THE ROLE OF FRAME-BASED REPRESENTATION IN REASONING

A frame-based representation facility contributes to a knowledge system's ability to reason and can assist the system designer in determining strategies for controlling the system's reasoning.

RICHARD FIKES and TOM KEHLER

A fundamental observation arising from work in artificial intelligence (AI) has been that expertise in a task domain requires substantial knowledge about that domain. The effective representation of domain knowledge is therefore generally considered to be the keystone to the success of AI programs [15] (see Figure 1). Domain knowledge typically has many forms, including descriptive definitions of domain-specific terms (e.g., "power plant," "pump," "flow," "pressure"), descriptions of individual domain objects and their relationships to each other (e.g., "P1 is a pump whose pressure is 230 psi"), and criteria for making decisions (e.g., "If the feedwater pump pressure exceeds 400 psi, then close the pump's input value"). Because of this emphasis on representation and domain knowledge, systems that use AI techniques to achieve expertise are often referred to as *knowledge-based systems*, or simply as *knowledge systems.*

In order for a knowledge system to use domain-specific knowledge, it must have a language for representing that knowledge. The basic criteria for a knowledge representation language are the following:

- *Expressive power*—Can experts communicate their knowledge effectively to the system?
- *Understandability*—Can experts understand what the system knows?
- *Accessibility*—Can the system use the information it has been given?

Experience has made it increasingly clear that none of the major knowledge representation languages is by itself able to satisfy all of these criteria. Early attempts at building intelligent systems used the first-order predicate calculus as their representation language (e.g., [10]). The predicate calculus was appealing because of its very general expressive power and well-defined semantics. However, because the language constructs are very fine grained and do not provide adequate facilities for defining more complex constructs, domain experts have difficulty using the predicate calculus or understanding knowledge expressed in it. Also, the generality of the predicate calculus has been a significant barrier to the development of effective deduction facilities for using knowledge expressed in it.

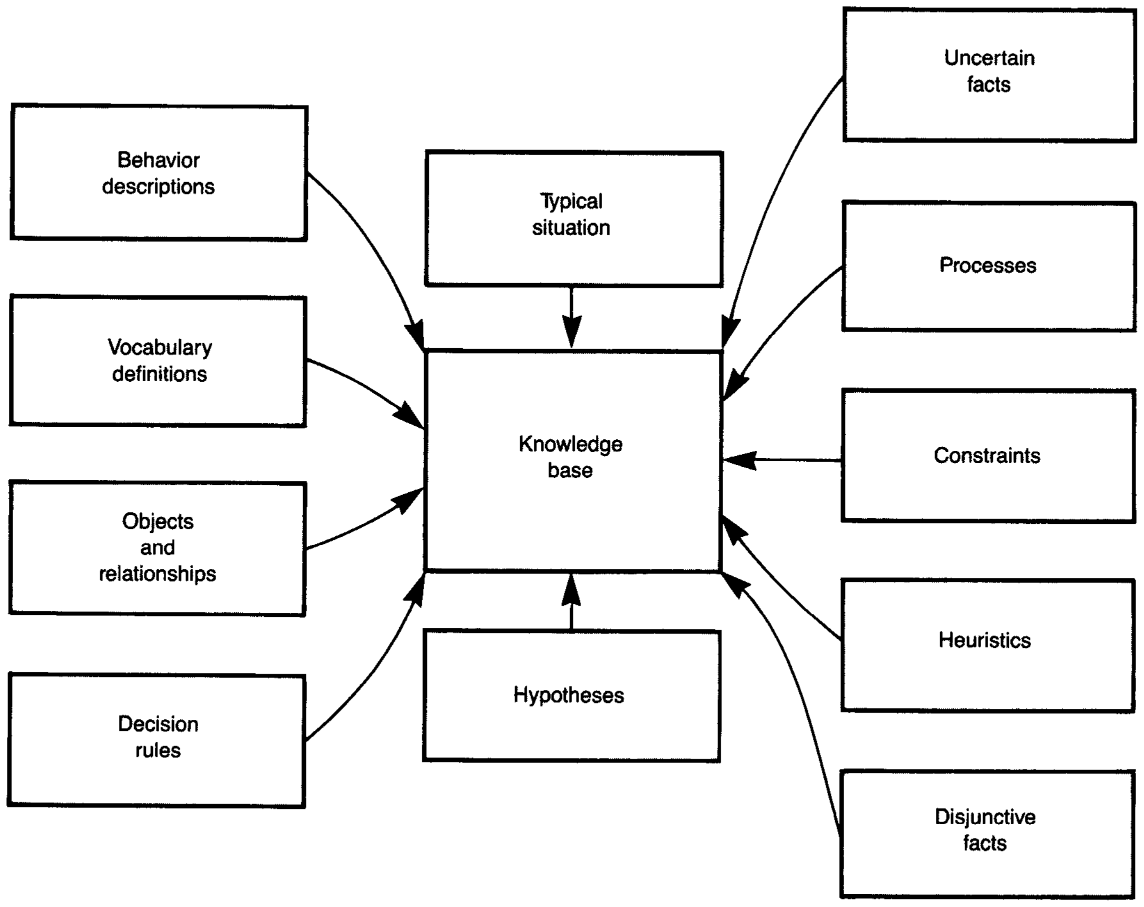
These difficulties helped motivate the development of "semantic networks" (e.g., [11]), and various "object-oriented" representation languages based on *frames* (e.g., [2, 4]). Frame languages provide the knowledge-base builder with an easy means of describing the types of domain objects that the system must model. The description of an object type can contain a prototype description of individual objects of that type; these prototypes can be used to create a default description of an object when its type becomes known in the model.

A frame provides a structured representation of an object or a class of objects. For example, one frame might represent an automobile, and another a whole class of automobiles (see Figure 2). Constructs are available in a frame language for organizing frames that represent classes into taxonomies. These constructs allow a knowledge-base designer to describe each class as a *specialization* (subclass) of other more generic classes. Thus, automobiles can be described as vehicles plus a set of properties that distinguish autos from other kinds of vehicles.

The advantages of frame languages are considerable: They capture the way experts typically think about much of their knowledge, provide a concise structural representation of useful relations, and support a concise definition-by-specialization technique that is easy for most domain experts to use. In addition, special-purpose deduction algorithms have been developed that exploit the structural characteristics of frames to rapidly perform a set of inferences commonly needed in knowledge-system applications.

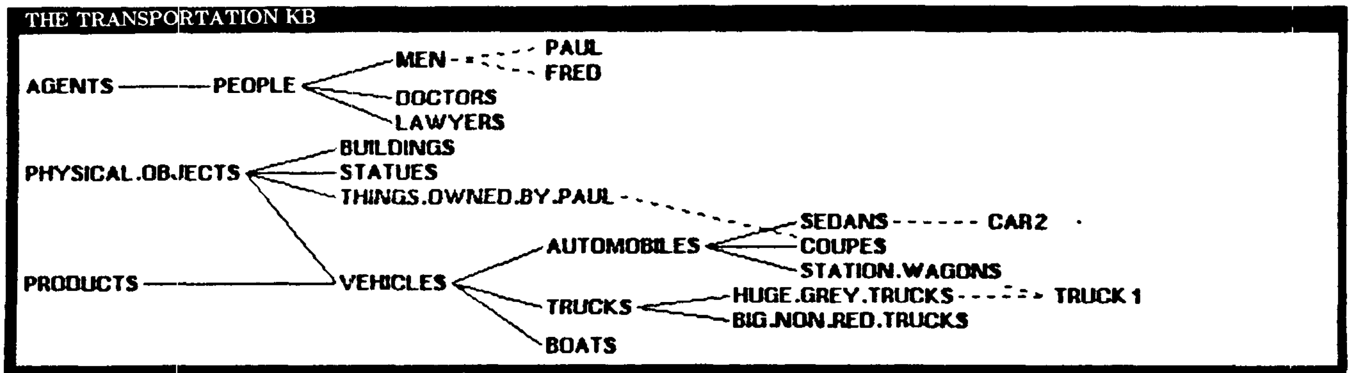
In addition to encoding and storing beliefs about a problem domain, a representation facility typically performs a set of inferences that extends the explicitly held set of beliefs to a larger, virtual set of beliefs. Thus, the representation facility participates in the system's reasoning activities by providing these "automatic" inferences as part of each assertion and retrieval operation. Frame languages are particularly powerful in this regard because the taxonomic relationships among frames enable descriptive information to be shared among multiple frames (via *inheritance*) and because the internal structure of frames enables semantic integrity constraints to be automatically maintained.

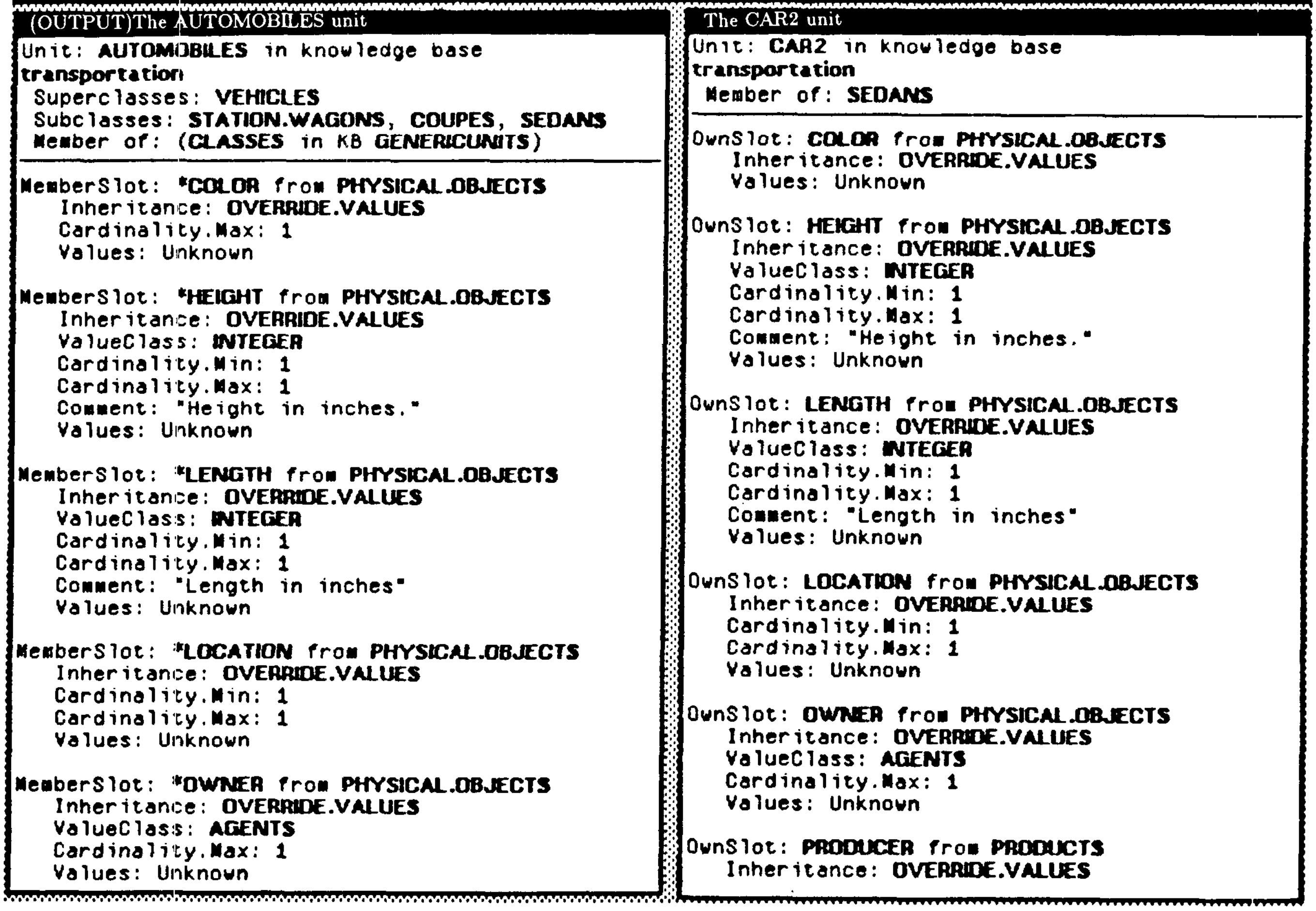
One of the basic tenets of knowledge-system technology is that domain knowledge can be more effectively used by a system and more easily understood by a system's users if it is represented in declarative rather than procedural form. Frame systems, however, provide no direct facilities for declaratively describing how the knowledge stored in frames is to be used. Traditionally, the only way of associating domain-dependent behavior with frames has been by attaching to them in various ways procedures written in the underlying programming language (e.g., LISP) (as in, for example, KL-ONE [4] and KRL [2]). Additional facilities are needed in such systems for declaratively describing domain-dependent inference rules, analysis decision rules, actions that can be taken in the domain by



Expertise in a task domain usually draws on many different kinds of knowledge about that domain. The representation and reasoning facilities in AI systems must be able to integrate different kinds of knowledge into a coherent knowledge base that can effectively support the system's activities.

FIGURE 1. The Kinds of Knowledge That Can Go into a Knowledge Base





(OUTPUT)The AUTOMOBILES unit

```
Unit: AUTOMOBILES in knowledge base
transportation
 Superclasses: VEHICLES
 Subclasses: STATION.WAGONS, COUPES, SEDANS
 Member of: (CLASSES in KB GENERICUNITS)

MemberSlot: *COLOR from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   Cardinality.Max: 1
   Values: Unknown

MemberSlot: *HEIGHT from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: INTEGER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment: "Height in inches."
   Values: Unknown

MemberSlot: *LENGTH from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: INTEGER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment: "Length in inches"
   Values: Unknown

MemberSlot: *LOCATION from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   Cardinality.Min: 1
   Cardinality.Max: 1
   Values: Unknown

MemberSlot: *OWNER from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: AGENTS
   Cardinality.Max: 1
   Values: Unknown
```

The CAR2 unit

```
Unit: CAR2 in knowledge base
transportation
 Member of: SEDANS

OwnSlot: COLOR from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   Values: Unknown

OwnSlot: HEIGHT from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: INTEGER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment: "Height in inches."
   Values: Unknown

OwnSlot: LENGTH from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: INTEGER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment: "Length in inches"
   Values: Unknown

OwnSlot: LOCATION from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   Cardinality.Min: 1
   Cardinality.Max: 1
   Values: Unknown

OwnSlot: OWNER from PHYSICAL.OBJECTS
   Inheritance: OVERRIDE.VALUES
   ValueClass: AGENTS
   Cardinality.Max: 1
   Values: Unknown

OwnSlot: PRODUCER from PRODUCTS
   Inheritance: OVERRIDE.VALUES
```

Frames provide structured representations of objects or classes of objects. The AUTOMOBILES *frame shown here* (lower left) represents the class of all automobiles, and the CAR2 frame (lower right) represents a specific automobile that is a member of that class. Frames allow classes to be described as *specializations of other more generic classes and* for those descriptions to be organized into taxonomies. Thus, automobiles can be described as vehicles plus a set of properties that distinguish autos from other kinds of vehicles. *The transportation taxonomy shown here (top) uses solid* lines to represent class-subclass relationships and dashed lines to represent class-member relationships. For example, VEHICLES is a subclass of both PHYSICAL.OBJECTS and PRODUCTS*, and* TRUCK1 *is a member of both* HUGE.GREY.TRUCKS and THINGS.OWNED.BY.PAUL.

FIGURE 2. A Frame Taxonomy

various agents, simulations of object behavior, etc.

The most popular and effective representational form for declarative descriptions of domain-dependent behavioral knowledge in knowledge systems has been pattern/action decision rules, called *production rules* (e.g., [6, 7]). Production rules are, in effect, a subset of the predicate calculus with an added prescriptive component indicating how the information in the rules is to be used during reasoning. Production rules can be easily understood by domain experts and have sufficient expressive power to represent a useful range of domain-dependent inference rules and behavior specifications. By themselves, however, production rules do not provide an effective representation facility for most knowledge-system applications. In particular, their expressive power is inadequate for defining terms and for describing domain objects and static relationships among objects.

The major inadequacies of production rules are in areas that are effectively handled by frames. A great deal of success, in fact, has been achieved by integrating frame and production rule languages to form hybrid representation facilities that combine the advantages of both component representation techniques (e.g., LOOPS® [18], KEE® (Knowledge Engineering Environment®) [12], and CENTAUR [1]). These systems have shown how a frame language can serve as a powerful foundation for a rule language. The frames provide a rich structural language for describing the objects referred to in the rules and a supporting layer of generic deductive capability about those objects that does not need to be explicitly dealt with in the rules. Frame taxonomies can also be used to partition, index, and organize a system's production rules. This capability makes it easier for the domain expert to construct and understand rules, and for the system designer to control when and for what purpose particular collections of rules are used by the system.

Although a primary motivation for Minsky's introduction of frames [4] was to semantically direct the reasoning of scene-analysis systems, most of the subsequent work on frame-based systems (e.g., KRL [2], UNITS [17], and KL-ONE [4]) has focused on structural representation issues rather than on the control of reasoning. The information stored in frames has often been treated as the "database" of the knowledge system, whereas the control of reasoning has been left to other parts of the system. This focus on structural representation issues has helped to elucidate the semantics of the common frame constructs and to demonstrate their usefulness for organizing and storing knowledge (e.g., [5]). Little attention, however, has been paid to whether and how those constructs can be useful for controlling reasoning.

Recent experience with frame-based representation facilities in complex application domains has shown that frames can play an important role throughout the system, including in the control of reasoning components. For example, the structural features of frame languages have proved to be very useful for organizing and controlling the behavior of large collections of production rules. These uses of frames are our central theme in this article. We elaborate the various ways in which a frame-based representation facility participates in a knowledge system's reasoning functionality and can assist the system designer in determining strategies for controlling a system's reasoning.

LOOPS is a trademark of Xerox Corporation.
KEE and Knowledge Engineering Environment are trademarks of IntelliCorp.

COMPONENTS OF A FRAME-BASED REPRESENTATION FACILITY

In this section we summarize the basic components of a typical frame-based representation facility in order to indicate the salient features of frame systems and to provide a context for the discussions in subsequent sections. The facility described is a component of the KEE system [12]. In order to highlight the role that a frame-based representation facility plays in the reasoning of a knowledge system, our description explicitly distinguishes between the semantic interpretation of frame language constructs (e.g., that a `MemberOf` link between frames M and C denotes the proposition that the object represented by M is a member of the class represented by C), and the reasoning services that are typically provided by a frame-based representation facility (e.g., that when a `MemberOf` link is created between frames M and C, the default description in C of members of the class represented by C is added to M).

Structural Features

Taxonomy Descriptions. The frame-based representation language included in the KEE system provides typical frame language constructs for describing individuals and classes of individuals in an application domain (see Figure 3). Each individual or class is represented by a frame.[1] Frames can be organized into taxonomies using two constructs that represent relationships between frames: *member links*, representing class membership, and *subclass links*, representing class containment or specialization. These links provide two standard interpretations of the meaning of "is–a" links, as in "A truck *is a* vehicle" and "TRUCK1 *is a* truck." (See [3] for a discussion of the variety of interpretations of "is–a" in frame systems.)

Frames can incorporate sets of attribute descriptions called *slots*. A distinguishing characteristic of frame-based languages is that a frame representing a class can contain prototype descriptions of members of the class as well as descriptions of the class as a whole. In the KEE system, prototype descriptions are distinguished from other descriptive information by the use of two kinds of slots, *own slots* and *member slots*. Own slots can occur in any frame and are used to describe attributes of the object or class represented by the frame. Member

[1] Some systems use other terms for what we are calling frames. For example, frames are called *units* in the KEE system and *concepts* in KL-ONE. We use the single generic term "frame" in all cases here for consistency.

```
--------------------------------------------------------

Frame: TRUCKS in knowledge base TRANSPORTATION
   Superclasses: VEHICLES
   Subclasses: BIG.NON.RED.TRUCKS, HUGE.GREY.TRUCKS
   MemberOf: CLASSES.OF.PHYSICAL.OBJECTS

--------------------------------------------------------
        .
        .
        .

MemberSlot: HEIGHT from PHYSICAL.OBJECTS
   ValueClass: INTEGER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Units: INCHES
   Comment:  "Height in inches."
   Values: Unknown

MemberSlot: LENGTH from PHYSICAL.OBJECTS
   ValueClass: NUMBER
   Cardinality.Min: 1
   Cardinality.Max: 1
   Units: METERS
   Comment:  "Length in meters"
   Values: Unknown
        .
        .
        .

OwnSlot: LONGEST from CLASS.OF.PHYSICAL.OBJECTS
   ValueClass: TRUCKS
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment:  "The longest known truck"
   Values: Unknown

OwnSlot: TALLEST from CLASS.OF.PHYSICAL.OBJECTS
   ValueClass: TRUCKS
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment:  "The tallest known truck"
   Values: Unknown
        .
        .
        .

--------------------------------------------------------
```

This frame describes class TRUCKS as a subclass of class VEHICLES and as a member of class CLASSES.OF.PHYSICAL.OBJECTS. Member slots in the TRUCKS frame like LENGTH and HEIGHT provide a prototype description of each class member. Own slots like LONGEST and TALLEST describe attributes of the class as a whole.

FIGURE 3. The TRUCKS Frame

slots can occur in frames that represent classes and are used to describe attributes of each member of the class, rather than of the class itself. For example, a frame representing the TRUCKS class might have own slots for LONGEST and HEAVIEST, and member slots for LENGTH and WEIGHT. Member slots allow class frames to play a role in knowledge bases similar to that of schemas in relational databases.

Frames representing classes may have slots whose values specify collections of subclasses that form disjoint decompositions or exhaustive decompositions of the class (e.g., to specify that a vehicle cannot be both a truck and a station wagon, or that all adults are either men or women). The semantics of these decomposition slots is considered to be part of the definition of the frame language. Thus, domain-independent methods can be included in a frame system for reasoning about decompositions.

Attribute Descriptions. An important source of the expressive power of frame-based languages is the facilities they provide for describing object attributes. For example, a frame representing a truck might include descriptions of the truck's height, length, and owner. These facilities allow frames to include partial descriptions of attribute values, and help preserve the semantic integrity of a system's knowledge base by constraining the number and range of allowable attribute values.

Slots in most frame systems, including KEE, can have multiple values (to provide appropriate support for attributes such as COUSIN, WHEEL, and VICEPRESIDENT) and a set of properties, which we are calling *facets.* Several frame systems, including KEE, have built-in facets for representing constraints on the number of possible values an attribute can have and for indicating the classes to which each value must belong. For example, the frame representing a person "John" could specify that John's SISTER slot has three values, each of whom is a doctor, without identifying the sisters.

In the KEE system, two facets, CardinalityMin and CardinalityMax, have been provided for constraining the number of values for an attribute represented by a slot. A CardinalityMin value of *m* indicates that the corresponding attribute has at least *m* distinct values; a CardinalityMax value of *n* indicates that the corresponding attribute has at most *n* distinct values.

The ValueClass facet of a slot can be used to describe the classes to which each value of the slot belongs. The value of the ValueClass facet of a slot can be a Boolean combination of class descriptions; for instance,

```
(INTERSECTION MEN
   (UNION DOCTORS LAWYERS)
   (NOT.ONE.OF FRED))
```

designates a man who is either a doctor or a lawyer, but is not Fred (see Figure 4). The value-class specification is a generalization of a standard programming language data-type specification. The KEE system provides a knowledge base containing class frames for standard data types (e.g., INTEGERS, STRINGS). The class frames in that knowledge base are available for inclusion in value-class specifications. For example, one could specify that a value must be any integer in the range 0 to 100 except 23 or 36 with the value class

```
(INTERSECTION INTEGERS
   (INTERVAL 0 100)
   (NOT.ONE.OF 23 36)).
```

The system's functions for adding slot values to a knowledge base use the slot's value-class and cardinality specifications as constraints that must be satisfied by any new value. Value-class and cardinality specifications also provide effective partial descriptions of unknown slot values, including the representation of a useful class of disjunctions and negations. For example, the UNION and ONE.OF value-class constructs can be used to express disjunctive information about the values of a slot, the NOT.IN and NOT.ONE.OF constructs can be used to express negative information, and a CardinalityMax value of 0 can be used to indicate that the slot has no values.

Behavioral Properties

Although frame languages provide no specific facilities for declaratively describing behavior, they do provide various ways of attaching procedural information expressed in some other language (e.g., LISP) to frames (see Figure 5). This procedural attachment capability enables behavioral models of objects and expertise in an application domain to be built. It also provides a powerful form of object-oriented programming whereby objects represented by frames can respond to messages.

The KEE system supports two standard forms of procedural attachment: *methods* and *active values.* Methods are LISP procedures, attached to frames, that respond to messages sent to the frames. Methods are stored as the values of slots that have been identified as *message responders.* Messages sent to frames specify the target message-responder slot and include any arguments needed by the method stored at the slot. Active values are procedures or collections of production rules attached to slots that are invoked when the slot's values are accessed or stored. Thus, they behave like "demons," monitoring changes and uses of the values. They can also be used to dynamically compute values on a "when-needed" basis. Methods and active values are typically written to apply to any member of a class of objects and are included by the knowledge-base designer in the class description as part of the prototype description of a class member.

Reasoning Services

A frame-based representation facility extends the system's explicitly held set of beliefs to a larger, virtual set of beliefs by automatically performing a set of inferences as part of its assertion and retrieval operations. These inferences, based on the structural properties of

```
--------------------------------------------------------

Unit: TRUCK1 in knowledge base TRANSPORTATION
   Member: THINGS.OWNED.BY.PAUL, HUGE.GREY.TRUCKS

--------------------------------------------------------

         .
         .
         .

OwnSlot: OWNER
   ValueClass: MEN (UNION DOCTORS LAWYERS)
               (NOT.ONE.OF FRED)
               AGENTS
   Cardinality.Max: 1
   Values: PAUL
         .
         .
         .

OwnSlot: WHEELS from HUGE.GREY.TRUCKS
   Cardinality.Min: 16
   Comment: "The vehicle's wheels"
   Values: Unknown

--------------------------------------------------------
```

The facilities provided by frame languages for describing object attributes help preserve the semantic integrity of a system's knowledge base by constraining the number and range of allowable attribute values. For example, the frame shown here, which represents a truck, specifies (by means of the `Cardinality.Min` facet of the `WHEEL` slot) that the truck must have at least 16 wheels and (by means of the `ValueClass` facet of the `OWNER` slot) that its owner must be a man who is either a doctor or a lawyer and not Fred.

FIGURE 4. The `TRUCK1` Frame

frames and taxonomies, can often play a major role in the overall reasoning of a knowledge system. Because they are "wired in" to the representational machinery and have a limited scope, they are much faster than general deduction methods, such as logic theorem provers or production rule interpreters.

Some of these inference methods perform what is commonly known as *inheritance.* If the `PHYSICAL.OBJECTS` frame has a subclass link to the `VEHICLES` frame, for example, and the `VEHICLES` frame has a subclass link to the `AUTOS` frame, the representation facility will "retrieve" the belief that `AUTOS` is a subclass of `PHYSICAL.OBJECTS` without recourse to other reasoning mechanisms.

Other automatic inference methods use constraints such as value-class and cardinality specifications to determine whether a given item could be a value of a given slot. For example, when a value is being added to a slot, the value is rejected if the slot already contains the maximum number of permitted values or if the value is not a member of the slot's value class.

Inheritance. The assertion and retrieval mechanisms for frame-based languages use the member links, subclass links, and prototype descriptions of class members to augment the descriptive information in a frame. Any frame can have a member link to one or more class frames (e.g., to represent that `TRUCK1` is a member of both `TRUCKS` and `THINGS.OWNED.BY.PAUL`). A frame is said to *inherit* the member slots of the class frames to which it has member links. Those inherited slots become own slots of the member frame, since they represent attributes of the member object itself. For example, the `TRUCK1` frame would acquire two own slots, `LENGTH` and `WEIGHT`, from the frame for `TRUCKS`.

Class frames (i.e., frames that represent classes) can also have subclass links to one or more other class frames (e.g., to represent that `TRUCKS` is a subclass of both `VEHICLES` and `PRODUCTS`). Since every member of a subclass is also a member of the superclass, a subclass frame inherits the member slots of its superclass frames as additional member slots. For example, if the `PHYSICAL.OBJECTS` frame has member slots `LENGTH` and `WEIGHT`, and the `VEHICLES` frame has a subclass link to the `PHYSICAL.OBJECTS` frame, then `LENGTH` and `WEIGHT` become member slots of the `VEHICLES` frame as well.

The KEE system also considers a class to be a describable object. Thus, the frame for a class can indicate the classes to which that class itself belongs. Like any other frame, a class frame inherits own slots from the frames that represent the "classes of classes" to which the class belongs. For example, VEHICLES might be a member of class PHYSICAL.OBJECT.TYPES. The PHYSICAL.OBJECT.TYPES frame might include the member slots LONGEST and HEAVIEST, which would thus become own slots of the VEHICLES frame.

Value Class and Cardinality Reasoning. A frame system considers value-class and cardinality specifications as constraints on the legal values of a slot. The system provides constraint checking procedures for determining whether a slot's value-class and cardinality specifications exclude a given item from being a value of the slot. An item is excluded if the slot already has its maximum number of allowable values or if the item is not a member of the slot's value class. These procedures can be called directly by the user. They are called by the system whenever a slot's values, value-class specifications, or cardinality specifications are changed. Calls by the system cause an error to be generated if a constraint is violated.

The value-class constraint checking procedures understand the semantics of basic set theory operators and numerical intervals. The primitive test of whether a given item is in a class represented by a frame is performed by sending a message to the frame; thus each

```
---------------------------------------------------------

Unit: TRUCKS in knowledge base TRANSPORTATION
   Superclasses: VEHICLES
   Subclasses: BIG.NON.RED.TRUCKS, HUGE.GREY.TRUCKS
   Member: CLASSES.OF.PHYSICAL.OBJECTS

---------------------------------------------------------
        .
        .
        .

MemberSlot: DIAGNOSE from TRUCKS
   Inheritance: METHOD
   ValueClass: METHODS
   Cardinality.Min: 1
   Cardinality.Max: 1
   Comment: "A method for diagnosing electrical faults:
   Values: TRUCK.DIAGNOSIS.FUNCTION

MemberSlot: ELECTRICAL.FAULTS from TRUCKS
   Comment: "Faults found by the DIAGNOSIS method"
   Values: Unknown

MemberSlot: LOCATION from PHYSICAL.OBJECTS
   Cardinality.Min: 1
   Cardinality.Max: 1
   Values: Unknown
   ActiveValues: UPDATE.LOCATION
        .
        .
        .

---------------------------------------------------------
```

Procedural information can be attached to frames in various ways. For example, the value of the DIAGNOSE slot in the TRUCKS frame is a method (i.e., function) for diagnosing electrical faults. The slot and method are inherited by the frames that represent individual trucks and enable each of them to respond to DIAGNOSE messages by calling the method. In addition, "demons" in the form of functions or collections of production rules can be attached to slots so that they are automatically invoked when the slot's values are accessed or stored. For example, a demon is attached to the LOCATION slot of the TRUCKS frame (by means of the ActiveValues facet) to update a geographical map being displayed by the system whenever the slot's value changes. The slot and its attached demon are inherited by the frames that represent individual trucks, so that the current location of every truck is displayed on the map.

FIGURE 5. Procedural Information in the TRUCKS Frame

class in a knowledge base can have its own membership test (e.g., for INTEGERS). The frame can respond yes, no, or unknown. A default method is supplied that looks at explicit membership links and decomposition specifications.

FRAMES AS A FOUNDATION FOR PRODUCTION-RULE SYSTEMS

A frame-based representation facility can serve as an important component in the design of a production-rule language and the reasoning facilities that interpret rules. The frame facility supplies an expressively powerful language for describing the objects being reasoned about by the rules and automatically performs a useful set of inferences on those descriptions. In addition, frames can be used to represent the rules themselves. When each rule is represented as a frame, rules can easily be grouped into classes, and the description of a rule can include arbitrary attributes of the rule. For example, a frame representing a rule could have an EXTERNAL.FORM slot containing the rule as the user wrote it, and a PARSE method for converting the rule into an internal form consisting of lists of expressions that are values of the slots CONDITIONS, CONCLUSIONS, and ACTIONS. Other slots that provide descriptions, such as rationalizations for the rule, records of usage, and goals the rule is useful for achieving, could be included in the frame at the user's or designer's discretion.

The architecture of the production-rule facility in the KEE system illustrates many of these points. Each rule is represented as a frame, and the facility uses a simple predicate logic language for representing a rule's conditions and conclusions. The predicates of the language reflect the relationships that can be represented in the frame language; for instance, class membership (IN.CLASS), the minimum cardinality of an own slot (OWN.MIN.CARD), and being the value of an own slot (OWN.VALUE). The rule designer therefore has full access to the frame language through these predicates. In addition, the language allows any LISP function to be a predicate, so that an arbitrary computation can be used to determine the truth value of a rule condition.

Consider, for example, the KEE system production rule shown in Figure 6. This rule states that trucks weighing more than 10,000 pounds, having at least 10 wheels, and having a color other than red are members of the class BIG.NON.RED.TRUCKS. The rule is represented in the KEE system as a frame and is parsed by a method attached to the frame. The parser translates the rule's conditions and conclusions into the logic language described above. The rule's internal form corresponds to

```
(IF (AND (IN.CLASS ?X TRUCKS)
         (OWN.VALUE WEIGHT ?X ?WT)
         (GREATERP ?WT 10000)
         (OWN.MIN.CARD WHEELS ?X 10)
         (NOT (OWN.VALUE COLOR ?X RED)))
 THEN (IN.CLASS ?X BIG.NON.RED.TRUCKS).
```

Suppose that a knowledge base is constructed as follows: PHYSICAL.OBJECTS is defined as a class of objects each of which has a color attribute with at most one value (Figure 7a); TRUCKS is defined as a subclass of physical objects (Figure 7b); HUGE.GREY.TRUCKS is defined as a subclass of TRUCKS the members of which have color grey and at least 16 wheels (Figure 7c); and TRUCK1 is defined as a huge grey truck weighing 15,000 pounds (Figure 7d). Note that the TRUCKS frame inherits all the member slots of the PHYSICAL.OBJECTS frame, that the HUGE.GREY.TRUCKS frame inherits all the member slots of the TRUCKS frame, and that the TRUCK1 frame inherits all the member slots of the HUGE.GREY.TRUCKS frame as own slots.

Given this knowledge base, consider using the example production rule to show that (IN.CLASS TRUCK1 BIG.NON.RED.TRUCKS), that is, that TRUCK1 is a big nonred truck. The rule interpreter queries the knowledge base about each condition of the instantiated rule in turn, and the queries are processed by the frame representation facility.

The first condition, (IN.CLASS TRUCK1 TRUCKS), is retrieved by the frame system as being true, even though there is no explicit class membership link between TRUCK1 and TRUCKS. The second condition, (OWN.VALUE WEIGHT TRUCK1 ?WT), involves a simple slot value lookup and bounds the variable WT to a limit of 15,000. The third condition, (GREATERP ?WT 10000), can then be evaluated by calling the LISP function GREATERP. The fourth condition, (OWN.MIN.CARD WHEELS TRUCK1 10), is inferred to be true by the frame system by means of both an inheritance to obtain the MIN.CARD restriction on TRUCK1's WHEELS slot and the deduction that if a slot has at least 16 values then it also has at least 10 values. Finally, the last condition of the rule, (NOT (OWN.VALUE COLOR TRUCK1 RED)), follows from the inherited MAX.CARD of 1 and VALUE of GREY for TRUCK's COLOR slot. When the rule is applied, (IN.CLASS TRUCK1 BIG.NON.RED.TRUCKS) is asserted, causing a member link to be created between TRUCK1 and BIG.NON.RED.TRUCKS. The creation of this link causes the TRUCK1 frame to inherit the values and facets of the member slots of the BIG.NON.RED. TRUCKS frame.

Application of this rule to conclude that TRUCK1 is a big nonred truck involves reasoning about subclass relationships, cardinality constraints, and inherited slot values, all of which is done by the frame system. Thus, we see that a significant portion of the reasoning involved in reaching this conclusion is done automatically by the frame system in support of the one domain-dependent rule supplied by the user.

USING FRAMES TO MANAGE RULE-BASED REASONING

As the number of production rules in a knowledge system grows, it becomes more difficult for a system designer to understand the interactions among the rules, to debug them, and to control their behavior. Production rules, like conventional programs, need to be organized into small, easily managed modules. A

```
-----------------------------------------------------------

Unit: BIG.NON.RED.TRUCKS.RULE in knowledge base TRANSPORTATION
   Member: TRUCK.CLASSIFICATION.RULES

-----------------------------------------------------------

OwnSlot: ACTION from RULES
   Inheritance: UNION
   Values: Unknown

OwnSlot: ASSERTION from BIG.NON.RED.TRUCKS.RULE
   Inheritance: UNION
   ActiveValues: WFFINDEX
   Values: |Wff:(?X IS IN CLASS BIG.NON.RED.TRUCKS)

OwnSlot: EXTERNAL.FORM from BIG.NON.RED.TRUCKS.RULE
   Inheritance: SAME
   ValueClass: LIST
   ActiveValues: RULEPARSE
   Values: (IF ((?X IS IN CLASS TRUCKS)
                AND
                (GREATERP (THE WEIGHT OF ?X)
                          10000)
                AND
                (?X HAS AT LEAST 10 WHEELS)
                AND
                (NOT (THE COLOR OF ?X IS RED)))
                THEN
                (?X IS IN CLASS BIG.NON.RED.TRUCKS))

OwnSlot: PARSE from RULES
   Inheritance: METHOD
   ValueClass: METHODS
   Values: DEFAULT.RULE.PARSER

OwnSlot: PREMISE from BIG.NON.RED.TRUCKS.RULE
   Inheritance: UNION
   ActiveValues: WFFINDEX
   Values: |Wff:(?X IS IN CLASS TRUCKS)
           |Wff:(THE WEIGHT OF ?X IS ?VAR29)
           |Wff:(GREATERP ?VAR29 10000)
           |Wff:(?X HAS AT LEAST 10 WHEELS)
           |Wff:(NOT (THE COLOR OF ?X IS RED))

        .
        .
        .
-----------------------------------------------------------
```

Using frames to represent production rules allows rules to be grouped into classes and to include additional descriptive information as frame slots. For example, the frame shown here represents a rule for identifying "big nonred trucks." The frame has an EXTERNAL.FORM slot that contains the rule as the user wrote it, and a PARSE method that converts the rule into an internal form consisting of lists of expressions that are values of the PREMISE, ASSERTION, and ACTION slots.

FIGURE 6. A Production Rule Represented by a Frame

```
--------------------------------------------------------

Frame: PHYSICAL.OBJECTS
   Superclasses: ...
   Subclasses: ...
   MemberOf: ...

--------------------------------------------------------
        .
        .
        .

MemberSlot: COLOR
   Valueclass: ...
   Cardinality.Min: 1
   Cardinality.Max: 1
        .
        .
        .

--------------------------------------------------------
```

(a) PHYSICAL OBJECTS is a class of objects having at most one color each.

```
--------------------------------------------------------

Frame: TRUCKS
   Superclasses: PHYSICAL.OBJECTS
   Subclasses: ...
   MemberOf: ...

--------------------------------------------------------
        .
        .
        .

MemberSlot: WHEELS
   Valueclass: ...
   Cardinality.Min: 4
        .
        .
        .

--------------------------------------------------------
```

(b) TRUCKS is a subclass of PHYSICAL OBJECTS.

A frame-based representation facility extends a system's explicitly held set of beliefs to a larger, virtual set of beliefs by automatically performing a set of inferences as part of its assertion and retrieval operations. These inferences, which are based on the structural properties of frames and taxonomies, can often play a major role in the overall reasoning of a knowledge system. For example, the

FIGURE 7. A Deductive Retrieval

```
-------------------------------------------------------

Frame: HUGH.GREY.TRUCKS
   Superclasses: TRUCKS
   Subclasses: ...
   MemberOf: ...

-------------------------------------------------------
          .
          .
          .

MemberSlot: COLOR from PHYSICAL.OBJECTS
   ValueClass: ...
   Cardinality.Min: ...
   Cardinality.Max: ...
   Values: GREY

MemberSlot: WHEELS from TRUCKS
   ValueClass: ...
   Cardinality.Min: 16
          .
          .
          .

-------------------------------------------------------
```

(c) HUGE.GREY.TRUCKS is a subclass of TRUCKS, the members of which have color grey and at least 16 wheels.

```
-------------------------------------------------------

Frame: TRUCK1
   MemberOf: HUGE.GREY.TRUCKS

-------------------------------------------------------
          .
          .
          .

OwnSlot: WEIGHT
   Values: 15,000
          .
          .
          .

-------------------------------------------------------
```

(d) TRUCK1 is a huge grey truck weighing 15,000 pounds.

information in the four frames shown here would be used in applying the production rule shown in Figure 6 to conclude that TRUCK1 is a big nonred truck. The virtual beliefs derived by the frame system's retrieval facilities during the rule application include the belief that TRUCK1 is a truck, that it has at least 10 wheels, and that its color is not red.

FIGURE 7. A Deductive Retrieval

frame-based representation facility can provide significant help with this *rule-management* task by providing a means of organizing and indexing modular collections of production rules according to their intended usage.

For example, a system designer might want to specify a collection of rules for diagnosing faults in the electrical systems of trucks. The intended purpose of the rules is well defined, and the designer wants a natural means of grouping them together and specifying when they are to be invoked. A frame-based system like KEE provides the desired capability by allowing designers to group the rules together into a class and associate the class with the frame representing the trucks class. The rules can then be invoked as a group whenever the system is performing a diagnostic task on a particular truck.

The rules could be associated with the trucks class in several ways. `TRUCKS`, the frame representing the trucks class, could have a member slot `DIAGNOSE` containing a method that invokes the rule class. That slot (with its method) would be inherited by all frames representing trucks, so that the rule class would be invoked whenever any truck was sent a `DIAGNOSE` message. Alternatively, the rule class could be attached as an active value to a member slot named `ELECTRICAL.FAULTS` in the `TRUCKS` frame. This active value would be inherited by all frames representing trucks and would be invoked whenever the value of the `ELECTRICAL.FAULTS` slot was requested for any truck.

Let us consider the system architecture for two types of diagnostic problems to see how frames can play an important role in managing a system's rule-based reasoning.

Classification of Situations

Knowledge systems have proved to be particularly effective for performing diagnostic tasks in a variety of domains (e.g., medicine [1, 16]). Such tasks involve determining a description of a given situation in terms of the types of situations the system knows about. Frame languages have several representational features that are particularly useful for designing and directing the reasoning processes that are involved in such diagnostic tasks.

First, the prototype descriptions included in class definitions provide a declarative means of specifying criteria for class membership. Also, because class descriptions are organized into taxonomies, each prototype need contain only those features that distinguish members of its class from arbitrary members of more general superclasses. An effective way to proceed, in fact, is to use a class–subclass taxonomy containing prototype descriptions of class members as a discrimination net for successively refining the classification of a given object [19].

For example, a classification algorithm could use a vehicle taxonomy to first determine that a given object is a vehicle as opposed to a building or a statue, then that it is an auto as opposed to a truck or boat, then that it is a sedan as opposed to a coupé or station wagon, etc. (see Figure 8). Each step in the classification uses the new information in the prototypes at the next most detailed level of the taxonomy to test for membership in each subclass.[2]

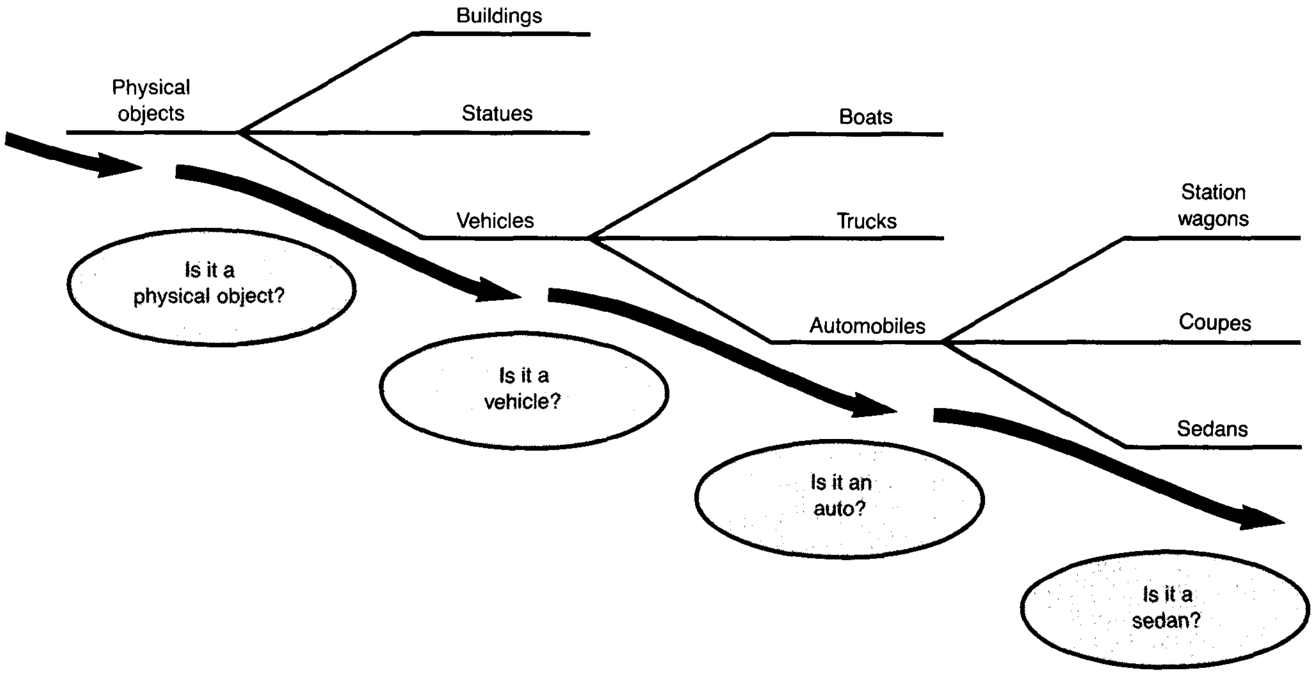
In order for such a classification algorithm to do its task, it must have criteria for determining membership in each class. Although some frame languages allow prototype descriptions that represent both necessary and sufficient conditions for class membership (e.g., KL-ONE [4] and KRYPTON [5]), most such prototypes specify only necessary conditions for class membership. For example, the prototype for `THINGS.OWNED.BY.PAUL` might indicate that the location of each of Paul's things is in Boston. However, this should not lead us to believe that Paul owns everything in Boston. Moreover, if the prototypes are considered to be only default descriptions, then they do not even specify necessary conditions. This form of prototype is satisfactory only for augmenting the description of individual class members, which is the primary use of prototypes in most systems.

Prototypes containing necessary but not sufficient conditions can be used by a classifier to conclude that an item is *not* a member of a class, but cannot be used to conclude that the item *is* a member. Production rules provide a natural way of augmenting class descriptions to include *sufficient* conditions for determining membership in a class (as in CENTAUR [1], for example). When rules are used for that purpose, the class–subclass taxonomy provides a guiding structure for designing and organizing the rules so that they are suitable for a successive refinement classification strategy.

A successive refinement strategy expects each class description to include rules that indicate whether members of some superclass can be members. The first condition of such rules is that the item be a member of the superclass. The succeeding conditions specify "local" conditions for membership in the subclass. For example, if the objective were to determine whether a particular item is a sedan, the first condition might be that the item must be an automobile. Successive conditions would then specify the conditions that an automobile must satisfy in order for it to be a sedan. The classifier can use such rules when they are associated with the appropriate class descriptions to successively work its way down a taxonomy to achieve increasingly more specific classifications.

In addition to providing a guiding structure for designing and organizing the rules that specify sufficient conditions for class membership in a diagnostic system, a frame language also provides a means of indexing and invoking rules that deduce or direct the acquisition of attribute values for the situation being diagnosed. For

[2] Note that we are considering here only the problem of determining in which classes a given item belongs as opposed to the more difficult problem of determining where a given class belongs in the class–subclass taxonomy. That general classification problem is discussed in, for example, [13].



An effective means of solving diagnostic tasks is to use a class-subclass frame taxonomy containing prototype descriptions of class members as a discrimination net to successively refine the classification of a given object. For example, a classification algorithm could use the vehicle taxonomy shown here to first determine that a given object is a vehicle as opposed to a truck or boat, then that it is a sedan as opposed to a coupé or station wagon, etc.

FIGURE 8. Classification by Successive Refinement

example, rules that determine the value of a slot can be associated with the slot as an active value. The rules will then be automatically invoked when the classifier needs to know the value of the slot.

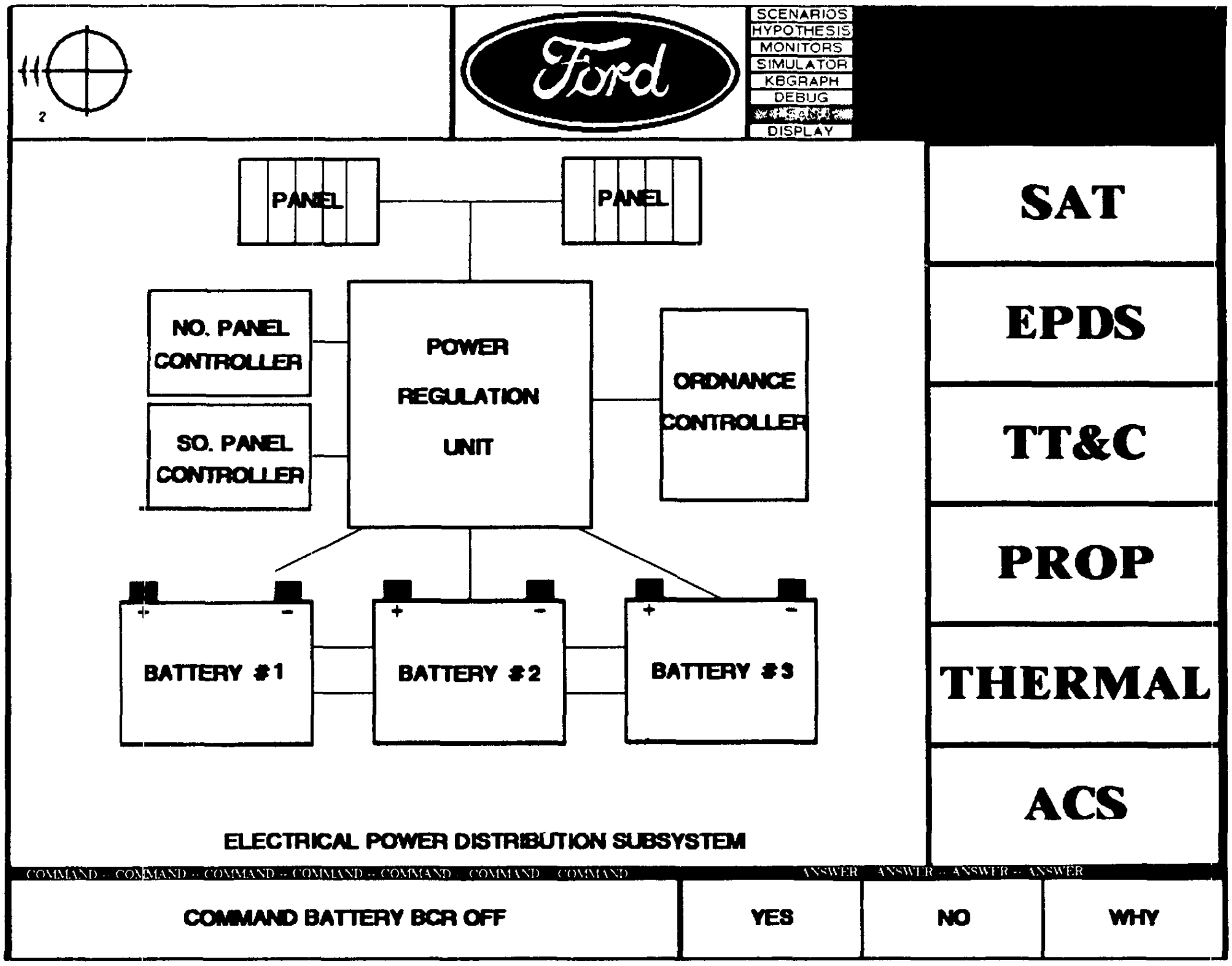
Reasoning from Significant Events

The capacity for attaching functions or rule classes that behave like demons to the slots of a frame has been used to great advantage to control reasoning in many systems (e.g., ODYSSEY [9]). The attachments that are invoked whenever the values of a slot are changed can serve as sensors, monitors, or alarms. For example, active values in the KEE system have been used as the basis for an "intelligent-alarm" facility that calls a user-supplied function only when a value of the slot crosses a critical boundary. The user establishes an alarm by providing the facility with a set of critical boundaries and the alarm function. The facility stores the boundaries and function as facets of the slot, and attaches a generic active value that checks for boundary crossings whenever a value of the slot changes.

An interesting example of this phenomenon can be found in a knowledge system under development at Ford Aerospace and Communication Corporation [8] (see Figure 9). The STAR-PLAN system is intended to serve as an intelligent aid to human satellite operators in the diagnosis and correction of satellite malfunctions. It should also be able to act alone as a simulator for training operators and diagnostic experts. An important aspect of this diagnostic task is that it requires detailed analysis from a diverse set of experts. The architecture of the prototype built by Ford using the KEE system makes effective use of demons attached to slots, methods that respond to messages sent to objects, and prototypes of experts that can be instantiated and deleted dynamically as needed during system operation.

The designers of STAR-PLAN had several requirements that led them to implement an architecture based on the integration of frames and production rules:

- They wanted the system's knowledge to be accessible and comprehensible to both diagnostic experts and satellite operators. This meant that the organization of knowledge in the system had to correspond closely to the organization used by experts and operators.
- The designers wanted to be able to build the system incrementally as experts became available and descriptions of additional satellite modules were obtained. Thus, the system's knowledge needed to be



The STAR-PLAN knowledge system being developed by the Ford Aerospace and Communications Corporation is an interesting example of how reasoning based on responses to significant events can be controlled. The system makes effective use of frames, including prototype expert frames that are instantiated and deleted dynamically as needed during

FIGURE 9. The STAR-PLAN Satellite Diagnostic System

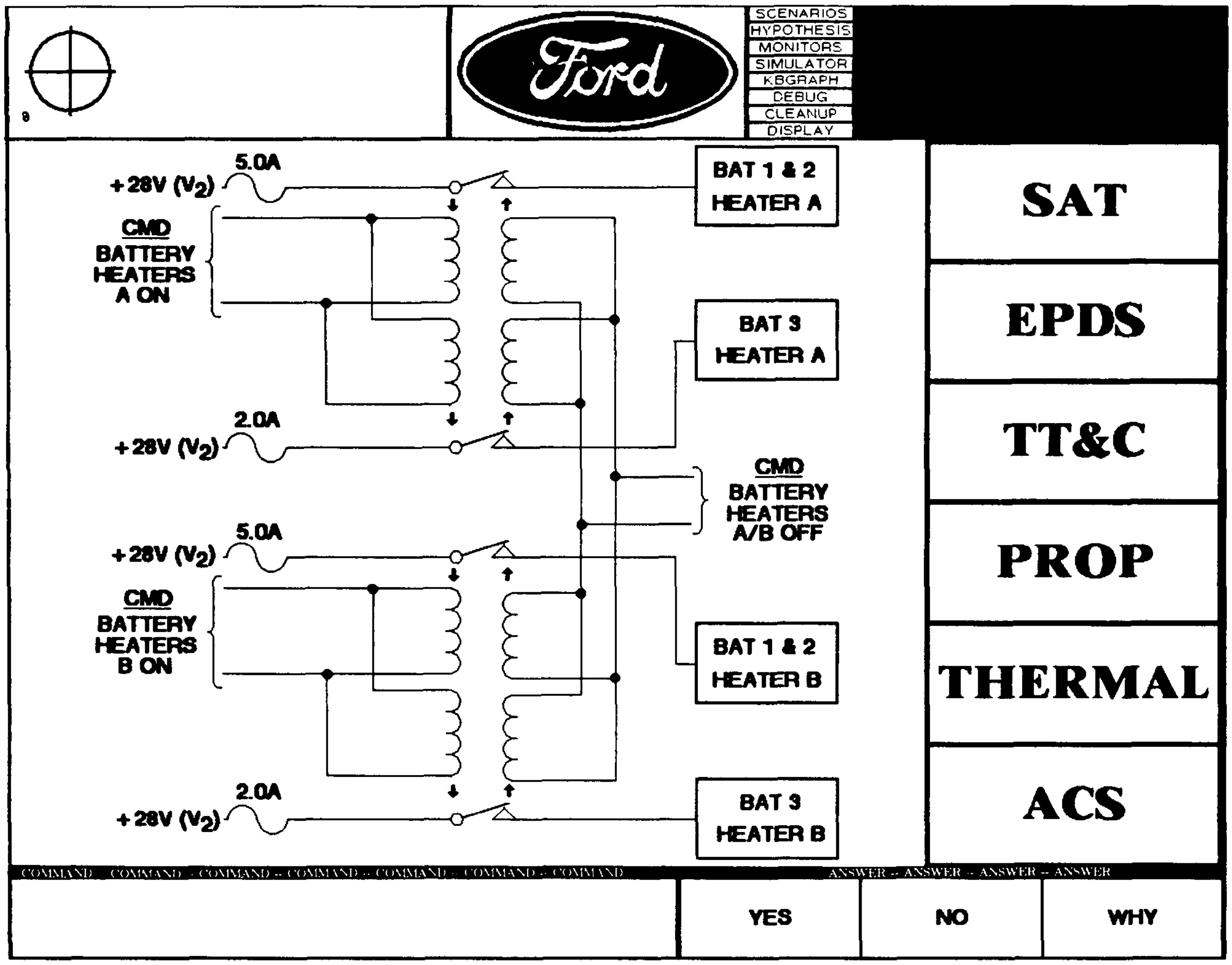
partitioned into chunks of expertise, either about particular satellite modules or about particular types of malfunctions.

- The designers knew that the system would eventually be very large and that it would be operating in a real-time environment. In order to meet speed requirements, they wanted to use a system architecture in which parts of the knowledge base could be "awakened" or "put to sleep" as situations required.

The designers of STAR-PLAN began by using the frame language to build a taxonomy describing the parts of a typical communications satellite. Methods and demons were then associated with the prototypes in the taxonomy to maintain the required relationships between the parts and to define each part's behavior. The result was a simulation capability specified in a relatively simple and natural way in an object-oriented programming style.

Two additional taxonomies were built to model the diagnostic experts. Each class in the first of these represents experts who are responsible for "watching over" a particular component of the satellite. Members of these classes are called *guardians*. Each class in the second of these represents experts who are responsible for responding to particular types of problems that occur in the satellite. Members of these classes are called *monitors*.

Guardians are created and initialized for each component of the satellite when the system is started up. When a guardian is sent an `INITIALIZE` message, it typically places intelligent alarms in the system's model of the satellite. These alarms wake up their guardian by sending it a message when a problem has occurred in



operation of the system. These screens show portions of the system's model of the satellite and its subsystems. The use of frames to build the system's models allowed the designers to organize them so that the knowledge could be easily accessible and comprehensible to both diagnostic experts and satellite operators.

FIGURE 9. The STAR-PLAN Satellite Diagnostic System

the satellite. Thus, the guardian is active only when the situation demands.

The methods associated with a guardian respond to messages from the demons by invoking a class of diagnostic rules for determining what kind of problem is occurring. The rules are applied by a *forward-chaining* rule interpreter that finds all the possible consequences of the anomalous situation that has tripped the alarms. The forward chainer proceeds by applying all rules that have a condition matching some aspect of the anomalous situation or by matching a conclusion of an already applied rule. Rule application continues in this manner until no matches remain.

The class of diagnostic rules associated with a guardian is typically very small—about 10 to 20 rules. This modularization gives the expert being modeled a small system of closely related rules to focus on while working with the system designers to develop a guardian.

When a guardian determines that a problem has occurred, it creates and initializes a monitor representing an expert for that problem. The monitor's task is to watch the evolution of the problem and make recommendations to the satellite operator. When initialized, a monitor may establish its own demons in the satellite model and put itself to sleep for a fixed period of time. Each time a monitor wakes itself up or is awakened by a message from one of its demons, it invokes a class of rules to analyze the status of the satellite. If the monitor is waking itself after a fixed period of time, the rules are typically invoked by a *backward-chaining* rule interpreter that tests specific hypotheses about the problem. The backward chainer attempts to find a sequence of rule applications that will conclude one of the hypotheses. When such a sequence is found, the rules are applied so that the hypothesis is asserted in the knowledge base. Depending on the conclusions reached by the rules, the monitor will then either put itself to sleep again or make recommendations to the operator. When a monitor concludes that the problem has been solved, it removes its demons from the satellite model and deletes itself. In so doing, it frees up memory and CPU time for the rest of the system.

The dynamic creation and deletion of monitors in the STAR-PLAN system model the way satellite problems are actually handled by human experts and operators. When a problem is recognized, the appropriate expert is called in, works with the team until the problem has been resolved, and then withdraws. A monitor's rules represent a small, problem-specific subsection of an expert's knowledge about a satellite. This modularization of rules and a familiar organizational structure makes it easier for each domain expert to create and debug rules.

CONCLUSIONS

We have described the characteristic features of frame-based knowledge representation facilities and indicated how they can provide a foundation for a variety of knowledge-system functions. We focused on how frames can contribute to a knowledge system's reasoning activities and how they can be used to organize and direct those activities.

We have also discussed the advantages of integrating frames and production rules into a single unified representation facility. The utility of such hybrid facilities is becoming increasingly evident with experience. One of the major advantages of this kind of hybrid facility is that it makes the organizational and expressive power of object-oriented programming available to domain experts who are not programmers. That is, it enables nonprogrammers to build behavioral models of application domains that include structural descriptions of the domain objects and declarative specifications of both the behavior of the objects and the behavior of experts that work with the objects. Thus, such facilities play a major role in making knowledge-system technology directly available to the application-domain experts who most need it to solve their problems.

Acknowledgments. The authors thank Bob Nado for contributions to the section on frames as a foundation for production-rule systems, and Ron Brachman, Jack Dynis, and Bob Filman for valuable comments on early drafts of this paper.

REFERENCES

1. Aikins, J.S. A representation scheme using both frames and rules. In *Rule-Based Expert Systems*, B.G. Buchanan and E.H. Shortliffe, Eds. Addison-Wesley, Reading, Mass., 1984, pp. 424–440.
2. Bobrow, D.G., and Winograd, T. An overview of KRL, a knowledge representation language. *Cognitive Sci. 1*, 1 (Jan. 1977), 3–46.
3. Brachman, R.J. What IS-A is and isn't: An analysis of taxonomic links in semantic networks. *Computer 16*, 10 (Oct. 1983), 30–36.
4. Brachman, R.J., and Schmolze, J.G. An overview of the KL-ONE knowledge representation system. *Cognitive Sci. 9*, 2 (Apr. 1985), 171–216.
5. Brachman, R.J., Fikes, R.E., and Levesque, H.J. KRYPTON: A functional approach to knowledge representation. *Computer 16*, 10 (Oct. 1983), 67–74.
6. Davis, R., and King, J. An overview of production systems. In *Machine Intelligence 8: Machine Representations of Knowledge*, E. Elcock, and D. Michie, Eds. Wiley, New York, 1977, pp. 300–332.
7. Fain, J., Gorlin, D., Hayes-Roth, F., Rosenschein, S.J., Sowizral, H., and Waterman, D. The ROSIE language reference manual. Tech. Rep. N-1647-ARPA, Rand Corp., Santa Monica, Calif., 1981.
8. Ferguson, J.C., Siemens, R., and Wagner, R.E. STAR-PLAN: A satellite anomaly resolution and planning system. Intern. Rep., Ford Aerospace and Communications Corp., Sunnyvale, Calif., 1985.
9. Fikes, R.E. Odyssey: A knowledge-based assistant. *Artif. Intell. 16*, 3 (July 1981), 331–361.
10. Green, C. Application of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence* (Washington, D.C., Aug.). Mitre Corp., McLean, Va., 1969, pp. 219–239.
11. Hendrix, G.G. Encoding knowledge in partitioned networks. In *Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Finder, Ed. Academic Press, New York, 1979, pp. 51–92.
12. Kehler, T.P., and Clemenson, G.D. An application development system for expert systems. *Syst. Softw.* 3, 1 (Jan. 1984), 212–224.
13. Lipkis, T.A. A KL-ONE classifier. In *Proceeding of the 1981 KL-ONE Workshop*. Bolt, Beranek and Newman, Cambridge, Mass., June 1982, pp. 128–145.
14. Minsky, M. A framework for representing knowledge. In *The Psychology of Computer Vision*, P. Winston, Ed. McGraw-Hill, New York, 1975, pp. 211–277.
15. Newell, A. The knowledge level. *Artif. Intell. Mag.* 2, 2 (Summer 1981), 1–20.
16. Pople, H.E., Jr. Heuristic methods for imposing structure on ill-structured problems: The structuring of medical diagnostics. In *Artificial Intelligence in Medicine*, P. Szolovitz, Ed. Westview Press, Boulder, Colo., 1981, pp. 119–185.
17. Stefik, M.J. An examination of a frame-structured representation system. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence* (Tokyo, Japan, Aug.). Kaufmann, Los Altos, Calif., 1979, pp. 845–852.
18. Stefik, M., Bobrow, D.G., Mittal, S., and Conway, L. Knowledge programming in LOOPS: Report on an experimental course. *Artif. Intell.* 4, 3 (Fall 1983), 3–14.
19. Woods, W.A. What's important about knowledge representation? *Computer 15*, 10 (Oct. 1983), 22–29.

FURTHER READING

Useful collections of papers on knowledge representation include the following:

Bobrow, D.G., and Collins, A.M., Eds. *Representation and Understanding.* Academic Press, New York, 1975. Includes papers describing early work on frame-based representation languages.

Findler, N.V., Ed. *Associative Networks: Representation and Use of Knowledge by Computers.* Academic Press, New York, 1979. Contains papers focused specifically on semantic networks.

IEEE Computer Society. *Computer*, Special Issue on Knowledge Representation, *16*, 10 (Oct. 1983). Provides a representative sampling of recent work in knowledge representation.

CR Categories and Subject Descriptors: I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search
General Terms: Design, Languages
Additional Key Words and Phrases: deductive retrieval, definition by specialization, frames, knowledge systems, managing rule-based reasoning, object-oriented representation, prototype descriptions

Authors' Present Address: Richard Fikes and Tom Kehler, IntelliCorp, Knowledge Systems Division, 707 Laurel Street, Menlo Park, CA 94025.