the core of the SIGHTPLAN system. If these expectations are borne out, we will extend ACCORD and other frameworks to include, a repertoire of prototype domain knowledge sources and introduce capabilities for automatically instantiating them in new domains.

7.1.5 Control Knowledge Sources

A framework facilitates the development of control knowledge sources in several ways. First, its action, event, and state templates articulate a set of candidate control concepts. Thus, PROTEAN's system builders had to discover key control parameters, such as action class, anchoree, and constraint, and appropriate modifiers, such as quickly, restricted, and strong. By contrast, SIGHTPLAN's sytem builders could begin by considering the formal parameters in ACCORD's action types as candidate control parameters and by considering the high-level concept types and conceptual modifiers in ACCORD's skeletal concept network. Second, as in the case of domain knowledge sources, some control knowledge sources transfer almost directly to applications in new domains. For example, the prototype SIGHTPLAN system uses the basic strategy that PROTEAN uses for small proteins. Of course, SIGHTPLAN introduces some new -modifiers and gives many of the common modifiers new procedural definitions. In addition, we expect to develop more powerful strategies for the two systems that differ more substantially. Again, however, the opportunity to transfer some of the control knowledge permits rapid prototyping of a new application. After we have gained more experience with a range of applications, we plan to develop skeletal control knowledge sources for different subclasses and automatic methods for instantiating them in new domains. Finally, a framework's perspicuous representation makes it easy to articulate and program alternative control strategies. We plan to comparatively evaluate a variety of control strategies for both PROTEAN and SIGHTPLAN.

7.2 The Scope of ACCORD

7.2.1 Arranging **Physical** Objects **in a** Spatial **Context**

ACCORD naturally applies to tasks involving the arrangement of physical objects in a spatial context. PROTEAN and SIGHTPLAN are esoteric examples of such domains. However, consider, for example, the mundane task of furniture arrangement: arrange a specified set of furniture in **a** designated room. We can define each piece of furniture as a physical-object in the **ACCORD** knowledge base and the room as **a'** context. **We** can identify part-whole relationships among furniture groups (e.g., the table-and-chairs includes the table and each of the chairs). We **can** identify part-whole relationships among areas of the room (e.g., the **northern** exposure includes a window area and a fireplace area). **We** can define object-based constraints **on** different pieces of furniture (e.g., each chair must be on a particular side of the table). We can define context-based constraints on the positions of particular pieces of furniture within the room (e.g., put the table near **a** window). Given this representation, we could use the ACCORD actions to define partial furniture arrangements, to position pieces of furniture within each partial arrangement, to refine the positions of furniture groups into the positions of their constituent pieces, and to integrate different partial furniture arrangements to form **a** complete room design.

7.2.2 **Arranging Procedural objects in** a **Temporal context**

We believe that ACCORD also applies to tasks involving the arrangement of procedural objects in a temporal context, For example, consider the **task** of travel planning: arrange a set of destinations in a designated time interval. We can define each destination as **a temporal-**object in the ACCORD knowledge&se and the time interval as a context. We can define **part-**whole relationships among sets of destinations (e.g., the India destination includes destinations: Srinagar, **Agra,** Jaipur, Udaipur, **Benares,** and Darjeeling). We can define part-whole relationships among sub-intervals of the designated time interval (e.g., the spring interval includes May and June). We can define object-based constraints on the relative times targetted for particular destinations (e.g., go to India after Japan). We can define context-based constraints on the absolute times targeted for particular destinations (e.g., go to Japan in time for the cherry blossoms). Given this representation of the knowledge, we probably could use the ACCORD actions to develop partial itineraries, to order destinations within partial itineraries, to refine high-level destinations into more detailed itineraries for their constituent destinations, and to integrate different partial itineraries to form a complete itinerary. We plan to build at least one application of **BB1-ACCORD** involving procedural objects in temporal contexts in order to gain empirical evidence of its applicability to this important subclass of arrangement problems.

7.2.3 Arranging Symbolic Objects in a Symbolic Context

Expanding the potential scope of ACCORD even further, it may be possible to apply it to **tasks involving** the arrangement of general symbolic objects in general symbolic contexts. In particular, **it** may apply to objects and contexts that are not metric in character.

For example, consider **a** simplified project-management **task:** assign a set **of** project tasks among **a** designated set of individuals. We **can** define each task as **a** task-object in the ACCORD knowledge base and the set of individuals as a context We can define part-whole **relationships** among task groups (e.g., the task **of** designing knowledge sources includes tasks for designing domain knowledge sources and designing control knowledge sources). We can define **part-whole** relationships **among** subsets of the individuals (e.g., the expert C programmers are John, Jim, Craig, and Bruce). We can define object-based constraints between different tasks **(e.g.,** the tasks of defining domain and **control** action languages must be performed by the same individual). We can define context-based constraints on the assignments of particular tasks to individuals (e.g., the geometry system must be implemented by expert C programmers). Given this representation, we. might be able to use the ACCORD actions to develop partial project plans, to assign tasks to individuals within partial plans, to refine the assignment of high-level tasks into assignments of their component **tasks,** and to integrate different partial plans to **form** a complete project plan.

Of course, most project-planning tasks also have **a** temporal dimension with associated constraints. Assuming that ACCORD applies to tasks involving the arrangement of procedural objects in **a** temporal context, it might be possible to apply it to the complete project-planning **task:** assign a **set** of project tasks to a designated set of individuals for completion at particular **times.**

8. Open Systems Integration: Multi-Faceted Systems

As discussed in section 1, we require that all modules within a level of the BB* environment satisfy uniform standards of knowledge content and representation. In adhering to this design principle, we aim to achieve *open systems integration* of modules within a level. That is, we aim to support the development of systems that: (a) configure and augment arbitrary sets of existing modules; (b) eliminate redundancy in the contents of those modules: (c) organize the actions enabled by those modules in any appropriate organizational scheme: and (d) superimpose on their reasoning uniform capabilities for control, explanation, and learning.

To illustrate the capability for and utility of open systems integration, consider a new class of *multi-faceted systems.* We define multi-faceted systems with reference to the three-dimensional space of knowledge identified in this paper: knowledge about different problem classes, knowledge about different problem-solving methods, and knowledge about different subject-matter domains. Most contemporary knowledge-based systems occupy a relatively small region of this space: each one knows how to solve a single class of problems by means of a single problem-solving method in a single subject-matter domain. In contrast, multi-facted systems expand their knowledge along one or more dimensions of the space: each one knows how to solve more than one class of problems or how to apply more than one problem-solving method or how to solve problems in more than one domain. Let us consider two hypothetical multi-faceted systems.

First, consider an expert *arrangement assembler-a* system that knows how to apply the assembly method to arrangement problems in each of several subject-matter domains. Figure 28 shows how BB* permits integration of the knowledge in ACCORD, PROTEAN, and SIGHTPLAN to form the arrangement assembler. We would add knowledge about refining prototypes, identifying analogous problems, and measuring different aspects of problem-solving performance. Given this knowledge and some problem-solving experience, the arrangement assembler could, for example: (a) automatically program prototype systems for new application domains; (b) transfer control knowledge among related problem types; and (c) assess the effectiveness of control knowledge for particular problem types. In general, the arrangement assembler could develop increasingly sophisticated arrangement-assembly expertise and apply its expertise to an expanding variety of arrangement problems.

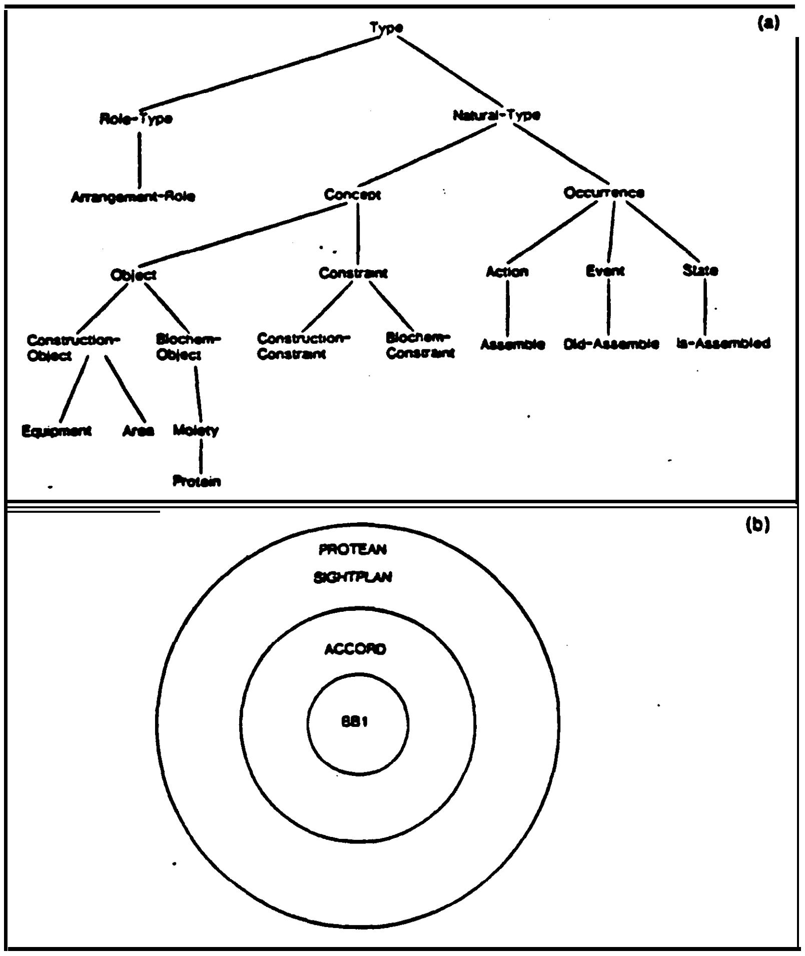


Figure 28. Open Systems Integration in BB*: An Expert Arrangement Assembler. The *arrangement assembler* integrates PROTEAN's biochemistry knowledge and SIGHTPLAN's construction knowledge within a single conceptual network. Similarly, it integrates their combined knowledge sources (not shown here) within the network without redundancy. (PROTEAN and SIGHTPLAN share several knowledge sources that refer only to domain-independent entities (see Figures 21 and 23)). With additional knowledge about refining prototypes, identifying similar problems, and assessing performance, the arrangement assembler could automatically program new applications and transfer strategic knowledge among similar problems.

Now consider an expert *project manager*--that is, a system that knows both how to assemble site plans and how to schedule individual contractors' daily use of a site. Figure 29 shows how BB* permits integration of the knowledge in: (a) ACCORD; (b) STGHTPLAN; (c) ADJUST--a hypothetical framework for planning a sequence of temporally and spatially constrained tasks by means of a prototype-refinement method; and (d) DAYPLAN-a hypothetical application system that would apply ADJUST to the tasks performed on a daily basis by individual contractors. We would give the project manager new knowledge about controlling the combined actions of SIGHTPLAN and DAYPLAN for particular purposes, for example to: (a) design a site plan and then schedule each contractor's daily use of the site; or (b) schedule and evaluate key contractors' daily use of hypothetical site plans during the design process and pursue only hypothesized designs that permit efficient daily use by them. Similarly, the project manager could explain and learn about its integrated actions in terms of the integrated strategy it had adopted. In general, the project manager could combine different kinds of expertise to solve a variety of more complex problems.

As these examples illustrate, BB*'s capability for open systems integration introduces the possibility of incrementally extending the depth and variety of knowledge within a single system to encompass new problem classes, problem-solving methods, and subject-matter domains. At the same time, the underlying knowledge base remains perspicuous, well-structured, and non-redundant Finally, the system continues to employ uniform methods for control, explanation, and learning, thereby presenting a coherent face for the system as a whole.

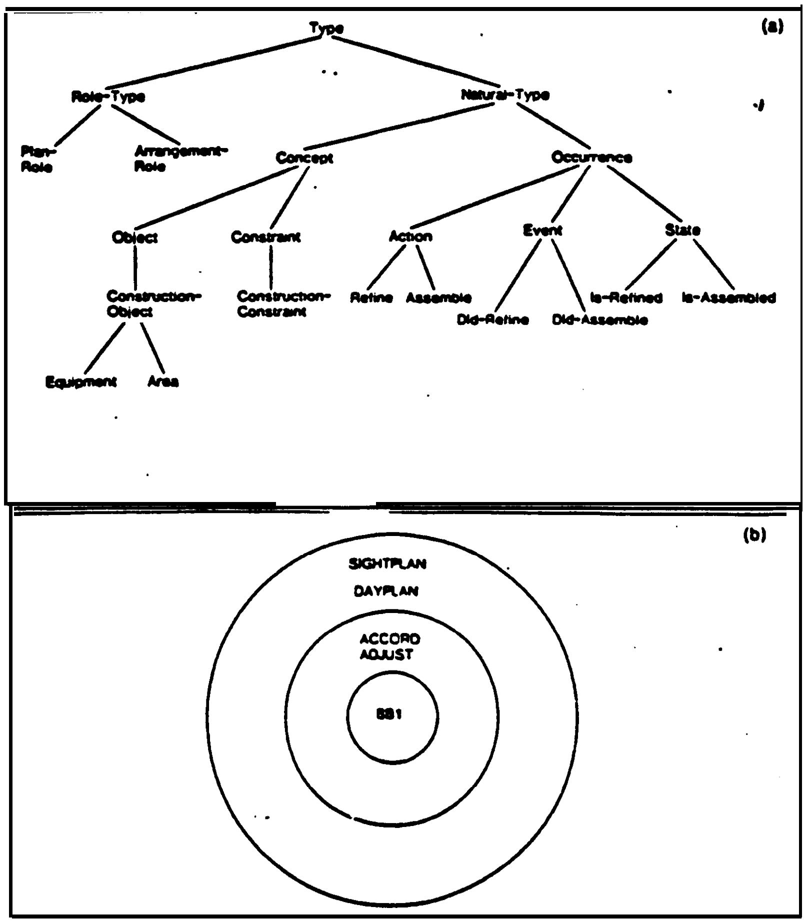


Figure 29. Open **Systems** Integration **in BB*: An Expert Project Manager. The *project manager* integrates the following within** a **single conceptual network: ACCORD's knowledge of the arrangement-assembly task. ADJUST's knowledge of the plan-refinement task, and SIGHTPLAN's and DAYPLAN's combined construction knowledge. Similarly, it incorporates all** of **SIGHTPLAN's and DAYPLAN's knowledge sources (not shown here) within the network. With additional knowledge about combining iu actions for particular purposes, the project manager could solve a variety of more complex problems and explain its efforts to solve those problems. For example, it could: (a) design** a **site plan and then schedule each contractor's daily use** of **the site: or (b) schedule and evaluate key contractors' daily use of hypothetical site plans during the design process and pursue only hypothesized designs that permit efficient daily use by them.**

9. The BB* Environment: Status and Plans

9.1 The BB1 Architecture

9.1.1 Generality and Utility

We put forth the blackboard control architecture, which is implemented as BB1, as a general architecture for intelligent systems. Table 1 (see section 1) briefly describes some of the application systems currently implemented or being implemented in BB1. Most of these applications are being developed by other scientists at Stanford and other research laboratories. In addition, we have shown elsewhere [23] that BB1 provides a natural architecture for the knowledge and control strategies of the Hearsay-II [12] speech-understanding system, the HASP [42] signal-interpretation system, and the OPM [26] task-planning system. The number, variety, and significance of these applications suggest that BB1 provides a generally useful architecture. As we and other scientists develop and classify new applications, we will identify empirical bounds on BB1's generality-and utility.

9.1.2 Control, Explanation, and Learning

In the area of control, BB1 currently has three sets of generic control knowledge sources. One set of knowledge sources refines an application-specific strategy by successively posting the names of control knowledge sources that post its prescribed subordinates. Another set of knowledge sources refines a strategy expressed in framework knowledge structures by successively replacing its parameter phrases with alternative legal values (see in section 6). A third set of knowledge sources posts goal-directed focus decisions that favor KSARs whose actions would enable other high-priority actions [30] (see section 6). All of these generic control knowledge sources can work together, along with application-specific control knowledge sources, to construct fully integrated control plans.

In the area of explanation, BB1 currently provides the graphics-based, menu-driven explanation capabilities discussed in section 2 and illustrated in Figure 26 above. We are investigating extensions of these capabilities to include knowledge-based reasoning about what kinds of explanations might be useful or otherwise appropriate for particular users under particular circumstances.

In the area of learning, BB1 currently provides the MARCK knowledge sources for learning new control heuristics from user intervention (see section 2 and Figure 27). We also have developed the WATCH knowledge sources for drawing inductive generalizations from domain experts' problem-solving actions. We have not yet developed the WATCH knowledge sources that automatically program new control knowledge sources to regenerate inductively acquired strategies during subsequent problem solving episodes. We also are investigating prototype instantiation and learning by analogy as methods for learning how to use general knowledge in **a new** domain and for transferring control knowledge among related applications.

In **add**[illegible]**n** to these new developments, we are conducting experiments to evaluate the cost/benefit tradeoffs of exploiting **BB1's** capabilities for control, explanation, and learning.

9.1.3 Framework-Interpreter and Related Functions

We have implemented **all** framework-interpreter procedures (parse, match, quantify, generate, translate) and incorporated them into the **BB1** scheduler, interpreter, and agenda manager. As mentioned above, the framework-interpreter is entirely independent of ACCORD and can be applied to any user-specified framework specified with the appropriate **BB1** knowledge structures. Moreover, all extensions to **BB1** are designed to accommodate systems **that** freely integrate **BB1** and framework knowledge structures.

In more advanced work, we are investigating a number of strategies that exploit the conceptual **network** for efficiency within framework-interpretation procedures. For example, we plan to exploit the natural discrimination networks entailed in root verb hierarchies for efficient triggering of knowledge sources that share related trigger patterns. As a second example, we plan to exploit the known relations between previous events and the states they promote to restrict the potentially explosive search required to instantiate arbitrary state patterns.

Finally, although the template grammar underlying our framework-interpretation procedures satisfies the requirements of current applications, we anticipate that it will prove too restrictive for later versions of these applications and for new applications. Therefore, we expect to replace it with a more powerful grammar at some time in the future.

9.2 Current and Planned Frameworks

ACCORD is the first framework developed in **BB*.** We have demonstrated ACCORD's applicability in PROTEAN's biochemistry domain and in **SIGHTPLAN's** construction domain. We also plan to investigate its applicability to problems involving procedural objects in temporal contexts and, more generally, to problems involving symbolic objects in symbolic contexts. We continue to extend and refine the knowledge in ACCORD as our understanding of specific applications grows.

We plan to develop new frameworks for several tasks, including: BB1's control, explanation, and learning tasks; and the several tasks--situation assessment, planning, plan monitoring, situation simulation, and plan modification--involved in real-time applications.

In general, as we and other scientists attempt to design new frameworks within BB1 and new applications within particular frameworks, we will increase our understanding of empirical 'bounds on: (a) the availability and utility of knowledge at this level; (b) the range of applicability of individual framework, and (c) the range of frameworks BB1 can accommodate.

9.3 A New Hierarchical Level: Shells

As discussed in section 1, architecture, framework, and application represent three discrete levels on what is probably a continuum of knowledge abstractions. We plan to introduce a fourth level, shells. Each shell will specialize a particular framework by augmenting its task-specific language with prototypical domain and control knowledge sources that are appropriate for a particular subset of tasks.

Like Clancey's Heracles system for heuristic classification [7] and Chandrasekaran's "tools for generic tasks" [5], these shells will articulate useful control strategies for solving particular subclasses of problems. For example, given our experience with SIGHTPLAN, we are building an ACCORD shell that captures a domain-independent form of the knowledge sources PROTEAN uses for small proteins. We believe that they will prove useful in other domains where problems involve a relatively small number of objects and constraints. Similarly, we might develop shells for arrangement-assembly tasks in domains involving physical versus temporal objects or for domains whose contexts involve nominal versus metric dimensions.

Shells will offer an incremental advantage over frameworks in the ease of developing new applications. The system builder has only to instantiate the skeletal branches of the concept network and, perhaps, the prototypical knowledge sources that require domain-specific information. As mentioned above, we are investigating automatic prototype-instantiaticn capabilities to relieve the system builder of the task of instantiating knowledge sources. Of course, the system builder pays for this advantage in loss of flexibility in the reasoning process.

Our shells will differ from systems such as Clancey's and Chandrasekaran's, however, in three ways. First, they will articulate control knowledge. rather than control procedures. As a consequence, a shell may support applications that exploit any of BB1's capabilities for control reasoning, ranging from systems that apply systematic control procedures to those that reason extensively about problem-solving strategy. In addition, they can exploit this knowledge for other purposes. Second, we do not presume that there is a single correct strategy for a given task. Thus, for example, there may exist several shells for arrangement-assembly tasks with different characteristics. Third, our shells will exist in the context of the BB* environment. As a consequence, they can be configured with any other modules from the environment to form more complex, but fully integrated systems, with BB1's general capabilities for control, explanation, and learning superimposed upon them.

10. Major Results

Our major results reinforce and manifest the four themes of the paper (see Figure 1 in section 1):

- that an intelligent system reasons about its actions;
- **that a** system must have knowledge of its actions
- that knowledge should be represented in an abstraction **hierarchy;**
- that knowledge modules within a level should satisfy uniform standards of **content** and representation.

We have developed the **BB1** architecture for systems that reason about their situations, their goals, **and** their actions. **BB1** systems integrate strategic and opportunistic methods to decide which goals to pursue **and** which **actions** to perform. They explain how their actions serve their goals **and** they learn from experience which actions help them to achieve their goals. **BB1** systems **reason in** these several ways by **dynamically** constructing, modifying, executing, explaining, and learning about explicit plans for their own actions in real time.

We have empowered these systems with the generic knowledge in **BB1,** the task-specific **knowledge in** frameworks such as ACCORD, and the more specific knowledge in applications such as PROTEAN. As a consequence, these systems know what facts and states obtain in particular contexts. They **know** what events and states they seek. They know what actions they can perform, what events and states are necessary to enable their actions, and what events and states their actions will produce. They use their knowledge to perform the control, explanation, and learning functions required of them. Since they represent all of these different kinds of **knowledge** explicitly, improving or extending their performance is a matter of improving or extending their knowledge.

We have organized existing modules in the hierarchically layered **BB*** environment: The **BB1** architecture supports multiple frameworks, each of which supports multiple applications. This organization enables us to understand and describe **BB*,** but more importantly, to apply and extend it. We apply **BB*** by building new systems that incorporate and augment existing knowledge modules, possibly exhibiting synergistic effects of independently constructed modules. We extend **BB*** by constructing new knowledge modules, **or** expanding existing modules. Existing high-level modules guide and discipline the 'construction of subordinate modules. Low-level modules substantiate superordinate modules and suggest new opportunities for abstracting superordinate modules. Some of these extensions can be made automatically.

Finally, we have adhered to uniform standards of knowledge content and representation in constructing modules at a given **BB*** level. We offer a single architecture, **BB1,** and its associated frame-based network of knowledge structures for representing actions, events, states, and facts. Frameworks such as ACCORD must specify task-specific knowledge about actions, events, states, and facts within **a** representation combining: a frame-based conceptual network, linguistic templates, partial match tables, and template translations. Applications **such as** PROTEAN must instantiate skeletal branches of the conceptual network and specify knowledge sources that instantiate particular problem-solving actions, events, and states. As a consequence of this within-level uniformity, **BB*** provides open systems integration. We can configure any existing knowledge modules within any appropriate strategic paradigm to attack new problems. Moreover, we can incrementally extend the **knowledge within a** given **system** to encompass additional problem classes, problem-solving methods, or subject-matter domains. At any **stage** in the system's evolution, we can superimpose upon it higher-level generic knowledge **about** control, explanation, and learning to produce a fully integrated and coherent face for the system as a whole.

From an engineering perspective, **BB*** may be viewed as a layered computing environment. **BB1** constitutes a general-purpose "virtual computer" for programs that articulate and reason **about their** own actions. it offers a data representation and instruction set of considerable generality. Frameworks such as ACCORD constitute higher-level programming languages. They provide the more complex data representations and macro operators relevant in narrower, but still significant, sets of programs. Applications such as PROTEAN constitute individual programs developed within the environment They can be programmed in the "machine language" of **BB1** or in the higher-level language of an appropriate framework. Like **higher-level languages** in conventional computing environments, frameworks harness the power of **BB1,** enabling applications . builders to write better programs more easily. **BB*** differs from conventional computing environments in its orientation toward intelligent systems: programs that perform knowledge-intensive reasoning about the problems they solve and about their own problem-solving behavior.

From a scientific perspective, **BB*** may be viewed as an elementary theory of intelligent systems. Like all scientific theory, theories of intelligence carry an inevitable tension between generality and power. Efforts to design encompassing architectures strive for generality: to formulate fundamental laws of artificial intelligence. Efforts to develop task-specific frameworks (or still more specific shells) strive for power. to articulate more constraining laws for a narrower range of intelligent behavior. In both cases, effective application systems confirm predictions of the proposed theory. The BB* environment--in which the **BB1** architecture supports multiple frameworks and each framework supports a range of specific shells and applications--constitutes a theoretical paradigm in which we can realize both generality and power.

References

[1] Allen, J.F.
Towards a general theory of action and time.
Artificial Intelligence 23:123-154, 1984.

[2] Altman, R.
FEATURE
Technical Report, Stanford University Knowledge Systems Laboratory, 1986.

[3] Brinkley, J., Cornelius, C., Altman, R., Hayes-Roth, B., Lichtarge, O., Buchanan, B., and Jardetzky, 0.
Application of constraint satisfaction techniques to the determination of protein tertiary structure.
Technical Report, Stanford Cal: Stanford University, 1986.

[4] Buchanan, B., Hayes-Roth, B., Lichtarge, O., Hewett, M., Altman, R., Rosenbloom, P., and Jardetzky.
Reasoning with symbolic constraints in expert systems.
Technical Report, Stanford, Ca.: Stanford University, 1985.

[5] Chandrasekaran, B.
Generic tasks in knowledge-based reasoning: Characterizing and designing. expert systems at the right' level of abstraction.
Proceedings of the IEEE Computer Society Second International Conference on Artificial Intelligence Applications , 1985.

[6] Clancey, W J.
Acquiring, representing, and evaluating a competence model of diagnostic strategy.
Technical Report HPP-84-2, Stanford, Ca.: Stanford University, 1984.

[7] Clancey, WJ.
Heuristic classif ication.
Artificial Intelligence 27:289-250, *1985.*

[8] Corkill, D.D., Lesser, V.R., and Hudlicka, E.
Unifying data-directed and goal-directed control: An example and experiments.
Proceedings of the AAAI :143-147, 1982.

[9] Davis, R.
Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases.
Technical Report Memo AIM-283, Stanford University Artificial Intelligence Laboratory, 1936.

[10] Dodhiawala, R., and Jagannathan, V.
SADVISOR.
Technical. Report, Boeing Computer Services, 1986.

[11] Duncan, D.
PROCHEM.
Technical Report, Stanford, Ca.: Stanford University, 1986.

[12] Erman, LD., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R.
The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty.
Computing Surveys **12:213-253,** 1980.

[13] Erman, L.D., London, **P.E.,** and Fickas, S.F.
The design and an example use of Hearsay-III.
Proceedings of the Seventh International Joint Conference on Artificial Intelligence **:409-415, 1981.**

[14] Feigenbaum, **E.A.**
The art of artificial intelligence.
Proceedings of the 5th International Joint Conference on Artificial Intelligence **:1014-1029,** 1977.

[15] **Fikes, R.E.,** Hart, **P.E.,** and **Nilsson,** NJ.
Learning and executing generalized robot plans.
Artificial Intelligence **3:251-288,** 1972

[16] Friedland, **P.E.,** and Iwasaki, Y.
The concept and ***implementation*** *of skeletal plans.*
Technical Report, Stanford University, 1983.

[17] Genesereth, **M.R.,** and Smith, D.E.
Meta-level *architecture.*
Technical Report **HPP-81-6,** Stanford, **Ca.:** Stanford University, 1982.

[18] Goos, **G.,** and Hartmanis, J. (Eds.).
Distributed systems - Architecture and implementation.
New York Springer-Verlag, 1981.

[19] Green, Paul E. (Ed.).
Computer network architectures and protocols.
New **York:** Plenum Press, 1982.

[20] Harvey, **J.,** and Hayes-Roth, B.
WATCH: Inductive abstration of control strategies.
1986.

[21] **Hasling,** D.W., Clancey, WJ., and Rennels, G.
Strategic explanations for a diagnostic consultation system.
Technical Report STAN-CS-83-996, Stanford, Ca.: Stanford University, 1983.

[22] Hayes-Roth, B.
BBI: An architecture for blackboard systems that control, explain, and learn about their own behavior.
Technical Report HPP-84-16, Stanford, Ca.: Stanford University, 1984.

[23] **Hayes-Ro**th, B.
A blackboard architecture for control.
Artificial Intelligence Journal **26:251-321,** *1985.*

[24] Hayes-Roth, B., and Hewett, M.
Learning Control Heuristics in a Blackboard Environment.
Technical Report HPP-85-2, Stanford, Ca.: Stanford University, 1985.

[25] Hayes-Roth, B., Buchanan, B Lichtarge, O., Hewett, M, Altman, R., Brinkley, J., Cornelius, C., Duncan, B., Jardetzky, 0.
Elucidating protein structure from constraints in PROTEAN.
Technical Report KSL-85-35, Stanford, Ca.: Stanford University, 1985.

[26] Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., and Cammarata, S.
Modelling planning as an incremental, opportunistic process.
Proceedings of the International Joint Conference on Artificial Intelligence 6:375-383, 197%

[27] Hayes-Roth, F., and Lesser, V.R.
Focus of attention in the Hearsay-IT speech understanding system.
Proceedings of the Fifth International Joint Conference on Artificial Intelligence :27-35, 1977.

[28] Jagannathan, V., Baum, L., and Dodhiawala, R.
KRYPTO.
Technical Report, Boeing Computer Services, 1986.

[29] Jardetzky, O., Lane, A., Lefevre, J-F, Lichtarge, 0, Hayes-Roth, B., and Buchanan, B.
Determination of macromolecular structure and dynamics by NMR.
Proceedings of the NATO Advanced Study Institue: NMR in the Life Sciences , 1985.

[30] Johnson, M.V., and Hayes-Roth, B.
Goat-directed reasoning in BBI.
Technical Report, Stanford, Ca.: Stanford University, 1986.

[31] Kitzmiller, T., and Baum, L
RAPS.
Technical Report, Boeing Computer Services, 1986.

[32] Klahr, D., Langley, P., and Neches, R.t.
Self-modifying production system models of learning and development.
Cambridge, Ma : Bradford Books, 1983.

[33] Lenat, D.B.
EURISKO: A program that learns new heuristics and domain concepts.
Artificial Intelligence 21:61-98, 1983.

[34] MacMillan, S.

Technical Report, FMC Central Engineering Laboratories, 1986.

[35] McCarthy, J.
The advice taker.
In Minsky, M. (editor), *Semantic Information Processing, . Cambridge,* Ma.: MIT Press, 1968.

[36] McDermott, D.
A temporal logic for reasoning about processes and plans.
Cognitive Science 6, 1982.

[37] Miller, G.A., Galanter, E., and Pribram, D.H.
Plans and the structure of behavior.
New York: Holt, Rinehart, and Winston, Inc, 1960.

[38] Mitchell, T., Utgoff, P.E., Nudel, B., and Banerji, R.B.
Learning problem-solving heuristics through practice.
Proceedings of the International Joint Conference on Artificial Intelligence :127-134, 1981. .

[39] Mostow, D J., and Hayes-Roth, F.
Operationalizing heuristics: Some AI methods for assisted AI programming.
Proceedings of the International Joint Conference on Artificial Intelligence , 1979.

[40] Murphy, A., and Dodhiawala, R.
SIMLAB.
Technical Report, Boeing Computer Services, 1986.

[41] Murphy, A., and Jagannathan, V.
PHRED.
Technical Report, Boeing Computer Services, 1986.

[42] Nii, H.P., Feigenbaum, E.A., Anton, J.J., and Rockmore, AJ.
Signal-to-symbol transformation: HASP/SIAP case study.
AI Magazine 3:23-35, 1982.

[43] Pearson, G., and Yao, J.
Mission planning for an autonomous vehicle.
Technical Report, FMC Central Engineering Laboratoris, 1986.

[44] Reddy, R., and Newell, A.
Multiplicative speedup of systems.
In Jones, A. (editor), *Perspectives on Computer Science,* . New York: Academic Press, 1977.

[45] Rosenbloom, P.S., and Newell, A.
Learning by chunking: Summary of a task and a model.
Proceedings of the American Association for Artificial Intelligence :255-258, 1982.

[46] Sacerdoti, E.D.
Planning in a hierarchy of abstraction spaces.
Artificial Intelligence 5:115-135, 1974.

[47] Schulman, R., and Hayes-Roth, B.
Explanation.
Technical Report, Stanford University: Knowledge Systems Laboratory, 1986.

[48] Simon, H.A.
The Sciences of the Artificial.
Cambridge, Ma.: The M.I.T. Press, 1969.

[49] Sowa, J.F.
Conceptual Structures: Information Processing in Mind and Machine.
Reading, Ma.: Addison-Wesley, 1984.

[50] Tanenbaum, A.S.
Computer networks.
Engelwood Cliffs, NJ.: Prentice-Hall, Inc., 1981.

[51] Tommelein, I. D., Johnson, M. V., Hayes-Roth, B., and Levitt, R.E.
SXGHTPLAN- a Blackboard Expert System for the Layout of Temporary Facilities on a Construction Site.
· ***Proceedings of the IFIP WG5.2 Working Conference on Expert Systems in Computer-Aided Design, Sydney, Australia*** , February 1987.

[52] Tommelein, I. D., Levitt, R.E., and Hayes-Roth, B.
Using Expert Systems for the Layout of Temporary Facilities on Construction Sites.
CIB W-65 Symposium, Organization and Management of Construction, Birkshire, U.K , 1987.

[53] Zimmerman, H.
A standard layer model.
In Paul E. Green (editor), ***Computer network architecture and protocols, . New York:*** Plenum Press, 1982.