Production System Extensions to KSL

By Fred A. Cummins

Object-Oriented and Artificial Intelligence Services
Electronic Data Systems
5555 New King Street
Troy, Michigan 48098

I. Introduction

KSL is an object-oriented language developed by EDS Research and Development in the mid-1980's [1]. It was developed to support the integration of artificial intelligence techniques. Consequently, extensions were developed for rule-based inferencing as well as logic programming [2].

The production system extension provides forward-chaining inferencing similar to OPS5, but with an open environment. The domain is represented by an object-oriented model and changes to the model can occur independent of inferencing. In fact, multiple production systems could be active for a single model. Changes to the model drive updates to rule satisfaction through the KSL active value facilities.

The following sections will discuss the KSL production system extension in greater detail. Section II will provide an overview of KSL. Section III will describe the production system facilities. Finally, Section IV will discuss some conclusions and observations.

II. An Overview of KSL

KSL is a class-based, pure object-oriented programming language. It is implemented on Sun workstations in the C language. It is the language of an interactive programming environment with automatic memory management and a graphical user interface.

KSL objects are fully encapsulated. Instance variables can only be accessed through message sends, even from local methods. Objects can be specialized by attaching behavior to specific instances[3]. Instance behavior definitions can intercept messages that would otherwise be resolved by the class, or they may provide additional behavior not defined by the class. This instance behavior mechanism is used to implement active values. To implement an active value, an instance variable message is intercepted by an instance behavior which can then invoke the class behavior and produce instance-level side effects based on the result. This facility was key to implementation of the production system.

KSL is also fully reflective. Classes are instances of the class "Class". Classes contain objects that specify instance variables and methods. The instructions in methods are also object structures which are interpreted through message sending that invokes evaluation methods. This interpreted mode supports the KSL interactive programming environment and the ability to dynamically change executable instructions either manually or programmatically. To improve performance, the KSL methods can be translated (again through message-sending) to produce C language code which is compiled and linked into the environment. Thus, in general, some methods will be interpreted and others will be compiled.

The reflective capability of KSL was also key to the implementation of the production system. Additional behaviors were defined on the language objects and new language objects were defined. The approach is described in some detail in the next section.

III. The Production System Extension

In KSL, a production rule set is a specialization of method. When an appropriate message is sent to an object, the class of that object will determine the appropriate behavior to be invoked. The behavior object, in this case the rule set, is "executed" by invoking its evaluation behavior with a context which contains the target object and the arguments of the original message. The rule-set evaluation behavior applies the rules to the situation defined by the context. The evaluation behavior of the rule set object can be viewed as the inferencing engine although the inferencing mechanism is comprised of many methods on a number of objects.

A special class of variables, called "search space" variables was defined for inferencing in KSL. A search space variable is associated with a collection. The search space variable is bound to members of the collection. Collections are assigned to search space variables by initialization instructions associated with a particular rule set based on the target object and the arguments passed in the invoking message.

The inferencing process begins by instantiating the rules for the particular context. Instantiation takes the form of building a conflict set and discrimination network to link the rule conditions to the object model. Each rule is translated to a network structure through recursive message sends to the language elements of the conditional expression. Equivalent expressions in different rules share branches of the discrimination network to eliminate redundant processing. The leaf nodes of the network are attached to the object model as recipients of "alert" messages from active values. Whenever a relevant change is made to the object model, the affected network nodes are activated by alert messages. The number of rule instantiations required is determined by the possible values of the referenced search space variables.

Once the rules are instantiated, they can be evaluated to determine the conflict set. At this point, inferencing proceeds in a manner equivalent to the well-known OPS5 language. A rule is selected from the conflict set and executed, possibly updating the object model. The conflict set is updated and another rule is selected. This continues until the conflict set is empty.

When the conflict set is empty, the KSL inferencing process is not necessarily complete. Control will be returned to the original message sender, but the rule instantiations remain connected to the object model. If and when some other action causes a relevant change to the object model, the discrimination net will be alerted. A continuation of the inferencing process will be scheduled and later executed. A new conflict set will be built based on the current situation represented by the object model and the inferencing process will resume.

In addition to the implementation of search space variables, the production system extension required two other additions to the basic KSL executable instruction set: the rule instruction and the pattern instruction.

A rule expression contains a condition expression comprised of basic KSL conditional expressions, including references to search space variables, along with a collection of action expressions comprised of basic KSL message expressions. The rule class includes methods for rule instantiation, firing, etc.

A pattern expression defines a context in which all references to the same search space variable will be bound to the same value. This improves the readability of the rule expressions and simplifies the network processing. A rule condition may contain multiple pattern expressions where each may have multiple instantiations. The number of rule instantiations is the product of the pattern instantiations.

Additional facilities were developed to assist the application developer in the creation and debugging of the rule sets. These provided interactive, graphical representations of the rules and trace facilities for analysis of the inferencing process.

IV. Conclusions and Observations

The KSL production system facilities were applied in a system to automate the operation of a large, on-line, commercial data processing application. The input to the system was the status and error messages normally observed by a system operator. The elements of the commercial application environment were represented with an object-oriented model. As operator messages were received, the model was updated. The expertise of the system operator was encoded in production rules. The production system monitored the object oriented model and issued commands for corrective action when appropriate.

The KSL production system extension provides a very flexible inferencing facility. The ability to invoke multiple, independent inferencing processes on the same model allows a very complex problem to be addressed while keeping the individual inferencing processes relatively simple and self contained. The linkage to an object model in an open environment, allows the inferencing process to occur in the background for applications where an intelligent monitor is desired for an interactive modeling application. The use of the discrimination network to link changes in the model to the inferencing process reduces the processing required to apply large rule sets.

The inferencing mechanism, while based on a discrimination network, still has considerable overhead. Since the system was implemented several years ago, this might be less a concern with more powerful workstations. To address this concern, a mechanism was implemented to "compile the rules." This process translated much of the evaluation logic of the discrimination network to C language code for compilation. This yielded an inferencing process that was about three times as fast.

The basic approach appears to be sound and offer significant advantages. Additional work is needed to improve performance.

Acknowledgments

The KSL language and environment was developed by William E. Bejcek, Scott W. Woyak, Mamdouh H. Ibrahim, Daniel J. O'Leary, Allan Cherri and the author. Much of the work of developing the production system extension was done by Scott Woyak and Allan Cherri, and much of the work of developing the rule compilation process was done by Scott W. Lagona. Development tools were developed by Scott Woyak and Gary S. Schiltz.

[1] Ibrahim, Mamdouh H. and Cummins, Fred A., "KSL: A reflective, Object-Oriented Programming Language, "Proceedings of the International Conference on Computer Languages, 1988.
[2] Ibrahim, Mamdouh H. and Cummins, Fred A., "Objects with Logic," Proceedings of the ACM Computer Sciences Conference, 1990.
[3] Ibrahim, Mamdouh H., Bejcek, William E. and Cummins, Fred A., "Instance Specialization without Delegation," Journal of Object-Oriented Programming, June, 1991.