The Knowledge Engineering Review, Vol. 5: 1, 1990, 35–50

Efficient knowledge representation systems

DARIO GIUSE

The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract

Frame systems occupy an important place among formalisms for computer-based knowledge representation. A common concern about frame systems, however, is that they are not efficient enough. We argue that this is not necessarily true of all possible systems, and that the trade-off between generality and efficiency has not been fully explored. While many systems provide generality at the expense of performance, systems closer to the low end of the spectrum have not been investigated nearly as much. Those systems are well suited for applications that need flexible knowledge representation but cannot afford the high performance price.

We describe in detail KR, a very efficient frame system that provides mechanisms for knowledge representation including user-defined inheritance and relations, object-oriented programming, and constraint maintenance. The system is simple and compact and does not include some of the more complex functionality, but it is highly optimized and offers excellent performance for a variety of applications.

1 Introduction

Ever since the introduction of the programmed digital computer, the need for computers to manipulate information from the real world has been apparent. Any piece of software, no matter what the application domain, works with data that reflect some property of the outside world and produces results that affect the outside world. Over the years, a distinction has emerged between programs that manipulate regular, simple-structured information and programs that manipulate less structured information. Although the distinction is not always clear-cut, authors tend to use the term *knowledge* for the less structured kind of information. The problem of representing this information in a way that is best suited for use by a computer program is known as *knowledge representation.* While knowledge representation techniques are widely used in Artificial Intelligence (AI) programs, they are by no means restricted to AI applications.

Many techniques have been proposed that allow a representation of knowledge formalized enough for use by a computer program. The roots of all such techniques can be traced back to formal logic, which is itself used as the knowledge representation formalism in several computer-based systems.

Among the different schemes for knowledge representation, frame systems have been one of the most successful. The frame model provides a great deal of flexibility and, at least potentially, great representational power. Many frame systems have been developed over the years, and some of the more recent ones have made their way into commercial products. In spite of this wealth of work, however, the space of possibilities for frame-based representation has not been explored in its entirety. The lower end of the spectrum, in particular, has been covered only lightly. We claim that this region offers several important advantages of its own, and thus it is well worth investigating. Frame systems at the low end of the spectrum combine great simplicity with good

performance, a combination that makes them suitable for many applications for which the heavy-duty systems at the opposite end may not be appropriate.

The next section presents a brief overview of the principal knowledge representation formalisms, and positions frame systems within that general area. The following section introduces the distinction between low-end frame systems and the more usual high-end systems, and justifies the need for low-end frame systems. The remaining portion of the document describes a particular low-end frame system, named KR, and shows how its features form a natural complement to those of the more complex systems. Because of its efficiency, KR is ideally suited for a number of applications that require flexible representation of knowledge but cannot afford the performance overhead often associated with full-fledged knowledge representation systems.

2 Computers and knowledge representation

Different schools of thought, each embracing a particular formalism, have emerged in the arena of knowledge representation. It is not yet clear which one will become dominant, nor indeed whether one will suffice for all knowledge representation needs in the future. Sloman (1985), for example, makes the case that there will continue to exist a need for a variety of formalisms and techniques. Brachman & Levesque (1985) give a comprehensive collection of some of the most influential papers on the history of knowledge representation systems. That work also includes an extensive bibliography of papers in the knowledge representation area.

We now briefly introduce the main formalisms that have been proposed in the literature, and then concentrate on a specific one, i.e., frame systems.

2.1 Knowledge representation formalisms

The first, and best understood, formalism suited for representing knowledge in a computer program is *formal logic*. Much of the work on knowledge representation can actually be said to originate from formal logic. Since the very beginning logic, in one of its different formulations, has been regarded as a powerful tool for representing knowledge in a precise, rigorous way. First Order Logic, i.e., predicate calculus, has traditionally been the choice, but other variants have also been used which allow one to deal with temporal reasoning, default reasoning, and so on.

A number of programming systems have been developed which use some form of logic as their primary foundation. PROLOG (Clocksin & Mellish, 1981), for example, is a typical logic-based programming system and uses a restricted form of First Order formulas (i.e., Horn clauses) to express knowledge and procedural information.

Concerns about the inefficiency of full-fledged theorem provers, required by a full implementation of formal logic, have motivated researchers to explore other possibilities. *Production rules* (Davis et al., 1975) are a well-known such formalism. Production rules are used to represent knowledge in the so-called rule-based systems, a particular type of AI systems. A production rule is expressed in IF ... THEN form and specifies how the conclusion in its right-hand side can be inferred if the conditions in its left-hand side are met. Rule-based systems attempt to provide a reasonable balance between representational power and speed of execution. They account for the majority of expert systems developed in the last 15 years (Feigenbaum et al., 1988).

A third formalism for representing knowledge is known as *frame systems*. This formalism is also closely related to logic, as indicated by Genesereth and Nilsson (Genesereth & Nilsson, 1987). Knowledge is represented as a network of connected frames. Frame systems provide some basic inferencing capability which allows them to infer certain facts about a frame by its connections with other frames. Frame systems are the main focus of this paper and will be discussed at length in the following.

A fourth formalism for representing knowledge is known as *neural networks*. Neural networks (Fahlman et al., 1983; Hinton et al., 1986), originally inspired by theories of the behavior of the human central nervous system, have been used both as an attempt to model the human brain and as

a computational paradigm. Unlike formal logic and frame systems, neural networks place less emphasis on the manipulation of abstract symbols and more on the activity of making decisions based on input stimuli and the internal state of a richly interconnected network of relatively simple computing elements. This structure makes neural networks suitable for other tasks besides knowledge representation, such as vision (Feldman, 1985) and speech recognition.

Each of the formalisms described above has made important contributions to the field of computer-based representation of knowledge, and it would be beyond the scope of this work to debate their relative merits and importance. For practical reasons we will limit ourselves to only one of them, and specifically to frame systems. The remainder of the paper will thus deal exclusively with frame systems, and in particular with frame systems that occupy a special region in the power/performance space.

2.2 Frame systems

This section contains a brief survey of influential ideas and systems in the field of frame-based knowledge representation. While this is not a comprehensive survey, it is meant to set the stage for the following discussion of low-end knowledge representation systems. Note that some fo the systems mentioned below would more properly be named semantic networks, but since we are only interested in the gross features of such systems we will ignore the distinction for the remainder of this paper.

The origins of frame systems in their current form can be found in ideas first expressed by Minsky (1963), but can actually be traced much further back in the history of philosophy. For example, Anderson and Bower (1973) trace some of the ideas all the way back to Aristotle's associationism. Mac Randal (1988) provides a good historical introduction to the field of semantic networks and an in-depth discussion of the semantics of various frame systems.

The first computer implementation of ideas proposed by Minsky came with Quillian's PhD thesis in 1966, which was aimed at the problem of natural language understanding. Quillian (1967, 1968) used the term *semantic memory* to describe his formalism. Each node in the system corresponds to a linguistic concept (a *word concept*) and represented the "meaning" of a word. Quillan's formalism was actually quite general, in spite of the emphasis on natural language understanding. Other work soon followed, such as that by Shapiro (1977), who gave a model of semantic networks, and by Woods (1975), who emphasized the precise description of the semantics of nodes and links in semantic networks.

One of the best-known examples of frame-based system is KL-ONE, developed by Brachman (1977, 1983), which embodies many of the notions found in most modern semantic networks, KL-ONE has been used in the development of several systems, as well as for research into the semantics and representational adequacy of semantic networks. One of the primary goals in the development of the system was in fact to investigate the so-called *epistemological adequacy* of frame systems, i.e., what could and could not be represented by frame systems and what type of semantics should be built into them. KL-ONE includes a powerful set of features such as multiple inference strategies and the so-called *classifier*, which automates the placement of frames within a taxonomy and may also be used for inferencing. Much of the work on KL-ONE provides important clarifications on the role of frames and, especially, links in a semantic network. Brachman and Schmolze (1983) gives a comprehensive description of the system and the underlying philosophy.

Other papers by Brachman (1979, 1985), based on that author's experience with KL-ONE, exerted a fundamental influence on the development of modern frame systems. In those papers Brachman posed basic questions about the semantics of frame systems and clarified important issues which had often been muddled in previous systems. The 1979 paper (Brachman, 1979) also made a clear distinction among the different levels in frame systems, which helped eliminate some of the confusion in the field. Brachman identified the *implementation level*, the *logical level*, the *conceptual level*, and the *linguistic level*. Each level builds on the previous ones and provides a foundation for the next level up. That paper, together with the explicit objective of achieving an epistemologically

adequate knowledge representation, was among the key reasons for the strong influence KL-ONE has had on modern frame systems.

Many other frame-based systems have been influential in the development of the field. Examples are Roberts and Goldstein's FRL (1977), Bobrow and Winograd's KRL (1977), and Wilensky's KODIAK (1984). Wilensky also presents an interesting discussion (Wilensky, 1987) of some problems often found in knowledge representation systems.

Other well-known frame systems that have been developed more recently are part of the "current generation", which was influenced by KL-ONE and its derived systems. A typical example of the "current generation" is a tool such as SRL, developed by Fox et al. (1984), which includes a variety of user-definable customizations such as user-defined inheritance paths, relations, contexts, support for object-oriented programming, and different types of procedural attachment. As was the case for many AI tools designed within the last decade, SRL was later developed commercially to become CRL (Lieberman, 1986).

After this brief survey of the best-known frame-based knowledge representation systems, we will now introduce a characterization of the power and efficiency of such systems. We will argue that the spectrum has not been covered uniformly, and that frame systems provide ample opportunity for simplicity and efficiency.

3 Frame systems: high-end versus low-end

This section identifies two broad areas within the spectrum of possible frame systems: *high-end* versus *low-end*. It will be seen that both ends of the spectrum provide useful features, and that a better understanding of the trade-offs among the two types of systems may greatly improve the usefulness of knowledge representation systems.

3.1 High-end systems

Most frame systems described in the literature have emphasized powerful representation of knowledge and powerful inferencing, possibly at the expense of performance. The reasons for this bias toward the high end have been several.

In some cases the choice was dictated by the desire to explore the epistemological possibilities of particular mechanisms. In others, it was dictated by the desire to serve the needs of complex AI application programs. Yet another factor has been the desire to provide enough generality to accommodate widely different applications. Finally, part of the reason has undoubtedly been, in some cases, the designers' attitude to include a feature simply because "someone might find a use for it".

High-end systems of this type have made significant contributions to the field of computer-based knowledge representation, since they have shown the full potential of frame-based representation. The high end of the spectrum has been, and will continue to be, the place where most advances in conceptualizing the role of knowledge representation have taken place.

The reverse side of the coin is that high-end systems tend to be very complex, and their performance is not always adequate. One should draw a clear distinction between representation problems that are inherently hard, and therefore require complex solutions, and problems that are inherently simple. While it is acceptable for solutions to complex problems to be computationally expensive, it is unfortunately the case that high-end systems often exhibit poor performance on simple problems. The most common performance problems fall into two categories.

The first category stems from the support necessary for the more complex representation issues. Many systems that provide fully general support for features such as meta-knowledge and alternative world reasoning fail to optimize the simple cases which so often occur in many applications. Very simple and very complex problems are all treated alike, which means that simple problems suffer from an undue performance penalty.

The second type of problem is not specific to knowledge representation systems, but is directly

related to the total size of the system. As all software systems, high-end frame systems are very large and complex artifacts. As such, they often grow by accretion of new features and layers on top of existing ones. Systems built in such a way are likely to become difficult to maintain, let alone restructure. Most high-end systems quickly reach a point where performance improvements are no longer practical.

A final, well-known problem with most high-end systems is that they are monolithic environments which make it virtually impossible to scale down application programs for smaller hardware platforms. Moving an application program originally developed in a high-end system from development environment to delivery environment sometimes requires rewriting the entire application.

3.2 Low-end systems

The low end of the the spectrum of frame systems has not been explored nearly as much as the high end. This is partially understandable, given the need for powerful representation mechanisms that many researchers have felt. Many important practical applications of frame systems, however, have been precluded by this apparent lack of interest in efficient knowledge representation.

Low-end systems cannot compete with high-end ones on the basis of features and (possibly) representational power. They can, however, provide three advantages over the more conventional high end of the spectrum. These advantages justify the need for more research in the low-end area.

The first, and more obvious, advantage is *efficiency*. If properly designed, a low-end knowledge representation system can offer superior performance, often approaching that of conventional data structures. The later sections of this paper present an actual example of a knowledge representation system, named KR, which substantiates this claim. Note that by efficiency we mean not only runtime efficiency, but also storage efficiency: a properly designed low-end system will use a smaller amount of storage than a corresponding high-end system.

The second advantage is *simplicity*. A low-end frame system typically offers only a fraction of the functionality found near the high end of the spectrum, and therefore it is much easier to understand and use. Equally important, it is easier to debug and maintain, which generally means that it is also more robust. While these two characteristics do not necessarily go hand in hand, it is certainly true that most high-end systems suffer from all the software engineering woes common to large systems.

The third advantage is *flexibility*. The kernel of a low-end knowledge representation system is very small and thus very flexible. A low-end system may be the ideal environment for studying the relative impact of different knowledge representation featues, since its flexibility enables each feature to be implemented and evaluated separately. One can regard a low-end system as the ideal workbench on which to carry out such experiments as determining the relative impact of a context mechanism, for instance, or of multiple inheritance.

The following sections describe in detail a low-end frame-based knowledge representation system named KR. We will present an overview of KR and position it within the spectrum of frame systems. We will then present some data which describe the performance of the system, and examples of applications which use it as their sole form of knowledge representation.

4 The KR system

KR is a knowledge representation system implemented in Common Lisp (Steele, 1984). It follows in the tradition of frame systems, but it has a distinct position within the low end of the spectrum. KR shares this area with very few other frame systems; the most noteworthy of these is FrameKit, described by Nyberd (1988), which also provides a simplified knowledge representation framework. As we will see, however, KR is more aggressive than FrameKit, both in terms of simplicity and of performance.

The historical derivation of KR is primarily from systems such as SRL and CRL. Some of the terminology is also derived from those systems: a frame, for example, is referred to as a *schema*, KR

was born out of our desire to retain the kernel functionality of SRL and CRL while at the same time providing a more economical system.

4.1 Structure of the system

KR includes three, closely integrated components: frame-based knowledge representation, object-oriented programming, and consraint maintenance.

The first component, *frame-based knowledge representation*, stores knowledge as a network of schemata. Each schema can store an arbitrary amount of information through any number of *slots*, which behave as attribute/value pairs. The slot name indicates the attribute name; the slot values (if any) indicate its values. Values are the actual data items stored in the schema, and may be objects of any LISP type. As in most frame systems, values in a schema can be interpreted as links to other schemata. This enables the system to support complex network structures which can be freely extended and modified by application programs.

The second component of KR is a simple *object-oriented programming* system. Schemata can be used as objects, and inheritance can be used to determine their properties and behavior. Objects can be sent *messages*, which are implemented as procedural attachments to certain slots; messages are inherited through the same mechanism as values. Instead of the class-instance paradigm, common in object-oriented programming languages, KR follows the more flexible prototype-instance paradigm (Liekerman, 1986), which allows properties of instances to be determined dynamically by their prototypes.

Finally, the third component of KR implements *constraint maintenance*. This component is logically independent of the first two, which may in fact be used stand-alone. It allows a value in a slot to be constrained by other values. More specifically, a value can be computed as a function of other values and modified whenever any of those values change. Constraint maintenance is closely integrated with the underlying knowledge representation, and for most users the distinction between the two is irrelevant. The user, for example, does not need to know which slots in a schema contain ordinary values and which ones are constrained by a formula, since the same access primitives may be used in both cases.

4.2 Frame representation

This section briefly presents the highlights of frame representation in KR. Please refer to the KR User's Manual (Giuse, 1987; Myers et al., 1989) for more details on this and the other components of the system.

4.2.1 Inheritance

Inheritance is widely used in KR and achieves three purposes: it reduces network size, it helps maintain consistency, and it allows local knowledge to override global knowledge. Inheritance reduces network size because whenever a piece of information for a schema is the same as in a more general one, the information need only appear once. It also helps maintain consistency, because it enables a change to a value in one place to be immediately visible everywhere else in the network. Finally, inheritance allows local redefinition of global knowledge, since a local value simply overrides any value which might otherwise be inherited. These three features are illustrated in Figure 1, which shows their effects on a very simple network.

Inheritance in KR is user-defined. In addition to a handful of predefined relations that perform inheritance (of which IS-A is the most typical example), application programs are free to define new inheritance relations. Any user-defined inheritance relation may or may not have inverse relations. New relations may be defined in order and at any time during execution.

KR fully supports multiple inheritance, i.e., the situation where a schema inherits values from more than one ancestor. This can be accomplished in two ways. The first way is simply to connect

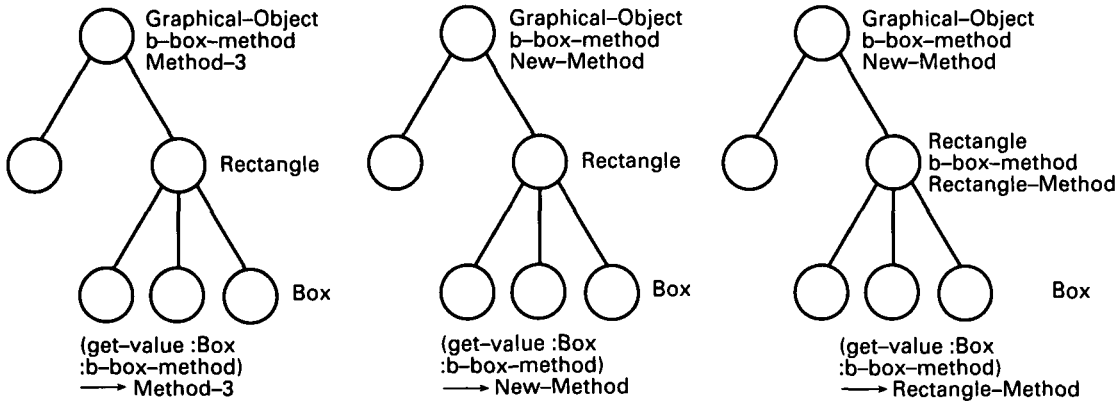

Figure 1 Three effects of inheritance

the schema to more than one ancestor schema through an inheritance relation (see below for the meaning of this term). An inheritance relation slot, in other words, may contain multiple values. The second way to achieve multiple inheritance is by using more than one inheritance relation. Any schema may have several slots defined as inheritance relations; in this case, all relations are searched in turn until a value is found. The two mechanisms may be combined, of course.

Several kinds of systems provide some form of inheritance. Besides frames systems, for example, object-oriented systems such as Smalltalk (1981) and CLOS, the Common Lisp Object System (Davis et al., 1975), support some form of inheritance. Many existing systems, however, suffer from limitations in either the power of inheritance or its performance. KR uses a unique mechanism which enables inheritance to behave in a completely dynamic fashion, just like frame systems, while providing performance comparable to that of much simpler systems. This mechanism is called *eager inheritance.*

Eager inheritance works as follows. The first time the value of a slot is requested, but the value is not present locally, the value is obtained by inheritance as described above. At this point, however, the value is also copied into the local schema (and in any intervening schema, if necessary) with a special marker which indicates that the value was inherited. The second time the value is requested, inheritance is no longer required and the value is immediately found locally. This makes successive accesses to inherited values much faster, and causes inheritance to be essentially as efficient as local values, no matter how many levels of inheritance were originally used.

It is vital for inherited values which were copied down into children schemata to be kept up to date. Any change in the upper portions of the schema hierarchy might change what values can be inherited by the lower levels, and inherited values which were copied down must be modified. KR performs this task immediately when a value which was inherited is changed, thus justifying the term *eager inheritance.* This technique ensures minimal overhead for both access and update of inherited values, and provides superior inheritance performance.

4.2.2 Relations

Slots such as :IS-A which enable knowledge to be inherited from other parts of a network are called *inheritance relations.* Inheritance along a relation is typically defined to proceed depth-first and may include any number of steps; in other words, the search terminates if a value is found or if no other schema can be reached via the relation. KR allows the user to define new inheritance relations as desired.

Any relation, including user-defined ones, may also be declared to have an inverse relation. When this is the case, KR automatically generates an inverse link any time the relation is used to connect one schema to another. Imagine, for instance, that we defined :PART-OF to be a relation having :HAS-PARTS as its inverse. Adding schema A to the slot :PART-OF of schema B would automatically add B to the slot :HAS-PARTS of schema A, thereby creating an inverse link. KR automatically maintains all

relations and inverse relations. This ensures that the state of the knowledge representation system is completely consistent at any point in time, independent of the particular sequence of operations.

4.3 Object-oriented programming

The object-oriented programming component of KR is fairly straightforward and implements two concepts: the concept of message sending, and the concept of prototype/instance.

An object in KR is simply a schema. Objects consist of data (represented by values in slots) and methods (represented by procedural attachments, again stored as values in slots). Procedural attachments are invoked by "sending a message" to an object; this means that a method by the appropriate name is sought ane executed. Different types of objects often provide different methods by the same name; this ensures that the same message may be sent to different objects, which respond by performing different actions.

Both the data and the methods associated with an object can be either stored within the object or inherited. The usual inheritance rules are followed, including of course multiple inheritance. This allows the behavior of objects to be built up from that of other objects; it is possible, in particular, to create complex graphs of method inheritance. The object-oriented component of KR allows some combination of methods, but this feature is not as fully developed as in full-fledged object-oriented systems such as CLOS (Bobrow et al., 1989).

The notion of *prototype* in KR is superficially similar to that of class in conventional object-oriented programming languages. A prototype object can be used to partially determine the behavior of other objects (its *instances*). A prototype, however, plays a less restricting role than a class, since it simply provides a place from which the values of certain slots *may* be inherited. The number and types of slots and methods which actually appear in an instance may well differ from that in its prototype. Prototypes in KR serve two specific functions: they provide an initialization method, and they provide default constraints which may be inherited by their instances.

4.4 Constraint maintenance

This section describes the constraint maintenance component of KR. Unlike other frame-based systems, constraint maintenance is an integral part of KR and is tightly integrated with the basic knowledge representation.

The KR constraint system offers two distinct mechanisms to cause changes in a part of network to propagate to other parts of the network. The first mechanism is *value propagation* and ensures that the network is constantly kept in a consistent state. The second mechanism is *action propagation*, allowing an application program to cause certain actions to be triggered when parts of a network are modified.

The constraint maintenance system ensures that whenever a value in a slot in a slot is changed, all slots whose values depend on its are immediately invalidated, although not necessary re-evaluated. This is what we refer to as value propagation. This strategy does not immediately recompute the value in the dependent slot, and thus it typically does less work than an eager re-evaluation strategy. The system simply guarantees that the correct values are recomputed when actually needed, thus yielding the same results as eager evaluation.

A second mechanism is also provided which allows an application program to perform special actions when a value is modified. We refer to this as action propagation. Action propagation is implemented through *demons*. A demon is an application-defined procedural attachment to a KR slot. Whenever a value of a slot in a schema is modified (either directly or as the result of value propagation), KR checks whether a demon is defined for that particular schema and slot. If so, the demon is invoked. This allows application programs to attach a certain behavior to their schemata and be notified every time a change occurs.

Constraints are implemented by *formulas*, arbitrary Lisp expressions which in general contain at least one reference to a particular KR value. The Lisp expression is used to recompute a value

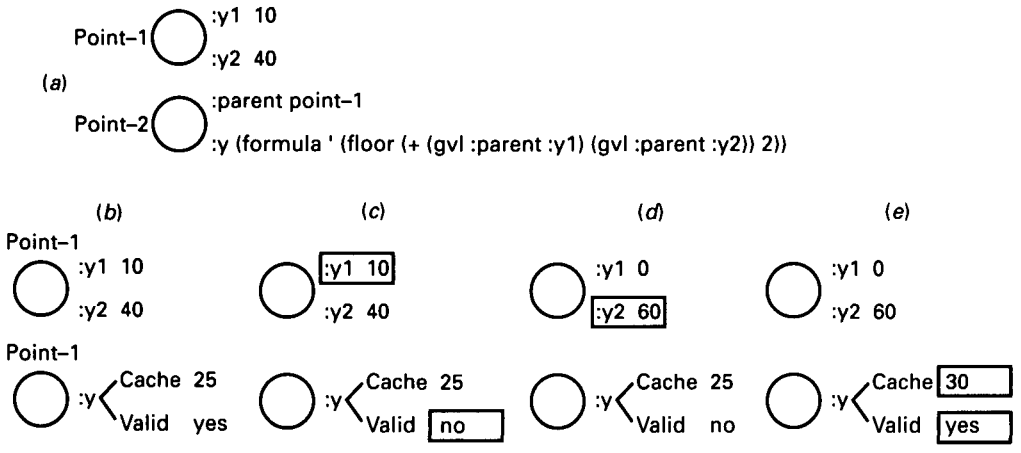

Figure 2 Successive changes in depended values

whenever a change in one of the depended values makes it necessary. Note that formulas are not recomputed every time their value is accessed; each formula, instead, keeps a cache of the last value it computed. Unless the formula actually needs to be recomputed, the cached value is simply reused. This causes a dramatic improvement in the performance of the constraint maintenance system.

Figure 2, part (**a**), shows an example of a formula installed on slot :Y of schema :POINT-2. The formula depends on two values, i.e., the value of slots :Y1 and :Y2 in schema :POINT-1. The formula, in particular, specifies that slot :Y is constrained to be the sum of the two values divided by 2, i.e., the average of the two values. Figure 2, part (**b**), shows the internal state of the formula in a steady-state situation where the formula has been evaluated and contains a valid cached value. Under these circumstances, any request for the value of slot :Y would simply return the cached value, without recomputing the formula.

Parts (**c**) and (**d**) show the effects of changes to the depended values. Changes are illustrated by small rectangles surrounding the modified information. The first change is to slot :Y1 and causes the value in the formula to be marked invalid. Note that the formula is not actually recomputed at this point, and the cached value is left untouched. The second change is to slot :Y2 and does not cause any action to take place, since the formula is already marked invalid.

Finally, part (**e**) shows what happens when the value in slot :Y is needed at least. The value of the formula is recomputed and again cached locally; the cache is marked as valid. The system is then back to a steady state. Note that the formula was recomputed only once, when needed, rather than eagerly after each value change.

5 The advantage of simplicity

Simplicity was an important criterion in the design of KR. This section describes some results of this choice and explains why low-level frame systems are especially suitable for experimentation.

One of the central design decisions in KR was to follow the philosophy of the underlying language as closely as possible. This decision provided a radical simplification of the interface and of the system itself. Rather than building up a monolithic system, complete unto itself, KR simply extends the LISP language. Functions that can be expressed in LISP are never duplicated, and the system only implements the lowest levels of knowledge representation (corresponding to Brachman's *implementation level* and parts of the *logical level*). For example, KR does not provide any operation to add values to the front of a slot. This is achieved through a LISP primitive, since KR provides a generalized SETF form for slots*. As another example, searching for a given value in a slot

* LISP generalizes the notion of variable assignment to that of assignment to an arbitrary **place**. A generalized place for assignment could be an element of an array, a structure reference, or any part of a data structure. SETF is the generalized assignment operator. Once a generalized place has a SETF form, it can participate in any modifying operation exactly as a variable could. Forms for incrementing, pushing, shifting, etc, any generalized place are provided automatically.

is achieved simply through LISP sequence operators, since KR slots are defined as sequences of values.

Besides simplifying the interface, relying on underlying primitives greatly reduces the burden on users and application programmers, who are not forced to learn yet another syntax for operations that are already supported by the underlying programming language.

5.1 Implementation in other programming languages

The KR model of knowledge representation is so simple that it lends itself well to implementation in programming languages simpler than LISP. This would undoubtedly be difficult for high-end systems such as KL-ONE, whose faithful re-implementation in a non-LISP language would present major engineering difficulties.

We are currently completing an experimental version of KR written in the C programming language. This version is an experiment in moving some of our ideas about low-end knowledge representation systems into different programming environments. Preliminary results suggest that this system may provide a viable alternative to more static means of knowledge representation for languages such as C. Performance, in particular, seems to compare very favorably with that of more traditional C data structures and with C-based object-oriented languages such as C++.

While this experiment is not complete, it shows the potential of this approach for migrating frame systems down to hardware platforms which do not easily support LISP environments. The programming environment provided by this experimental system is virtually identical to that found in KR, given the obvious variations in style imposed by differences between LISP and C.

5.2 Exploring the cost of orthogonal features

An important advantage of KR's simplicity of design is that it enables experimentation on different features. Some of this experimentation would be very difficult with many high-end frame systems.

A line of research we find particularly intriguing is the evaluation of the *relative cost of different knowledge representation features.* Since so little of the more complex features of high-end systems is hard-wired in KR, it is possible to evaluate each feature independently. This can be achieved easily with the assurance that the experiment will not be perturbed by other aspects of the system.

The best example to date of this research was the implementation of the constraint system. When we started working on constraint maintenance in KR, we were initially simply interested in evaluating the feasibility and the cost of this feature. Constraint maintenance, which started as an isolated experiment, later became an integral part of the system itself.

The design goals of the original system, which was closely modeled after the Coral system developed by Szekely and Myers (1988), were:

- Total transparency—access to slots with constraints should be indistinguishable from access to slots without constraints.
- Lazy evaluation—most operations on constraints should be delayed as much as possible.
- Consistency—constraints maintenance should be automatic and always keep the network in a consistent state.
- Excellent performance—operations involving constraints should be comparable to ordinary operations.

We are able to achieve the above goals with a separate layer built on top of KR, without any modification to the system itself. Constraints were implemented as formulas, which were lazily evaluated on demand. The outcome of the experiment was very gratifying and produced an actual running system. Based on the results of the experiment, we later decided to integrate constraint maintenance with KR.

The entire constraint layer took less than two weeks to implement, debug, and optimize. We are convinced that such a rapid implementation would have been very difficult in a different

environment. To substantiate his claim, consider that Coral, the system which served as the model for ours, took more than two months to implement in Common Lisp using CLOS, the Common Lisp Object System (Bobrow et al., 1985). Most of the difficulties in the implementation were due to the static nature of the class-instance relationship in CLOS, which is typical of most of the object-oriented programming systems.

The KR implementation, on the other hand, took full advantage of the highly dynamic characteristics of frame systems and allowed us great freedom during the exploratory stages of the implementation. Interestingly enough, the KR implementation ended up being significantly more efficient than the particular version of CLOS we were using. This comparison should clearly be taken with a grain of salt, since the particular version of CLOS on which it is based is a public-domain, portable implementation that is not fine-tuned for the hardware we were running it on. On the other hand, KR is also completely portable and is not fine-tuned for any particular hardware. We feel that the main obstacle to good performance in the CLOS implementation was the underlying complexity: By the time the CLOS implementation reached the intended functionality, it was complex enough to make optimization an unpleasant task.

6 Evaluation

We now present an evaluation of KR along three different dimensions: performance, complexity, and support for applications. We will characterize KR by comparing it to other frame systems and to other programming paradigms, such as conventional data structures and objected-oriented programming systems.

6.1 Performance

We begin this section by showing a short table with performance figures for KR in Common Lisp (see Table 1). These figures were collected on an IBM RT/PC running CMU Common Lisp (McDonald, 1987) under the Mach operating system (Rashid, 1986). All figures refer to compiled code and are expressed in microseconds per call.

The table indicates that KR performs quite well. To put those figures in perspective, consider that an empty function call and return in the same environment takes about 10 μs. The time to execute the simplest and most commonly used access functions, G-VALUE and GET-VALUES, is of the order of less than 2 function calls.

Note that accessing a value from a schema through 1 and 2 levels of inheritance is essentially as efficient as accessing a value which is present locally, as shown by the three entries for G-VALUE. This is possible because of eager inheritance, as explained previously. The small difference between local and inherited access is due to the cost of the very first access, which is amortized over all successive calls.

Table 1 Performance figures for some primitive functions. The first three functions in the table deal exclusively with knowledge representation. G-VALUE is the equivalent function for constraint maintenance; it uses formulas when present. S-VALUE is the value-setting function; it also uses formulas when needed

Function	*Explanation*	*Levels of inheritance*	*Execution time*
GET-LOCAL-VALUE	access, no inheritance	0	5.6 μs
GET-VALUE	access a single value	0	18.9 μs
GET-VALUES	access multiple values	0	18.5 μs
G-VALUE	uses formulas	0	25.6 μs
G-VALUE	uses formulas	1	26.6 μs
G-VALUE	uses formulas	2	26.6 μs
S-VALUE	set the value in a slot	0	49.3 μs

Table 2 Execution times (in μs) for various representation systems (IBM RT/PC)

	LISP	*KR*	*FRAMEKIT*	*CRL*
GET-VALUE (1 0)	5.7	18.9	493.0 (*80 bytes*)	118.0
GET-VALUE (4 0)	5.7	31.0	518.7 (*80 bytes*)	142.6
GET-VALUE (1 1)	n/a	18.9	2090.0 (*240 bytes*)	417.3
GET-VALUE (1 4)	n/a	18.9	3962.5 (*576 bytes*)	1191.0
SET-VALUE (1 0)	10.0	49.3	4921.8 (*576 bytes*)	465.3

Table 2 compares the execution times for some of the most common functions in different representation systems. Each entry in the table indicates the actual execution time in microseconds. When the operation generates temporary storage (garbage), this is indicated by a second number in parentheses which shows the number of bytes allocated per operation.

Each row begins with the name of the operation and two numbers. The first number indicates the position of the slot, which is important since in most frame systems the access time is position-dependent. The second number determines the levels of inheritance; 0 means that the slot is local and no inheritance is used. The second row, for example, shows the execution time for retrieving the fourth slot locally (i.e., with 0 levels of inheritance). The fourth row shows the execution time for retrieving the first slot with 4 levels of inheritance, i.e., from a schema 4 levels up in the hierarchy.

The column labeled "LISP" shows figures for a straightforward implementation where a LISP structure was used to simulate conventional data structure access. These figures give a lower bound for the access time. Times in the "LISP" column show the slot access time (which is position-independent) and the slot update time. Inheritance is, of course, not applicable to this simple data structure.

The table clearly shows the large difference in performance among different systems. Whereas the cost of a GET-VALUE operation in KR is only 3.3 times that of the fastest possible LISP access, the cost becomes 20.7 times in CRL. The performance ratio between CRL (a reasonably optimized high-end system) and KR is around 6.2 for the simplest operations and up to 60 times for operations with inheritance.

The figures for FrameKit deserve a little explanation. While FrameKit is a low-end system, as shown by its reduced functional interface (see Table 3), its performance is actually significantly worse than that of CRL. This is because at the time these figures were collected FrameKit had not undergone any optimization, and thus it suffered from all the usual overhead for the simple cases. Careful optimization should bring the performance to more acceptable levels.

6.2 Complexity

We will now present a simple measure of the complexity of KR versus two other frame systems. Table 3 shows the number of functions and special variables in the program interface of three frame systems. The table only displays the interface for the frame-based representation and object-oriented component of each system, since neither FrameKit nor CRL contain a constraint maintenance component.

KR and FrameKit belong to the low end of the spectrum, whereas CRL is a commercial, high-end system. The table indicates the relative complexity of the programming interface of each system by giving a count of all identifiers exported to the application programmer. As the table shows, the difference between low-end and high-end systems is quite marked. The CRL interface contains a total of 139 exported symbols (103 functions and 36 special variables), whereas KR only has 28. Most CRL functions, moreover, has a fairly complex structure and take optional arguments and keyword arguments. Learning such as complex interface is a rather challenging task. Note that FrameKit, a low-end frame system, is much closer to KR in pursing a streamlined interface.

Table 3 Complexity of the functional interface for different frame systems

FUNCTION GROUP:	*KR*	*FRAMEKIT*	*CRL*
create schema, slot	4	11	16
access values	6	12	33
update values	5	11	19
delete schema, slot	2	6	11
information	7	8	8
debugging, tracing	2	3	2
object-oriented	2	1	14
miscellaneous	0	4	0
TOTAL	28	43	103
special variables	0	13	36

6.3 Applications

Several application programs use KR for knowledge representation, object-oriented programming, constraint maintenance, or a combination of the above. Some of these applications use a mixture of KR and conventional data structures, whereas others rely entirely on KR.

The first application we developed using KR was the Chinese Tutor (Giuse, 1988a,b), an intelligent tutoring system designed to teach Chinese to English speakers. The Chinese Tutor uses KR as its sole form of knowledge representation. KR is used, among other things, to store the online dictionary of Chinese characters and words, complete with English translations and various structural hierarchies. The evaluation of the student's familiarity with the language and a simple system of rules is also stored in a network of KR schemata.

The major application of KR is the Garnet User Interface Development Environment (Myers, 1988; Myers et al., 1989), developed at the School of Computer Science of Carnegie Mellon University. KR is the underlying layer for the entire Garnet system and is used to implement a graphical object system, a package to specify the behavior of graphical interactors, and a user-interface editing toolkit (Myers et al., 1990). The system was recently released to the Common Lisp community as a general-purpose user interface development environment.

In addition to these applications, other groups at Carnegie Mellon (especially groups engaged in speech understanding research, such as the MINDS project (Young et al., 1989) are actively using KR. Users have mentioned simplicity and excellent performance as the deciding factors which motivated them to select KR for their knowledge representation needs.

7 Future work

Important research topics remain to be addressed in the area of frame systems. Some of these topics can be best addressed within the framework of low-end frame systems such as KR. One such topic, as we already mentioned, is the careful evaluation of the impact of different features on performance and complexity. While high-end frame systems offer an all-or-nothing choice, systems at the low end of the spectrum show the potential for careful, independent evaluation of each feature. Analyzing each feature in isolation will provide designers and users with realistic data about expected cost and benefits.

We have already performed one such experiment, during which we added support for constraint satisfaction and evaluated its effect on implementation and performance. Similar questions that will have to be addressed in future research concern the impact of:

- Alternative world reasoning (*contexts*) capabilities.
- User-defined inheritance paths and search strategies (such as breadth-first versus depth-first).
- Used-defined slot and value restrictions and, more generally, *meta-knowledge.*
- Separate inference mechanisms, such as those available in KL-ONE.

Based on the results of these evaluations, it may become possible to build "knowledge representation toolkits" from which users could select exactly the features they need. While some amount of customization will always be required in order to assemble a practical frame system out of these features, this approach is likely to produce systems that are easier to customize as needed. This would open new possibilities for the significant problem of converting existing AI systems and applications to smaller software environments and hardware platforms.

A second interesting area of research which becomes possible within low-end knowledge representation systems is that of aggressive storage reduction. For applications that need extremely large knowledge bases this may well be the dominant factor in the choice of a knowledge representation system. The great simplicity of low-end systems makes them ideal candidates for research on aggresive storage reduction techniques, possibly by trading off flexibility for compactness. This is equivalent to "compiling out" much of the dynamic information in order to yield a maximally compact data representation.

Finally, low-end frame systems are ideal candidates for research into non-LISP implementations. Simplicity becomes a fundamental benefit when working with languages such as C which do not provide the rich data structure manipulation capabilities and the homogeneous treatment of program as data found in LISP. Low-end systems may provide the right combination of flexibility and performance needed by many practical application programs. While we have begun a limited research in this direction, much work remains to be done before the possibilities of this approach are exhausted.

8 Conclusions

Research on frame-based knowledge representation has been mostly concentrated on the high end of the spectrum and has emphasized representational power and flexibility over performance. While this direction of research has yielded important theoretical and practical progress, it is unfortunate that the reduced emphasis on performance has originated the myth that frame systems are only appropriate for large, complex AI applications.

It is time to correct this problem by placing equal significance on the issues of simplicity and efficiency. The domain where such issues can best be addressed is that of low-end frame systems, which emphasize economy while still retaining most of the flexibility of frame representation. Besides resulting in useful knowledge representation systems, we have argued that this line of research will actually yield a better understanding of the relative costs of different knowledge representation features which are currently lumped together in high-end frame systems.

In this article we have presented a low-end frame-based knowledge respresentation system named KR and have used it to show the main differences among the two types of frame systems. KR is a simple, very efficient knowledge representation system which implements the basic paradigm of frame-based knowledge representation. The program interface to the system consists of a small number of carefully tuned functions that are easy to understand and to use.

Low-end frame systems like KR provide a natural complement to the more traditional high-end ones. This region of the power/performance spectrum shows an amount of flexibility and efficiency that is not usually found in the other regions. Besides being useful systems in their own right, low-end frame systems should also act as an incentive for implementors of any knowledge representation mechanism to regard simplicity and performance as essential features that cannot be sacrificed.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), DARPA Order No. 4976, under contract number F33615-87-C-1499, monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Divison (AFSC), Wright Patterson AFB, Ohio 45433-6543. The views and conclusions are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

References

Anderson, JR and Bower, GH, 1973. *Human Associative Memory*, Holt, New York.

Bobrow, DG and Winograd, T, 1977. "An overview of KRL, a knowledge representation language" *Cognitive Science* **1**(1) 3–46.

Bobrow, D et al., 1985. 'Common Loops: merging Common Lisp and object-oriented programming" *9th International Joint Conference on Artificial Intelligence.*

Bobrow, DG, DeMichiel, LG, Gabriel, RP, Keene, SE, Kiczales, G and Moon, DA, 1989. "Common lisp object system specification" *Lisp and Symbolic Computation* **1**(3/4) 245–394.

Brachman, RJ, 1977. "A structural paradigm for representing knowledge" PhD Thesis, Harvard University, Cambridge, MA.

Brachman, RJ, 1979. "On the epistemological status of semantic networks" In: *Associative Networks: Representation and Use of Knowledge by Computers*, Academic Press, New York, pp. 3–50.

Brachman, RJ, 1985. "'I lied about the trees', or, defauls and definitions in knowledge representation" *The AI Magazine Fall*, 80–92.

Brachman, RJ and Levesque, HJ, 1985. *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, CA.

Brachman, RJ and Schmolze, JG, 1983. "An overview of the KL-ONE knowledge representation system" *Cognitive Science*, **9**(2) 170–216.

Clocksin, WF and Mellish, CS, 1981. *Programming in Prolog*, Springer-Verlag, Berlin, New York.

Davis, R, Buchanan, B and Shortliffe, E, 1975. *Production Rules as a Representation for a Knowledge-Based Consultation Program. Tech. Rept. STAN-CS-75-591*, Stanford Artificial Intelligence Laboratory.

DeMichiel, LG, 1989. "Overview: the common lisp object system" *LISP and Symbolic Computation*, **1**(3/4) 227–244.

Fahlman, SE, Hinton, GE and Sejnowski, TJ, 1983. "Massively Parallel Architectures for AI: NETL, Thistle, and Boltzmann Machines" *Proceedings of the AAAI-83 Conference*, Washington, DC.

Feigenbaum, EA, McCorduck, P and Nii, HP, 1988. *The Rise of the Expert Company*, Times Books.

Feldman, JA, 1985. "Connectionist models and parallelism in high level vision" *Computer Visions, Graphics, and Image Processing* **31** 178–200.

Fox, MS, Wright, JM and Adam, D, 1984. "Experiences with SRL: an analysis of a frame-based knowledge representation" *First International Workshop on Export Database Systems.*

Genesereth, MR and Nilsson, NJ, 1987. *Logical foundations of artifical intelligence*, Morgan Kaufmann, Los Altos, CA.

Giuse, D, 1987. *KR: an Efficient Knowledge Representqtion System. Tech. Rept. CMU-RI-TR-87-23*, The Robotics Institute, Carnegie-Mellon University.

Giuse, D, 1988a. "LISP as a rapid prototyping environment: the Chinese Tutor" *LISP and Symbolic Computation* **1**(2) 165–184.

Giuse, D, 1988b. "Intelligent tutoring systems for foreign language acquisition" *Proceedings of the Asia–Pacific Conference on Computer Education (APCCE 88)*, Shanghai, China, pp. 33–58.

Hinton, GE, McClelland, JL and Rumelhart, DE, 1986. "Distributed representations" In *Parallel Distributed Processing*, Bradford Books, Cambridge, MA.

Knowledge Craft Reference Manual, 1986. Carnegie Group, Inc., Pittsburgh, PA.

Lieberman, H, 1986. "Using prototypical objects to implement shared behavior in object oriented systems" *Sigplan Notices* **21**(11) 214–223. *ACM Conference on Object-Oriented Programming Systems Languages and Applications; OOPSLA '86.*

Mac Randal, D, 1988. "Semantic networks" In *Approaches to Knowledge Representation: An Introduction*, Research Studies Press, Forest Grove, OR, pp. 45–79.

McDonald, DB, 1987. *CMU Common Lisp User's Manual – Mach/IBM RT PC Edition. Tech. Rept. CMU-CS-87-156*, Computer Science Department, Carnegie-Mellon University.

Minsky, M, 1963. *Computers and Thought*, McGraw Hill, New York.

Myers, BA, 1988. *The Garnet User Interface Development Environment: A Proposal. Tech. Rept. CMU-CS-88-153*, Carnegie-Mellon University.

Myers, BA, Giuse, D, Dannenberg, RB, Vander Zanden, B, Kosbie, D, Marchal, P, Pervin, E and Kolojejchick, JA, 1989. *The Garnet Toolkit Reference Manuals: Support for Highly-Interactive, Graphical User Interfaces in Lisp. Tech. Rept. CMU-CS-89-196*, Carnegie Mellon University Computer Science Department.

Myers, BA, Vander Zanden, B and Dannenberg, RB, 1990. "Creating graphical objects by demonstration", submitted for publication.

Nyberg, EH, 1988. *The FrameKit user's guide. Center for Machine Translation*, Carnegie Mellon University, March.

Quillian, MR, 1967. "Word concepts: a theory and simulation of some basic semantic capabilities" *Behavioral Science* **12** (1967).

Quillian, MR, 1968. "Semantic memory" In *Semantic Information Processing*, The MIT Press, Cambridge, MA, pp. 227–270.

Rashid, RF, Accetta, M, Baron, R, Bolosky, W, Golub, D, Tevanian, A and Young, MW, 1986. "Mach: a new kernel foundation for UNIX development" *Proceedings of Summer Usenix.*

Roberts, RB and Goldstein, IP, 1977. *The FRL Primer. Tech. Reot. 408*, MIT AI Lab.

Shapiro, SC, 1977. "Representing and locating deduction rules in a semantic network" *SIGART Newsletter* **63** 14–18.

Sloman, A, 1985. "Why we need many knowledge representation formalisms" *Tech. Rept. Cognitive Studies Research Papers, CSRP 052*, School of Social Sciences, University of Sussex.

Steele, GL, 1984. *Common LISP—The Language*, Digital Press, Burlington, MA.

Szekely, PA and Myers, BA, 1988. "A user interface toolkit based on graphical objects and constraints" *Sigplan Notices* **23**(11) 36–45.

Tesler, L, 1981. "The smalltalk environment" *BYTE* **8** 90–147.

Wilensky, R, 1984. "KODIAK: A knowledge representation language" *Proc. of the Sixth Annual Conference of the Cognitive Science Society*, Boulder, CO, pp. 344–353.

Wilensky, R, 1987. "Some problems and proposals for knowledge representation" *Tech. Rept. 87-351*, University of California, Computer Science Division.

Woods, WA, 1975. "What's in a link: foundations for semantic networks" In *Representation and Understanding: Studies in Cognitive Science*, Academic Press, New York, pp. 35–82.

Young, SR, Hauptmann, AG, Ward, WH, Smith, ET and Werner, P. 1989. "High-level knowledge sources in usable speech recognition systems" *Communications of the ACM* **32**(2) 183–194.