Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

SIMULATION WITH RULES AND OBJECTS

Michael E. McFall and Philip Klahr
Inference Corporation
5300 W. Century Blvd.
Los Angeles, Ca. 90045

ABSTRACT

We will describe a transformation of an object-oriented knowledge-based simulation language implemented with procedural behaviors: Rand Corporation's ROSS language, into ART-ROSS, an integrated rule and object based implementation and extension of the ROSS language. The implemention of ART-ROSS in a state-of-the-art integrated language (Inference Corporation's ART™) provides benefits of increased flexibility, simplified architecture, full explanation capability, and access to the vectorized message passing capability known as multi-methods.

1. ROSS into ART-ROSS

Rand Corporation's ROSS language [McArthur, 1986] is an object-oriented simulation language used primarily in the area of military war-game simulation. This language was one of the first to provide inheritance from multiple classes of objects. This feature of multiple inheritance has been well proven in other areas of knowledge based programming. ROSS also allows arbitrarily complex English expressions to exist as messages and incorporates a sophisticated pattern matcher to trigger behaviors when these messages are sent.

Within the ROSS language there exist several types of objects, including the clock, generic class objects, and individual objects representing instances of the generic objects. The clock has three attributes associated with it: the current simulation time, the size of the simulation time step or tick-size, a list of the current objects running and the time corresponding to the next action for each object. The individual objects have one default attribute, a list of things to do, each of which contains the time at which an action is to take place in the simulation and a pattern describing what the action is.

In ROSS-based simulations, such as SWIRL [Klahr,1982] and TWIRL [Klahr,1986], specialized objects, the Physicist, the Mathemetician, and the Scheduler, have been added. These objects answer questions such as 'Will I collide with another object?' and 'How far am I from object 123b?'. These objects are artifacts of the procedural nature of the ROSS language. Some object somewhere must have the knowledge on how to query the state of the entire simulation. Any pure object-oriented simulation must make the choice between having some procedural arbiter having knowledge of how to calculate relationships and query the appropriate individual objects of the simulation, or have each object in the simulation have the capability of querying every other object in the system.

Moving from ROSS into ART is a simple process. The multiple-class inheritance feature of ROSS is also a feature of ART's schema language. ART's inheritance mechanism is even more general, allowing the user to define custom inheritance relationships. These two features allow the knowledge base semantics to reflect as closely as possible the actual real-world relationships and classifications of the simulation objects. For example all radar sites have a special relationship called SUPERIOR, whose inverse is SUBORDINATE. In our example [Fig. 1] there are two types of radars, surface installations and AWACS, with AWACS also inheriting the attributes of airborne-objects. With the flexible semantics of user-defined relationships, by identifying surface-installations as having a SUPERIOR of a particular instance-of filter-center, the filter-center automatically has the surface-installation instance listed as a SUBORDINATE.

Since message passing is triggered by pattern matching in ROSS, ART's powerful pattern matching and rule language provide a natural mechanism for implementing ROSS message passing. An initial simplified prototype of SWIRL, a military air battle simulation originally implemented in ROSS at Rand [Klahr,1986], was implemented in ART in less than two days. ART also allows the retention of ROSS's ability to have arbitrarily complex English expression of behaviors.

The conversion from the essentially procedural implementaion of ROSS to the rule-based implementation in ART has a number of obvious benefits. The first are modularity and ease of expression. Rules are a very natural way to express the behavior of objects, particularly in relation to, and in combination with, other objects. A complex series of queries in the form of messages between objects can be easily modeled by a series of patterns matching against attributes of any number of objects on the left hand side of a simple IF.. THEN rule. Rule based programming also provides benefits of easier knowledge acquisition, explanation of actions and ease of development are other benefits.

A major benefit of rule-based simulation is in the simplification of the overall architecture. ROSS required a class of objects which knew of, and could query, other objects in the data base for finding the overall state of the simulation, or for establishing relationships between objects in the data base. These objects were triggered by requests from other objects in the simulation. In the ART implementation this state information is stored at all times in the pattern and join network, with the behavior rules having immediate access to the necessary information. The following is a rule that will change the heading of any airborne object at the appropriate time without requiring an intermediate Scheduler object. Variables are identified in ART by a preceding '?' and constraints on variables are preceded by '$:' which is read "and such that...". This rule uses ART's default natural language syntax for each slot or attribute. This syntax can be further customized by the user.

```
(Defrule Change-Heading
  "for any airborne object, penetrator, fighter or awacs"
  IF
     (The sim-time of the-clock is ?current-t) AND
     (The tick-size of the-clock is ?delta-t) AND
     (The instance-of of ?object is airborne-object) AND
     ?n <- (The next-action of ?object is
            (?time$:(<= ?time (?current-t + ?delta-t))
            CHANGE HEADING ?direction DEGREES))
            AND
 ;; the next action pattern includes a test and a binding for
 ;; later retraction on the right hand side of the rule.
     (The position of ?object is (?x ?y ?old-time)) AND
     (The velocity of ?object is (?vx ?vy))

  THEN
     (BIND (?new-x ?new-y ?new-vx ?new-vy)
        (revector ?x ?y ?vx ?vy    ;; a LISP function
           (- ?current-t ?old-time) ?direction))
     (retract ?n)  ;; retract the value of the next-action slot
     (MODIFY
        (The position of ?object is
                              (?new-x ?new-y ?current-t))
        (The velocity of ?object is (?new-vx ?new-vy))))
```

ART is a data-driven language: when the conditions of a rule are matched, the rule is immediately activated for firing. In the previous rule, the time for the next scheduled action is tested against the current time plus the simulation time step size. This is accomplished by a very efficient implementation of the Rete algorithm [Forgy,1982]. Only those rules which can use a pattern of a particular type are notified of the pattern's existence in the data-base. As these patterns are asserted into the data-base, the system compares these patterns based on the restrictions and constraints of the rules of the system. Partial matches of rules are maintained in the join memory. These join memories maintain the current state of all partially and fully matched rules. When an object in the data base changes state, the appropriate patterns and joins are notified and any fully matched rules are activated for firing.

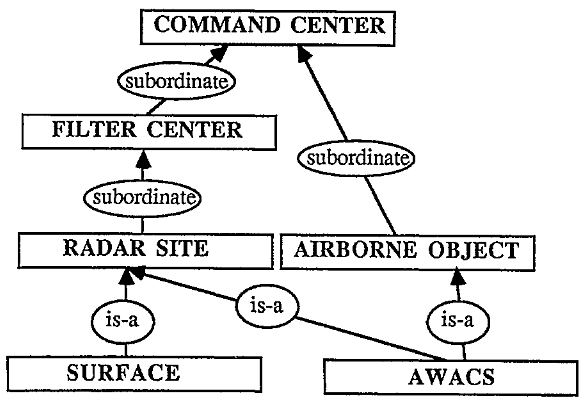

Figure1: Inheritance and Relational Network

2. Performance

While this method of data-driven programming is the most efficient way to implement a rule language, there are potential performance penalties. Each individual join memory is the cross product of all combinations of patterns that enter into the join. If a particular join has 1000 partial matches, 1000 comparisons occur at that join for each new fact that affects that join. When this single fact affects more than one join (which can and does occur if the rules are not written appropriately), peformance degradations do occur.

The obvious solution is to try to minimize the size of the join memories. In ART 2.0 this was done by careful analysis of the left-hand side of the rules and the order of the patterns in the individual rules. With ART 3.0 the user now has explicit access to the join topology. One can now isolate particularly active join memories and optimize them independently of the remaining pattern and join network. This is particularly easy to do in the area of simulation. In the case of our implementation of SWIRL, a large class of rules was concerned with the spatial locations of all objects in the database. By identifying these patterns as being common among many rules and explicitly isolating these patterns in their own join memory, the effect of a single object's change in location and the affect on system performance is minimized.

Section 2 discusses the benefits of having a single rule being able to query the attributes of multiple objects, without having to send multiple messages to each object. This benefit is extended to the right-hand or action component of rules in ART 3.0 with multi-methods. Multi-methods provide the capability of associating a procedure with a typed vector of objects. In the context of SWIRL, different classes of missiles may behave differently when sent against different targets due to varying types of electronic counter measures. A single procedure defining the appropriate behavior can be written for each combination of missile and target. When a missile is sent against a target, the appropriate procedure is called based on the missile and target classes. The benefits are further modularization of the procedural code, and a simpler message-sending syntax that does not require complex case statements. Procedural code not relevant to the problem at hand is further simplified. ART 3.0 integrates procedural attachments for active values, methods and multi-methods into its rule and object knowledge-base, which allow the user to avoid the chauvinism often associated with a strictly rule-based programming approach. This flexibility enables the programmer to select the most appropriate form of knowledge representation for a given situation.

A major difference between ROSS and other simulation languages is the distribution of the tasks at hand. Each object in ROSS has its own list of tasks and the associated execution times. This feature is also easily implemented in ART. Other simulation systems have queue objects and associated dequeueing functions based on various distribution schemes. Multi-methods can be used to great advantage in these situations also. Queueing and dequeueing functions can be associated with the appropriate objects with multi-method typed vectors.

3. ART SWIRL

As has been shown, the ART implementation of SWIRL uses objects and pattern matching much as the original ROSS implementation. Two major differences simplify the ART implementation. The ROSS implementation of the clock included a list of the next actions for each active object in the simulation. The clock would send messages to the successive objects telling them when to act. The ART implementation stores these lists of actions only in the objects themselves. ART's data-driven architecture allows the rules to recognize automatically when the next action for a given object takes place since all rules match against the simulation time.

ROSS also has the global objects used for calculating various multi-object states and relationships. Since the entire state of the simulation is maintained in the pattern-join net, the rules defining behaviors determine when, for example, a given aircraft is within the air-space of another object. No intermediate object is required. The ART implementation enables the appropriate behavior to be defined at the most natural location in the code, in this case the rule defining the desired behavior.

4. Future Work

Inference Corporation has been involved with the development of several simulation systems including an emulator for custom digital hardware microcode development. We are currently developing and exploring extensive applications using integrated rule-method architectures. We anticipate developing principles of appropriate knowledge representation schemes for general simulation systems, and to weigh the benefits of rule-object-multi-method approaches to simulation.

REFERENCES

Forgy, C.L. "Rete: A fast Algorithm for the Many Pattern / Many Object Pattern Match Problem." Artificial Intelligence, 14, 1982, 17-37.

Klahr, P. et al. "SWIRL: An Object-Oriented Air Battle Simulator." Proceedings of the Second National Conference on Artificial Intelligence, Pittsburgh, 1982.

Klahr, P. et al. "TWIRL: Tactical Warfare in the ROSS Language." In Expert Systems (P. Klahr and D. Waterman, Eds.), Addison-Wesley, 1986

McArthur, D.J., Klahr P., and Narain, S. "ROSS: An Object-Oriented Language for Constructing Simulations." In Expert Systems (P. Klahr and D. Waterman, Eds.), Addison-Wesley, 1986.

ART is a trademark of Inference Corporation

AUTHORS' BIOGRAPHIES

MICHAEL E. MCFALL is a Knowledge Engineer in Inference Corporations Los Angeles office. He received his B.S. in Mathematics at the University of Washington in 1985 and developed several frame-based expert systems while at Boeing Aerospace Corporation in Seattle, Washington. His current research interests include knowledge based simulation architectures and knowledge based human interface design.

Michael E. McFall
Inference Corporation
5300 West Century Blvd.
Los Angeles, Ca. 90045
(213) 417-7997

PHILIP KLAHR has been involved in artificial intelligence research and applications for over fifteen years. He has a B.S. in Mathematics from the University of Michigan and an M.S. and Ph. D. in Computer Sciences from the University of Wisconsin. He is currently Director of Applications at Inference Corporation where he is responsible for managing the design and development of expert systems for government and industry. Prior to joining Inference, Dr. Klahr directed The Rand Corporation's research program in artificial intelligence and advanced computer science. At Rand, he pioneered the application of artificial intelligence technology to military simulation. He is best known for his development of the ROSS object-oriented simulation language and two combat simulations written in ROSS, called SWIRL and TWIRL. He also supervised the development of the ROSIE expert system language and the Time-Warp mechanism for distributed simulation using parallel processing. Dr. Klahr has published over twenty-five articles in artificial intelligence and is co-author of Expert Systems: Techniques, Tools and Applications, to be published by Addison-Wesley.

Philip Klahr
Inference Corporation
5300 West Century Blvd.
Los Angeles, Ca. 90045
(213) 417-7997