Using Object Concepts to Match Artificial Intelligence Techniques to Problem Types*

Barrie R. Nault
Graduate School of Management
University of California, Irvine
Irvine, California, USA, 92697-3125
Phone: (714) 824-8796; Fax (714) 824-8469
brnault@uci.edu

Veda C. Storey
Dept. of Computer Information System
College of Business Administration
P.O. Box 4015
Atlanta, GA 30302-4015
Phone: (404) 651-3894; Fax: (404)-651-3842
vstorey@gsu.edu

7 May 1998

Keywords: Objects, Knowledge Representation, Frames, Semantic Networks, Production Rules

Section: Research

Original submission: 13 August 1996
Returned to authors for change: 29 May 1997
Resubmitted: 22 September 1996
Accepted: 26 April 1998

 The authors wish to thank the editor and reviewer for their detailed comments on an earlier version of this paper.

Using Object Concepts to Match Artificial Intelligence Techniques to Problem Types

Abstract

Using object-concepts as a matching framework, we provide guidelines for identifying what types of problems are best served by which knowledge representation technique. We find that production rules are best for hierarchical classification problems, because they support classification/instantiation of data. Frames are best for data retrieval and inference problems, because, using data abstraction, frames can operate on data within a frame. Finally, semantic networks are best for consequence finding problems, because of independence of the primitives in the hierarchy. Providing guidelines for this matching is important, because the success of different information systems designs have been shown to depend explicitly on problem characteristics.

Keywords: Objects, Knowledge Representation, Frames, Production Rules, Semantic Networks, Problem Types.

1 Introduction

A key determinant of information systems (IS) success has been the matching of IS design characteristics to characteristics of the problem the IS is intended to support. For example, in interface design, classical studies on decision support systems (DSS) [5], [6], [16] conclude that the problem type is critical in determining which presentation format is best. In assessing empirical work in managerial support systems, Benbasat et al. [7] suggest that studies of knowledge-based systems, such as expert systems, are likely to follow along lines similar to those in DSS, refined by additional theory focusing on various aspects of knowledge, including its representation. This paper continues by conducting an analysis that matches the three classical knowledge representation (KR) techniques (frames, production rules, and semantic networks) to a three-category classification of problem types. Object concepts are used as a framework to perform the analysis. As a result, we contribute to the definition of a criteria for matching "techniques to tasks" and identify which specific knowledge representation techniques are most suitable for certain problem types.

Knowledge representation provides a formalism with which to represent information about the real world [34]. Although there are several well-established techniques, there has not been a formal, systematic comparison among them that provides guidelines for identifying the best technique to employ, given the characteristics of a particular problem.

We believe that object concepts can be useful for distinguishing between KR techniques for

specific tasks. Object-oriented (OO) concepts and approaches appear throughout the IS literature [1, 9, 17, 21]. Viewing the world as being made up of objects is considered very *natural*; that is, allowing a simple and direct translation between the real world and IS models. These benefits are considered to be important and the approach is often termed a *paradigm* (e.g. [28, 40, 41]).

The object paradigm is, therefore, adapted as a formal, unifying framework from which to analyze knowledge representation techniques and the mapping between them and problem types. Previous research has recognized that connections exist among OO programming, KR and linguistics, and data modeling (e.g., [42]). The object framework provides a mechanism for highlighting similarities and differences among the various KR techniques, and predicting the strengths and weaknesses of each technique as it is applied to different problem types. The objectives of this research are:

- to propose object concepts as a framework for evaluating and contrasting KR techniques and for explaining the distinctions among them;
- to assess the extent to which each of the main object properties can be captured by a given KR technique; and
- to use object concepts to identify the characteristics of an application that make one KR technique more appropriate for representing it.

2 Objects

The term object in the IS world means "a combination of 'data' and 'program' that represent some real-world entity" [20]. The object paradigm is usually associated with a set of concepts that indicate whether an application, programming language, or design technique is OO [29]. These concepts are more often defined in terms of *data abstraction*, *classification, inheritance, independence, message passing, homogeneity*, and *concurrency. Composition* and *polymorphism* may also be included [22]. [1] Unfortunately, these concepts are defined at different levels of abstraction. Data abstraction, classification, inheritance, and composition are defined at a conceptual level: they correspond to properties of things that are present in the real world. Concurrency and polymorphism are usually implementation-level concepts. Independence, message passing, and homogeneity fall

[1] Polymorphism, in OO programming, refers to the capability for different classes of objects to respond to exactly the same protocols [39].

somewhere in between. Here we compare and analyze knowledge representation techniques at the conceptual level, so the implementation-level concepts are not considered further.[2]

Object concepts are used as a framework for analyzing each of the KR techniques. Because they appear throughout the analysis, working definitions are needed for *stores* and *methods*:

- A *store* is a state variable; that is, a variable that instantiates a value for some attribute or feature. For example, 'Matilda' may be a value of the employee attribute, "name".
- A *method* is an operation that may be performed on an object. It operates on the values of an object's attributes is a local operation that manipulates the internal state of an object [2]. Methods can be divided into three basic types. The first is *rules* that govern the structure of stores. These are similar to semantic integrity constraints; for example, "employee-age" must be between 15 and 75. The second type changes values in stores in response to a *trigger*; for example, "apply overtime rate to salary when the number of hours exceeds 40 per week". The third type manages the *interface between objects*; for example, a message of type Z can only be exchanged by objects that share a particular interface definition.

The object concepts are outlined below. Similar definitions and further details are found elsewhere [15, 24, 43].

Data Abstraction: In an object, states (stores) and procedures (methods) that operate on the states are defined as components of the same basic unit. Objects contain descriptions of both their state and the behaviors that govern change in state. For example, a particular configuration of pieces on a chessboard can be represented in the same structure that defines the range of legal next moves.

Classification/instantiation: An object class is a grouping to which objects belong. An object type defines stores and methods for each instance in the class; these are created when stores take on specific values.[3] *Classification* is a relationship between an instance and its type. The inverse relationship is *instantiation*. For example, the class of possible chess board configurations is defined

[2]Homogeneity could be considered an implementation property only.

[3]This description intentionally avoids describing classification/instantiation as a grouping of objects based on common stores because the definition of an object *type* is important in the object paradigm.

by an object type that can represent any configuration; a given board configuration is an instance of the object type.

Inheritance: Specialization occurs when a subtype is defined from an object type; for example, *Manager* is a subtype of *Employee*. A subtype can inherit all of the stores and methods from its parent type via *inheritance*. A subtype may also contain additional stores and methods that are not defined for its parent. For example, *Manager* could have an additional store, such as "number-of-people-reporting". Similarly, an instance of the subtype inherits stores and methods from its parent type and additional stores and methods from its (sub)type, and gives them values. Repeated subtypes result in a hierarchical structure. Multiple inheritance allows multiple parent types and results in a *lattice*. A *Student-Employee*, for example, is a subtype of both *Student* and *Employee* and, thus, it can inherit stores and methods from both.

Independence: Objects are entities that have control over their own state; that is, the only way to change an object's state (the values in its stores) is through its own methods. These methods respond only to messages from other objects.

Message Passing: Objects communicate by exchanging messages. The protocols for sending and receiving messages are part of each object's methods: those that manage the interface. Thus objects do not "operate" directly on one another, thereby maintaining independence.

Homogeneity: Objects are *fractal* in the sense that the same structure appears at all levels of analysis. Moreover, similar capabilities are maintained at all levels. There is, however, a circularity associated with homogeneity.

Composition: Composition is similar to database aggregation and makes use of a flexible (i.e., not strict) form of multiple inheritance. Partial inheritance enables an aggregate object type to inherit only selected stores and methods from its components; for example, a car may inherit an attribute "horsepower" from its engine. The composite may have *emergent* properties, which relate to the properties of the components, but need not be derivable from them; for example, "processing power" of a computer system. Some emergent properties are derivable through methods; for example, "number-of-components". The set of stores and methods from emergent properties cannot

be obtained from partial inheritance. Instead, they must be specified in a separate object type.

3 Knowledge Representation Techniques and Object Concepts

Representation of Knowledge is defined by Brachman and Levesque [10] as "the writing down, in some language or communicative medium, descriptions or pictures that correspond in some salient way to the world or a state of the world." KR is concerned with the way in which large bodies of knowledge can be formally described [3]. Various knowledge representation techniques have been proposed, the most well-known of these are frame, production rule, semantic network, and logic. Logic is a formal system that can, in principle, represent anything. However, the existence of rigorous semantics is not always helpful, since the semantics are often not at the right level of abstraction.

3.1 Frames

3.1.1 Overview

Minsky [26] first defined frames as "data structures for representing stereotypical situations". A frame is a collection of attributes and associated values that describe some entity in the real world [35]. Frames provide a convenient way to combine declarations and procedures within a single knowledge representation [30].

A frame is a network of nodes and relationships. The top levels are fixed to capture a situation. The lower levels are *terminals* or *slots* filled with specific data. Terminals can have conditions defining legal assignments, similar to integrity constraints, or a method. They can also be grouped into collections of related frames, called "frame systems" where different frames share the same terminals. Different frames can represent different ways of using the same information located at common terminals. Terminals can be considered to be *subframes* of a higher-level frame.

Two aspects of frames are relevant to objects [23]. First, frames operate with instantiation, so, in principle, a frame exists for each particular situation. Second, frames and subframes are analogous to descriptions inside descriptions, thus, maintaining homogeneity across levels.

3.1.2 Frame Object Concepts

Because frames are a single construct KR technique, a frame is the appropriate construct to compare to an object.

Data Abstraction: The original model of a frame contained both terminals and methods that could govern the contents of the terminals. Although some implementations of frame-based systems include trigger methods, these are not actually part of the conceptual structure. Explicit communication between frames is not considered; therefore, there are no methods corresponding to an interface. Like objects, frames may contain multiple methods (through attached predicates). These methods are flexible as they can be defined differently for each frame, and can operate on stores within the frame.

Classification/instantiation: Frames support classification and instantiation. The original situation frame (corresponding to an object type) is gained from experience [36]. Specific instances of the situation are then represented by attaching more detailed information to the terminals. This further information is not data attached to new terminals; rather, it is information that replaces the weakly-connected default assumptions.

Inheritance: Surprisingly, there is no mechanism for single or multiple inheritance in frames. Nevertheless, the mechanisms for propagating stores and methods between types and subtypes can be represented through the methods, thus, representing inheritance explicitly. For example, a frame-based class may be defined by a type that has subclasses.

Independence: Frames do not operate *on* each other so part of the independence concept is implicit. Data attached to terminals is updated through experience with a particular situation rather than changed by another frame. On the other hand, frames can share terminals or subframes. Thus, changes in one frame can affect changes in another, and in this way, frames are interdependent.

Message Passing: Frames do not support message passing because frames are not *active*. Thus, the notion of communication does not apply.

Homogeneity: This and composition capture the power of frames. Homogeneity exists because of subframes. Frames, however, contain only terminals (if one defines the top-level, fixed information as terminals), and each terminal is a subframe. As a result, all stores are frames. Frames display only partial homogeneity because methods that govern the structure of terminals are not necessarily specified as frames.

Composition: Composition is the object property most exploited by frames. Frames can be grouped together based on shared terminals. Using the connection between frames and subframes through terminals, a hierarchical composition can be obtained. A frame type can have terminals defined only for that type. Terminals can also be aggregations of terminals from other frames. Therefore, emergent properties can be represented: both those that are not derivable and those that are directly derivable from terminals in other frames.

3.1.3 Frame-Object Summary

Frames differ from objects in that frames are not independent, frames do not communicate through message passing, methods are not defined as frames, and inheritance must be represented explicitly. Objects maintain their own stores at the expense of redundancy. Frames, on the other hand, employ shared terminals, even between frame arrays or compositions. Thus, frames compensate for the lack of inheritance through homogeneity and composition.

The deficiency in frames, compared to objects, is the lack of a mechanism for handling dynamics, such as methods and message passing. To a certain extent, the default information removes some of the need for additional reasoning. Modification of terminal values and matching through an information retrieval network are mechanisms that could be used in *conjunction* with frames, as opposed to being part of the frame construct itself. As such, the infrastructure for managing dynamics is not part of a frame.

A related distinction between frames and objects is the definition of a type. Frames, as originally developed, have no initial definition of a frame type prior to the development of a frame for a given situation. Frames can add new terminals over time and, thus, the type can evolve. Objects, however, require a redefinition of an object type in order to add new stores.

3.2 Production Rules

3.2.1 Overview

Production systems, originally proposed by Post [32], are widely used in expert systems [8].[4] Production rules are claimed to be both natural and plausible, thereby providing a means of representation that is very similar to the structure of fundamental mechanisms of human cognition [14, 37]. Production rules are often used in systems for diagnostic or categorization tasks (e.g., [25]). In general, they are well-suited for problems that: (a) have many, relatively-independent facts rather than a single, unified theory; (b) have processes that consist of a set of independent actions, as opposed to those involving multiple interconnected processes or subprocesses; and (c) have knowledge that can be separated from the manner in which it is used, as opposed to knowledge that describes how a prespecified problem is solved (for example, a recipe) [4]. A production system has three main components: 1) a rule base; 2) a working memory (also called a context); and 3) an inference engine.

The rule base contains a set of production rules, the general form of which is:

IF: Condition
THEN: Action

E.g.:
IF: Client(X)'s bill has not been paid
THEN: Send Client(X) overdue notice

A rule usually corresponds to a particular "chunk" of knowledge about an application domain. The condition (premise or antecedent) is often a predicate or conjunction of predicates that must be present before the action (conclusion or consequent) can take place. The working memory contains information about a given problem and the inferences a system makes during a consultation session. The inference engine actually runs the production rules. To accommodate incomplete or uncertain information, "degrees of confidence" or certainty factors are attached to a given rule.

3.2.2 Production Rules Object Concepts

In production systems there is only one construct, a production rule, because a fact is an unconditional rule, or a rule in which the premise is null.

[4] Bench-Capon even refers to production rules as a knowledge representation paradigm.

Data Abstraction. The stores of a given production rule are variables based in the condition and action of the rule. There is only one kind of method. The rule variables in the action part of the rule have their values re-evaluated as rules are "fired" (executed). The result of the firing provides a "trigger" method. Thus, the rule mechanism corresponds to the object method which changes values in local stores.

Classification/instantiation. Production rules support classification/instantiation in both types and instances. A rule type, corresponding to an object type, can be defined as:

IF: Student(X)
THEN: Pay Fees(X).

The rule is applicable to each instance of student. Production rules have an additional feature: data or stores can also be typed, as illustrated with the variable X; that is, X is defined across rule types.

Inheritance. Inheritance must be defined explicitly. For example, if A is a subtype of B, and there is a rule whose premise involves B and whose consequent is C, then the rule also applies to A.

IF: Manager(X)
THEN: Employee(X)
and
IF: Employee(X) & Performance(X, "superior")
THEN: Wage increase(X, 10%)
Implies:
IF: Manager(X) and & Performance(X, "superior")
THEN: Wage increase(X, 10%)

Independence: Production rules are intended to be independent. However, stores in premises and conclusions from different rules are shared. That is, they appear in premises or conclusions in other rules. Therefore, similar to frames sharing terminals, production rules share variables. As a result, a change in the value of a variable in the action portion of one rule can change the impending (after firing) status of a separate rule [18]. The firing of one rule, however, does not force another rule to fire.

Message Passing: Production rules do not support formal message passing. There is no protocol for sending or receiving messages. The value of a variable may be changed by one rule that directly affects the evaluation of another rule, but this outcome is due to shared variables rather than message passing.

Homogeneity: The methods embodied by production rules are homogeneous. The stores in the premise and consequent are not in rule form. Thus, rules support only partial homogeneity.

Composition: A limited form of composition is possible. Consider a bicycle as an aggregate of a seat, wheels, and spokes. In representing the fact that these components are sufficient for the whole, the components would appear in the premise:

IF: Seat(X) and Wheel(Y) and Spokes(Z)
THEN: Bicycle(XYZ)

Alternatively, if these components are a necessary part of the whole, they would be part of the conclusion, and the above example would be reversed. Only by explicitly writing both rules can some kind of (artificial) aggregation be captured. Full composition is not possible because production rules cannot represent emergent properties.

3.2.3 Production Rule-Object Summary

Compared to objects, production rules are severely restricted in their ability to represent knowledge: only in classification/instantiation do production rules fully capture object concepts. Production rules have a single, inflexible (although general) method. They also lack independence and the ability to pass messages directly. Both inheritance and composition must be represented explicitly. The data in a rule is not structured as a rule, so production rules exhibit only partial homogeneity.

Although no direct capabilities exist for passing messages, between-rule dynamics are maintained by the inference engine that decides which rule to "fire" next, and the working memory that stores the (shared) variables. Both of these features are implementation dependent.

3.3 Semantic Networks

3.3.1 Overview

A semantic network is "a structure for representing knowledge as a pattern of interconnected nodes and arcs" [38]. Originally developed by Quillian [33] and others as a psychological model of human associative memory, semantic networks are very general.

A semantic network is made up of nodes and arcs. Nodes represent concepts of entities, attributes, events, and states. Arcs are conceptual relations representing relationships between the concept nodes. The nodes may be physical things, conceptual entities, or descriptors. The links can be *is-a* links, representing subclass-superclass relationships, or *has-a* links, indicating that one thing is an attribute of another. Links may also be definitional; for example, "employees *work-for* companies". They can even capture heuristic information; for example, "overworking *causes* headaches" [19]. However, there are no well-defined formal semantics for network representations [31]. Although semantic networks and frames have a great deal in common, a frame system has more structure on the attributes and inference mechanisms and knowledge is organized into larger chunks. Figure 1 shows an example of a semantic network.

The nodes in a semantic network can be arranged into an *inheritance hierarchy*, whereby nodes lower in the hierarchy automatically inherit properties of the higher-level nodes. Inheritance gives new nodes the ability to know information about many attributes, capabilities, and constraints as soon as these nodes are created. For example, if a new employee is added, he or she automatically inherits the information from the nodes above it [27]. Nodes can be used to represent both *instances* of things (e.g., 'James Rogers') and the *type* of a thing (e.g., *Manager*). Moreover, inheritance is through inference rather than shared variables.

3.3.2 Semantic Network Object Concepts

A node, along with its related arcs, is compared to an object.

Data Abstraction: Nodes represent things, either by variables (e.g., *Manager*), values (e.g., 'James Rogers'), or as compositions (aggregations) of other things (e.g., *Company* has links to its parts; *Department* is one of them). Variables or constants that represent values are both stores; thus, a node can act as a single store. Methods, such as semantic integrity constraints, can be

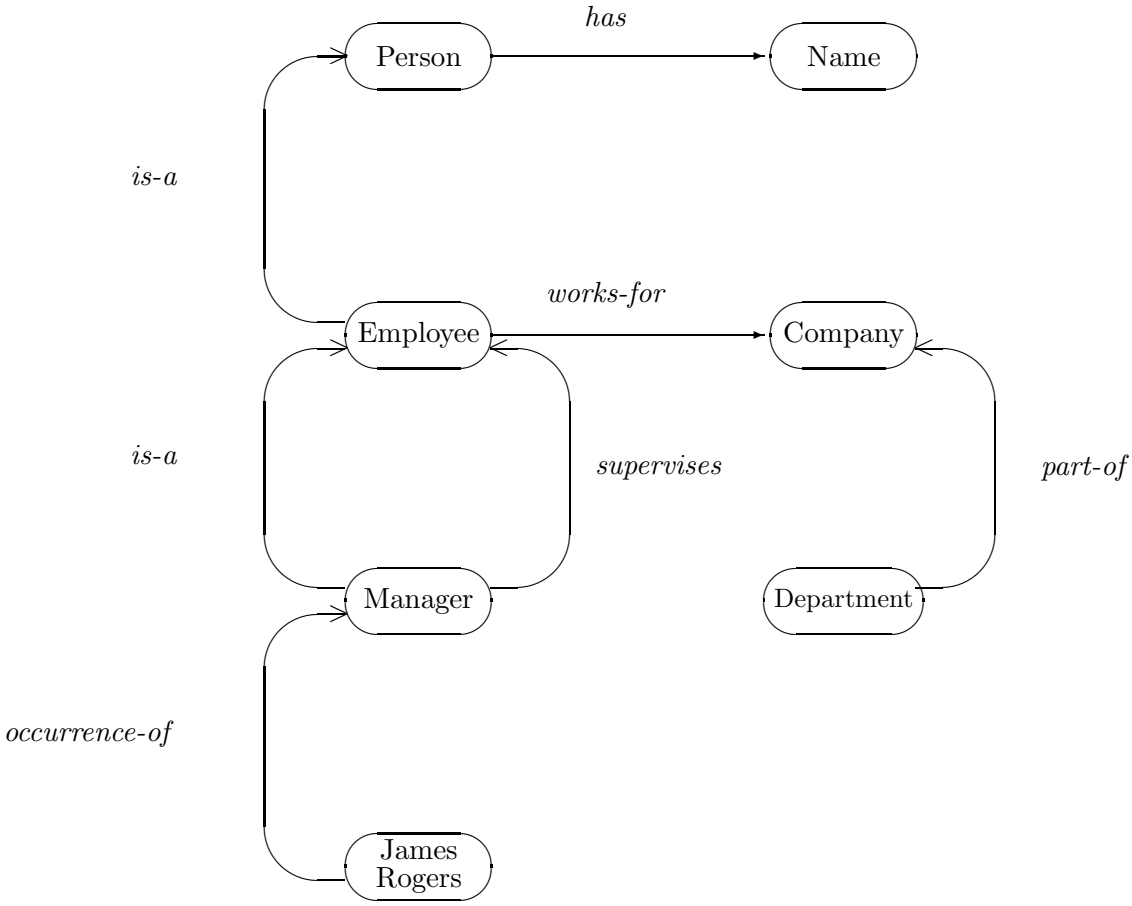

Figure 1: Semantic Network

defined as relationships (for example, Manager *supervises-maximum-7* Employees) between nodes. In this way, multiple methods can easily be defined. Methods cannot operate on data in the nodes.

Classification/instantiation: Classification and instantiation are accommodated in a semantic network by designating nodes such that one node defines the type and another its subtype, through an *is-a* arc (i.e., specialization). An arc labelled *occurrence-of* indicates instantiation.

Inheritance: The main quality of semantic networks is inheritance. Nodes can be arranged into subtypes and supertypes using *is-a* links. This specialization is used to infer properties, rather than directly specifying properties. Multiple inheritance is supported; for example, an *Intern is-a Student* and an *Intern is-an Employee* can both be defined on the node, *Intern*. Partial inheritance is not accommodated.

Independence: As in frames, semantic networks do not operate on each other, nor do they share variables directly; therefore, one aspect of independence is implicit. To illustrate, suppose a particular individual is represented in a semantic network and changes her name (e.g., from 'Joanne' to 'Joan'). This will change the value in the *name* node, but it will not directly affect other nodes.

Message Passing: Semantic networks do not support message passing. To achieve this objective, message passing channels would need to be represented by different kinds of arcs than have traditionally been used.

Homogeneity: Semantic networks exhibit some degree of homogeneity because all things are represented by nodes. For example, both attributes (e.g., "color") and values of attributes (e.g., 'green') can be organized in a semantic network. Relationships, however, are not represented by nodes.

Composition: Composition can easily be represented. A thing that is an aggregation of other things can be captured by *has-a* arcs (e.g., address *has-a* street, city, state, and zip code). Aggregation of physical objects can be represented by labelling each directed arc from a component to an aggregate as *part-of*. Emergent properties can be added as new attribute nodes and associated arcs.

3.3.3 Semantic Network-Object Summary

Semantic networks capture many of the object concepts. However, semantic networks do not support message passing, nor do they allow for any operations on data or stores. Semantic networks are extremely good for dealing with inheritance, both at the type level (from type to type nodes) and at the instance level (occurrences inherit the properties of the type). The ability of semantic networks to represent static knowledge is evident from the claim that they are well-suited for representing taxonomies of knowledge. Part of this is due to their being extremely flexible (for example, both types and occurrences can be represented within the same semantic network).

4 Discussion of Analysis

The object-oriented approach incorporates and refines concepts that have proven to be effective in the classical approaches to KR. The object concepts that also appear in the KR techniques are classification/instantiation, inheritance, and composition. Binding of data and procedures (data abstraction), the goal of much work in computer science, is facilitated through the combination of independence, message passing, and homogeneity. These are the major improvements that make the object approach an effective way to model information systems. Thus, the object paradigm has emerged as a composite of concepts from earlier KR techniques and philosophical modelling simplifications.

The preceding discussion examined the extent to which the three KR techniques accommodate the main object concepts. The results are summarized in Table 1. We conclude that: 1) frames best capture the essence of data abstraction; 2) production rules have an advantage in classification/instantiation; and 3) semantic networks are strong in inheritance, independence and homogeneity.

An advantage of frames is that they can contain multiple methods that provide for flexibility in definitions; most importantly, the methods can operate on data or stores within a frame. The main characteristic of production rules, relative to the other KR schemes, is the classification/instantiation of data through the use of variables. Thus, stores can be typed. Semantic networks are differentiated by their direct support of inheritance, independence of arc-node combinations, and homogeneity, with the first two being the most important.

We use object concepts to distinguish among the three knowledge representation techniques

Table 1: **Knowledge Representation: Object Concepts**

	Frames	*Production Rules*	*Semantic Networks*
DATA ABSTRACTION			
• Multiple Methods	Yes	One (If-Then)	Yes
• Flexibility of Definition	Yes	No	Yes
• Operate on Data	Yes	Yes	No
CLASSIFICATION/ INSTANTIATION	Yes (Instance-Type)	Yes (Instance-Type-Data)	Yes (Instance-Type)
INHERITANCE	Must be represented explicitly	Must be represented explicitly	Yes
INDEPENDENCE (Share Variables)	No (shared variables)	No (shared variables)	Yes (except for *is-a* nodes.)
MESSAGE PASSING	No	No	No
HOMOGENEITY	Partial (slots can be frames; methods cannot)	Partial (method is a rule; data is not)	Yes
COMPOSITION	Yes (can represent emergent properties)	Yes (must be explicitly represented; cannot represent emergent properties)	Yes (can represent emergent properties)

because we believe that object concepts provide a means for "matching techniques to tasks".

5 Problem Types

Developing a topology of problem types that is robust to a wide variety of problems is extremely difficult, as is evident from the vast amount of literature that can be found on this subject. The most prolific work on identifying generic tasks has been carried out by Chandrasekaran [11] who identifies between three and five generic problems. We focus on the three most well-defined problem types: 1) hierarchical classification; 2) data retrieval and inference; and 3) consequence finding. Our objectives are to:

- describe each problem type;
- assess their knowledge representation requirements relative to the object concepts used to differentiate the KR techniques; and
- develop a mapping between knowledge representation techniques and problem types based on the object concepts. This should enable us to predict which KR techniques are most suitable to a given problem type.

5.1 Hierarchical Classification

Hierarchical classification has a wide variety of applications. A problem solving approach has a hierarchical classification if it involves establishing concepts that are more general at the top and become more refined as one moves down the hierarchy. Thus, this type of problem solving involves establishing whether a concept is true or false, and if true, identifying which of its successors may result. Often characterized as an *establish-refine* problem solving approach, hierarchical classification supports distributed problem solving; that is, one can attempt to establish different concepts at the same time. However, the major difficulty with this type of distributed problem solving is providing a representation that captures the current state of affairs across the hierarchy; that is, one must find a way to connect the concepts that have been established. Hierarchical classification is the identification of a case description in which a specific node appears in a predetermined diagnostic hierarchy. The terms "diagnostic task" [12] and "classificatory task" [11] have been used to refer to hierarchical classification.

5.2 Hierarchical Classification and Object Concepts

Frames, production rules, and semantic networks each have object concepts that are potentially useful for hierarchical classification. Methods operating on data within a store allow for the establishment of concepts throughout distributed frames. Independent arc-node combinations also support distributed problem solving. Neither of these advantages, however, have the potential to capture the current state from a set containing established concepts, refuted concepts, and non-established concepts.

The only object concept differentiating production rules *would* allow for such a distributed state to be captured. Production rules support classification/instantiation for *data* as well as for types and instances. They are able to represent instances through variables; i.e., an instance of a "problem" can be represented by a particular variable (say X) that links together all of the rules. Thus, a top-down, establish-refine distributed problem solving approach can proceed for a given instance of a problem. The state of the hierarchy can be determined through the instantiation of variables. Thus, by analyzing production rules using object concepts, we find that production rules have an advantage in hierarchical classification problem solving tasks.

5.3 Data Retrieval and Inference

Data retrieval and inference refers to a problem solving approach that involves determining how the value of a datum could be obtained and possibly inferring defaults if the datum is not directly available. These are problems where records first need to be accessed before conclusions can be drawn.

Inference chains of more than a single rule may be required to determine a given item of information; e.g., suppose a medical rule says that if a patient has been exposed to anesthetics, then hepatitis should be considered. The data at hand, however, indicates only that the patient has had recent major surgery. Diagnostic reasoning, supported by hierarchical classification, would not necessarily make the connection that major surgery (usually) requires anaesthesia.

Consistent with one of [12], we conclude that the main characteristic of this problem solving approach is the integration of the knowledge base and a problem solver; that is, problem solving must be embedded in the knowledge structure.

5.4 Data Retrieval and Inference, and Object Concepts

The key to joining problem solving and domain knowledge is the object concept of data abstraction: the knowledge representation technique that best captures the features of data abstraction will be best. Both frames and semantic networks accommodate multiple methods and flexibility of definition. However, frames are also able to operate on data within the frame, and thus, are the better knowledge representation technique for this type of problem.

It might appear that the inheritance object concept advantage of semantic networks suggests that they have a great advantage in data retrieval and inference. This, however, is hardly an advantage because inference is not always based on inheritance, and frames can explicitly represent inheritance through the definition of methods. Unlike hierarchical classification, classification/instantiation of data is not critical here, and, thus, does not provide production rules with an advantage here.

5.5 Consequence Finding

Consequence finding, also called "what-will-happen-if" state abstraction, involves the prediction of consequences from a given action. This problem solving differs from hierarchical classification in a fundamental way: the search for effects rather than causes. Consequence finding involves a bottom-up search in contrast to diagnosis which is top-down. Consequence finding results in a state change, where the action provides a change in state and the objective is to determine the effect on other parts of the system as a result of this state change.

The knowledge base for consequence finding typically encompasses expertise about the structure of the device or system, or functionality of components and subcomponents, decomposed hierarchically. The corresponding problem solving is triggered by a change in the state of a primitive (at the bottom of the hierarchy) and the consequence of this change is determined by a bottom-up recomposition through the hierarchy. Independence of the primitives at the bottom of the hierarchy is critical, of course.

5.6 Consequence Finding and Object Concepts

Message passing would be a natural mechanism for propagating the state change from the primitive where the change in state has occurred, upwards in the hierarchy. Unfortunately, none of the three

knowledge representation techniques includes a facility for message passing.

The other object concept important for consequence finding is independence, specifically of the primitives in the hierarchy. Semantic networks are the only knowledge representation technique with this object property. Node-arc combinations in semantic networks do not share variables, whereas both frames and production rules do. It is this separation of variables in the primitives of the hierarchy that is necessary for the bottom-up process of consequent finding to result in an unambiguous effect. Inheritance, another object concept that is unique to semantic networks, is a top-down technique and thus does not provide consequence finding. Homogeneity, the third object concept where semantic networks have an advantage, may also be useful in representing the components of the decomposition.

5.7 "Matching Techniques to Tasks": Results and Predictions

Our analysis can be summarized in three hypotheses:

- H1: Production rules are more appropriate for hierarchical classification problems.
- H2: Frames are more appropriate for data retrieval and inference problems.
- H3: Semantic networks are more appropriate for consequence finding problems.

We conclude that:

1. the ability to operate on data is the main advantage for data retrieval and inference problems;
2. instantiation of data is critical for hierarchical classification problems; and
3. independence is essential for consequence finding problems.

The hypotheses are descriptive rather than explanatory. Our conclusions explain *why* a given technique is more likely to be used for a given problem type. Empirically verifying these explanations would involve examining whether the object concepts we have identified as being important in the mapping of techniques to tasks are, in fact, the critical ones.

Obviously, any empirical evaluation requires that a given implementation of a knowledge representation technique actually has the object concepts needed and does not embody concepts that

are unique to other knowledge representation techniques. Otherwise, a given implementation of might have more object concepts than our analysis would indicate. Indeed, implementations in many expert-system toolkits include components from more than one technique.

6 Conclusion

This research has taken three steps towards using the object approach as a framework from which to evaluate knowledge representation techniques and problem types. First, three KR techniques have been characterized in terms of the object concepts captured by each. Second, using the object concepts that distinguish each of the knowledge representation techniques, three problem types were shown to differ in the degree to which the critical distinguishing object concepts are necessary. Finally, from our normative analysis we generated predictions about which knowledge representation techniques are more likely to be used for various types of problem solving tasks. We conclude that production rules are particularly well-suited for hierarchical classification, frames are best suited for data retrieval and inference problems, and semantic networks are most appropriate for consequence finding.

References

[1] Adam, N.R., and Bhargava, B.K. (Eds.), *Lecture Notes in Computer Science: Advanced Database Systems*, Springer-Verlag, New York, 1993.

[2] Albano, A., Bergamini, R., Ghelli, G., and Orsini, R., "An Object Data Model with Roles", *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Ireland, 24-27 August, 1993, pp.39-51.

[3] Ashrafi, N., Kuilboer,J.-P., and Wagner, J.M., "Expert Systems Reliability: A Life Cycle Approach," *Information & Management*, Vol.28, 1995, pp.404-414.

[4] Barr, A. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence*, William Kaufmann Inc., Vol.1, 1981.

[5] Benbasat, I. and Dexter, A.S., "Experimental Evalutation of Graphical and Color-Enhanced Information Presentation", *Management Science,* Vol. 31, No. 11, 1985, pp. 1348-1364.

[6] Benbasat, I., Dexter, A.S. and Todd, P., "An Experimental Program Investigating Color-Enhanced and Graphical Information Presentation", *Communications of the ACM,* Vol. 29, No. 11, 1986, pp. 1094-1105.

[7] Benbasat, I., DeSanctis, G. and Nault, B.R., "Empirical Research in Managerial Support Systems: A Review and Assessment", *Proceedings of the NATO Advanced Study Institute on Decision Support Systems,* A.B. Whinston and C. Halsapple eds., Springer-Verlag, 1991, pp. 383-437.

[8] Bench-Capon, T.J.M., *Knowledge Representation: An Approach to Artificial Intelligence*, Academic Press, Harcourt Brace Jovanovich, Publishers, London, 1990.

[9] Booch, G., *Object Oriented Design with Applications*, Benjamin/Cummings Publishing Company, Redwood City, California, 1991.

[10] Brachman, R.J., and Levesque, H.J., "Introduction", in *Readings in Knowledge Representation*, Brachman, R.J., and Levesque, H.J. (Eds.), Morgan Kaufmann Pub., 1985, pp. xiii-xix.

[11] Chandrasekaran, B., "Towards a Taxonomy of Problem Solving Types", *The AI Magazine*, Winter/Spring 1983, pp. 9-17.

[12] Chandrasekaran, B., "Expert Systems: Matching Techniques to Tasks", in *Artificial Intelligence Applications for Business*, Chapter 4, Ablex Publishing Corporation, Norwood, N.J., 1984, pp. 41-64.

[13] Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design", *IEEE Expert*, Fall 1986, pp. 23-30.

[14] Davis, R., and King, J., "An Overview of Production Systems", in E. Elcock and D. Michie (Eds.), *Machine Learning 8*, Chichester, England: Ellis Horwood, 1977, pp. 300-332.

[15] Deng, P.-S., Fuhr, C.L., "Using an Object-oriented Approach to the Development of a Relational Database Application System," *Information & Management*, Vol.29, 1995, pp.107-121.

[16] Dickson, G.W., DeSanctis, G. and McBride, D.J., "Understanding the Effectiveness of Computer Graphics for Decision Support: A Cumulative Experimental Approach", *Communications of the ACM,* Vol. 29, No. 1, 1985, pp. 40-47.

[17] Gray, P.M.D., Kulkarni, K.G., and Patron, N.W., *Object-Oriented Database: A Semantic Data Model Approach*, Prentice Hall, New York, 1992.

[18] Hanson, E., and Widom, J., "An Overview of Production Rules in Databases", *The Knowledge Engineering Review*, Vol.8, 1993, pp.121-143.

[19] Harmon, P., and King, D., *Expert Systems*, Wiley, 1985.

[20] Kim, W., "Object-Oriented Database Systems: Promises, Reality and Future", *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993, pp.676-692.

[21] Kim, W., *Introduction to Object-Oriented Databases*, The MIT Press, Cambridge, Massachusetts, 1990.

[22] Korson, T., and McGregor, J.D., "Understanding Object Oriented: A Unifying Paradigm", *Communications of the ACM,* 33 (9), September 1990, pp. 40-60.

[23] Kuipers, B.J., "A Frame for Frames: Representing Knowledge for Recognition", in *Representation and Understanding,* Bobrow, D.G., and Collins, A. (Eds.), Academic Press: New York, 1975.

[24] Low, G.C., Henderson-Sellers, B., Han, D., "Comparison of Object-oriented and Traditional Systems Development Issues in Distributed Environments," *Information & Management*, Vol.28, 1995, pp.327-340.

[25] Mahmood, M.A., Gowan, M.A., and Wang, S.-P., "Developing a Prototype Job Evaluation Expert System: A Compensation Management Application," *Information & Management*, Vol.29, 1995, pp.9-28.

[26] Minsky, M., "A Framework for Representing Knowledge", reprinted as Chapter 12 in *Readings in Knowledge Representation,* Brachman, R.J., and Levesque, H.J. (Eds.), Morgan Kaufmann Pub., 1985.

[27] Mockler, R.J., *Knowledge-based Systems for Management Decisions*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[28] Monarchi, D.E., "An Introduction to Object-Oriented Systems", presented at *International Conference on Information Systems*, Orlando, Florida, 6 December 1993.

[29] Nierstrasz, O.M., "What is the 'Object' in Object-oriented Programming", in D. Tsichritzis (Ed.), *Objects and Things,* Centre Universitaire d'Informatique, University of Geneva, March 1987, pp. 1-13.

[30] Parsaye, K., Chignell, M., Khoshafian, S., and Wong, H., *Intelligent Databases: Object-Oriented, Deductive Hypermedia Technologies*, John Wiley and Sons, Inc., New York, 1989.

[31] Partridge, D., *A New Guide to Artificial Intelligence*, Ablex Publishing Corporation, Norwood, New Jersey, 1991.

[32] Post, E.L., "Formal Reductions of the General Combinatorial Decision Problem", *The American Journal of Mathematics*, 65, 1943, pp. 197-268.

[33] Quillian, M.R., "Semantic Memory", in Minsky, M. (Ed.), *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968.

[34] Reichgelt, H., *Knowledge Representation: An AI Perspective*, Albex Publishing Corporation, 1991.

[35] Rich, E., and Knight, K., *Artificial Intelligence*, 2nd edition, Mc-Graw Hill, Inc., New York, 1991.

[36] Schank, R.C. and Abelson, R., *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum, Hillside, NY, 1977.

[37] Shadbolt, N., "Knowledge Representation in Man and Machine", in *Expert Systems: Principles and Cases*, Forsyth, R., (Ed.), Second Edition, Chapman Hill Computing, New York, 1989, pp. 142-170.

[38] Sowa, J.F., "Issues in Knowledge Representation", in Sowa, J.F. (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991, pp.1-11.

[39] Stefik, M., and Bobrow, D.G., "Object-Oriented Programming: Themes and Variations", *AI Magazine*, 1986.

[40] Sully, P., *Modeling the World with Objects*, Prentice Hall, New York, 1993.

[41] Thomas, D., and LaLonde, W., "Object-Oriented Programming and Systems", in Oren (Ed.), *Advances in Artificial Intelligence in Software Engineering*, 1990, pp.57-106.

[42] Villeneuve, A.O., and Fedorowicz, J., "Understanding Expertise in Information Systems Design, Or, What's All the Fuss About Objects?", *Decision Support Systems*, 1995.

[43] Wang, S., "Object-oriented Task Analysis," *Information & Management*, Vol.29, 1995, pp.331-341.