Statement of Interest

Lewis M. Norton
Paoli Research Center
Unisys Corporation
PO Box 517
Paoli, PA 19301
norton@prc.unisys.com
1989

For the past six years, our group at the Paoli Research Center of the Unisys Corporation has been developing a configurator for Unisys computer systems. At the heart of this configurator is the KNET knowledge representation system, conceived and originally implemented by the late Michael W. Freeman. KNET is a KL-ONE-inspired semantic network representation formalism, implemented using logic programming. In the course of work on this project, we have accumulated extensive practical experience with the use of KNET as the knowledge base for expert systems. In particular, we have developed a paradigm for the creation of such a system, based on the single idea that one can add procedural control information to a KNET knowledge base by means of some algorithm that traverses the semantic network and "fires" small modules of code, which we call "constraints", associated with particular nodes of the network.

Using KNET augmented with control information according to our paradigm has provided us with a solution extremely well-suited to the needs of the configurator project, in terms of representational adequacy, computational effectiveness, and maintainability. In this note, I will provide an overview of our system, and elaborate on our experiences with it, for the many insights we have gleaned should be of interest to this workshop audience. I will also mention a few of the issues which we have not had time to stop and
properly consider due to the pressures of developing an "industrial-strength" expert-system application with limited resources. It is my hope that these insights and issues will stimulate discussions which will be mutually beneficial to designers, implementors, and users of related knowledge representation systems.

Space does not permit a detailed description of KNET, but some background information is needed to make the following discussion comprehensible. KNET has not changed significantly since the early 1980's, and therefore may appear primitive to those familiar with the current variations of semantic network formalisms. It has the usual specialization (class-subclass) hierarchy of concepts and individuals, it permits inheritance in the usual way, and it allows no exceptions to inheritance strict subsumption is enforced. Also, KNET has no classifier; its applications have no need for an automatic decision procedure for placing independently-created concepts into an existing network. Orthogonal to the specialization hierarchy is the aggregation (part-whole) hierarchy, used to represent both the decomposition into subcomponents of a complex concept, and any attributes which the concept may have. This is done by attaching roles to concepts and individuals. Roles have certain "facets", the most important of which are the role's type, range, and name. The "type" is a concept, often referred to in the literature as the value restriction of the role. The "range" is a pair of numbers, used to record the minimum and maximum number of fillers that an instance of the concept can have for the role in question.

In addition to the two hierarchies just described, KNET provides for objects called "constraints", which are (usually brief) modules of executable code. Constraints also have "facets": an owner (some concept which scopes, in the aggregation hierarchy, all roles referred to by the constraint); a set of roles to which the constraint refers; a distinguished role at which the constraint is to be triggered; an application identifier (thus allowing constraints for more than one application to be co-resident in the same knowledge base); the constraint body, i.e., the actual code of the constraint; and the constraint's name. Constraint bodies can call upon a full library of subroutines for knowledge base access and modification.

In KNET, roles and constraints are ordered. A total ordering is imposed upon the roles owned (or inherited) by a given concept or individual, and a total ordering is imposed upon the constraints with a given application identifier triggered at a given role. Using these orderings, one provides a
well-defined traversal algorithm (different algorithms may be associated with different applications) which, as it reaches each role of the network in the course of its traversal, executes all the constraints (with the appropriate application identifier) triggered at that role.

From this very general description of how we create an expert system using a KNET knowledge base, we turn to some specific examples. First, we consider a utility used to augment a knowledge base for the configuration application. The basic idea is that we wish to separate relatively stable from relatively volatile information, in order to simplify the task of maintenance. So we ask a modeller to create, using a KNET browser-editor, a KNET network modelling a certain computer system, using general concepts such as memory-module, cpu, disk-subsystem, disk-drive, etc. (For instance, disk-drive will be the type of a role of disk-subsystem.) The intention is to model only fairly stable information about the computer system. Much of this will be captured in the network structure, but some will be encoded in constraints, such as a requirement that the system needs either a disk drive or a tape drive. More volatile information, such as which particular disk drives Unisys sells today for this computer system, is omitted from the model, and stored in a database in relational form. At configuration time, however, that information needs to be in the KNET model. The solution is to have a utility we call the "database loader", which is a very specialized example of our paradigm. For each role whose type needs to be specialized in order to represent database information, the modeller enters a constraint to be triggered at that role, which will call a "load" subroutine, and give this constraint the identifier 'database'. Then a simple depth-first left-to-right traversal algorithm firing these constraints results in the creation of new KNET structure corresponding to database information. For instance, suppose that a role has disk-drive for its type, and that the concept disk-drive has roles for transfer rate and capacity. Further suppose that today’s available disk drives are represented in relational table form with rows such as:

['B9494-32', disk-drive, 3.0, 576]

meaning that B9494-32 is the catalog identifier for a disk drive capable of transfer at 3 MB/sec. and capacity 576 MB. The database loader will create a specialization of disk-drive, named 'B9494-32', with the types of its transfer
rate and capacity roles restricted to 3 and 576 respectively. Note that 'B9494-32' is NOT an individual, but a concept. We have merely succeeded in separating stable from volatile knowledge about generic concepts. In short, the database loader can be considered to be a simple expert system which, driven by data from relational tables, simulates a user of the KNET browser-editor.

We call the augmented KNET knowledge base resulting from a run of the database loader a "loaded" knowledge base. The configurator application works only on loaded knowledge bases, and is of course a much more complex program than the database loader. We can give only a rough outline of it here. Basically, using a slightly different, "controlled" depth-first left-to-right traversal that fires constraints with two different identifiers at appropriate times, the configurator incrementally creates a new network of individuals corresponding to a particular instance of the modelled computer system. Sometimes the structure of the generic model fully determines what must be created in the individuation network, sometimes information hard-coded in constraint bodies makes the determination, and sometimes the subroutines called by the constraints have to interact with the configurator's user. The configuration process makes heavy use of two further KNET constructs, decomposition and differentiation. Decomposition is a relation grouping one or more of a concept's descendants (usually immediate children, but not always). For example, the specializations of disk-drive which the database loader creates are taken to be a decomposition of the concept disk-drive. In effect, if the type of a role has a decomposition, the members of the decomposition become candidates for the filler of the role (more precisely, newly created individuations of these members will be the fillers). Differentiation is more familiar if a role has multiple fillers, the appropriate number of differentiators of the role are dynamically created in the individuation network.

For instance, if a computer system is allowed to have 1 to 10 disk-subsystems, and the user chooses to have 2, two differentiators of the role whose type is disk-subsystem will be created. And if the disk-drive for one of these disk-subsystems is to be a B9494-32, then within the aggregation hierarchy of the differentiator corresponding to that disk-subsystem, a newly-created individual individuating the concept B9494-32 will be installed as the filler of the role (inherited from disk-subsystem) whose type is disk-drive.

To complete this brief description, the configurator's task is to create an individuation network where all role ranges are converged, all types are
"converged" (roughly equivalent to making specific choices from all decompositions), all configuration restrictions encoded in constraints are observed, and subsumption with respect to the generic model has been strictly enforced. The configuration is thereby guaranteed to be correct "by construction", assuming of course that the generic model of the computer system is correct. The actual list of parts corresponding to the configuration can be obtained directly from the completed individuation network.

We have made other applications following this general paradigm, and have found it a very convenient way to construct expert systems using a semantic network as the knowledge base. In fact, we have found that our systems are free of the problems that have been reported in the literature for expert systems based on large, unorganized sets of production rules. In all candor, however, practice has outrun theoretical analysis. The structural portion of a model (i.e., a KNET knowledge base without its constraints) is relatively well understood. For one thing, since we are not dealing with a classification algorithm, subsumption becomes no more than a pair-wise check between concepts or individuals, and is completely tractable. But our use of constraints has been undisciplined. In practice, the context afforded by knowing a scoping concept for a constraint, and a cross-index of constraints and the roles involved in them, has made the collection of constraints perfectly manageable. However, it would be better to have some further knowledge about the constraints. For example, some constraints represent knowledge about the system being modelled, independent of application, while others (often most, in our paradigm) are definitely artifacts of a particular application, and embody control information for the application rather than interrelations between the nodes of the knowledge base. Also, maintenance would be even easier if we had some way of knowing which of the roles referred to by a constraint were possible targets of modification, and which were "read-only".

Another interesting issue involves the distinction between concepts and individuals. In our use of KNET, there are NO individuals in our knowledge bases per se. Only the configurator creates individuals. However, not every object that the configurator creates is properly an individual. For example, suppose that disk-subsystem has roles for drives and a controller. If we pick a drive with a particular transfer rate, it makes sense to restrict the decomposition of disk-controller to just those that can operate at least at that transfer rate. In effect, we dynamically create, for at least temporary use, an
object representing controllers that can operate at such rates. This clearly is not an individual, but a new concept which specializes disk-controller. But which facet of an individual's role should have such a new concept as its value? KNET, while it allows restricted ranges on individual roles, does not allow restricted types, only fillers. This in hindsight is an error, and probably one that would be easily corrected if we had the time to do something as fundamental as change the primitives of our semantic network representation. Incidentally, note that the absence of a classifier can cause harmless anomalies in this respect depending on the details of a model, the configurator may create a second concept specializing disk-controller for controllers supporting an even higher transfer rate, thus failing to capture the fact that this new concept really represents a subset of the first specialization of disk-controller.

Unfortunately, for reasons unrelated to the technical content of our system, Unisys will not be deploying this configurator, and therefore further work on the use of KNET as a knowledge base for expert systems will have to be done in the context of as yet unspecified projects. But we will be able to build upon the experience of developing a score of models of complex computer systems, each with on the order of 10000 assertions needed to represent the structure of the model, and 300-400 constraints. KNET has proved to be admirably suited for this application, and we hope to apply this technology to other applications in the near future.

Bibliography:


pp. 400-403.