A Rule-Based Process Server Component for Constructing Rule-Based Development Environments

Gail E. Kaiser
Columbia University
Department of Computer Science
New York, NY 10027

We have been working on the MARVEL rule-based development environment (RBDE) for several years [9, 7], and are currently in the process of scaling up to supporting cooperation among multiple users working together on large projects [1]. This line of research led us to breaking up the previous single user MARVEL 2.6 implementation into the two main pieces of multiple user MARVEL 3.0: the clients representing user interface and activity management, and the server consisting of modules that support process model enaction, concurrency control, and object management [4, 2]. We are now entertaining the definition of a set of components, generalized from MARVEL's modules, which could be used to construct a range of RBDEs. Here we discuss our initial ideas regarding the most fundamental component, the *rule-based process server*.

The proposed Rule-Based Process Server component (RBPS) would would provide a toolkit for defining rule languages for specifying the process, and corresponding chaining engines supporting a spectrum of assistance models that execute or interpret the process in some manner. RBPS would generalize our previous work on rule-based process modeling as part of the MARVEL environment kernel.

In MARVEL, each activity corresponds to a user command and is encapsulated by a condition that must be satisfied before initiating the activity and an effect that asserts the result of completing the activity on the state of the process [8]. The activity is treated as a ‘‘black box’’, and is typically an external tool invoked through an extended shell script envelope [6]. Thus a MARVEL rule may have multiple mutually exclusive effects, representing the multiple possible results of such tools — usually success and failure cases.

MARVEL is only one instance of what we call a RBDE. We have defined a spectrum of RBDEs in terms of the *assistance model* they support, ranging from pure automation to pure consistency preservation. The original single-user MARVEL supported a pure automation assistance model. Backward chaining was employed to attempt to make the condition of a user-selected activity satisfied, while forward chaining was used to trigger activities whose condition was made satisfied by the effect asserted for the original activity. Darwin [10] supports a pure consistency preservation model, in that backward chaining is employed only to determine the truth or falsity of a proposed program change, which is modeled as a Prolog-style fact regarding whether the change is permissible. AP5 [5] supports separate automation and consistency rules, with consistency maintained via forward chaining to recompute derived relationships among objects. Only forward chaining is employed for automation rules as well.

The new multi-user MARVEL 3.0 supports a sophisticated ‘‘maximalist’’ assistance model that combines automation with consistency maintenance [3]. Predicates in conditions and effects may be designated either automation or consistency. Forward chaining over consistency predicates is employed to fulfill the implications of a consistency predicate in the effect of the activity that matches a predicate in the condition of another rule. Backward chaining is thus never needed, since by definition all consistency implications have already been asserted. The activity of a rule triggered during forward consistency chaining must be reversible, or null, since either an entire consistency chain must execute or none of it, in the sense of classical transactions. In the current implementation we support only consistency chaining among rules with null activities, except for the original rule that triggers the chain, and a consistency predicate in an effect that triggers a rule with a non-null activity is treated as an automation predicate. Automation chaining (i.e., chaining due to automation predicates) operates essentially the same as in the single-user MARVEL 2.6, and may trigger irreversible activities, since by definition an automation chain may be terminated after any rule without affecting the validity of the original user-selected activity or other previous activities in the chain.

```
# deposit: deposit an object.  This rule works on the same objects as the
#          reserve rule.

deposit[?old:CFILE]:
    # Only deposit this file if the object is a child of a MODULE
    # type object through the "cfiles" attribute
    (exists MODULE ?m suchthat (member [?m.cfiles ?old])):
    #    No chaining allowed on these predicates
    #
    # 0. Only deposit if user who locked object is current user
    # 1. Only deposit if object was in fact CheckedOut.
    # ---------------------------------------------------
    (and no_chain (?old.locker = CurrentUser)
         no_chain (?old.reservation_status = CheckedOut))

    { RCS deposit ?old.contents ?old.version ?old.history }

    [?old.reservation_status = Available];

# This rule will "touch" a module if any of its CFILEs has become
# "NotCompiled" or compiled in "Error".
# also, if the reservation status became available
# as a result of deposit touch the module.

hide touchup[?M:MODULE]:
   (and
    (forall CFILE  ?c suchthat (member [?M.cfiles ?c]))
    (forall YFILE  ?y suchthat (member [?M.yfiles ?y]))
    (forall LFILE  ?x suchthat (member [?M.lfiles ?x]))
    (forall MODULE ?m suchthat (member [?M.modules ?m])))
    :
    (or [?c.status     = ErrorCompile]         # from compile CFILE
        [?c.reservation_status = Available]    # from deposit FILE
        [?y.status     = ErrorCompile]         # from compile YFILE
        [?x.status     = ErrorCompile]         # from compile LFILE
        [?m.archive_status     = NotArchived])
    { }
    [?M.archive_status = NotArchived];
```

Figure 1: C/Marvel Deposit and TouchUp Rules

Figure 1 shows two actual rules from the C/Marvel programming environment for C, which we are using in our implementation of MARVEL 3.1. The rules correspond to the user `deposit` command and the hidden `touchup` rule (i.e., the `hide` directive makes it unavailable from the user menu, since it is used only during chaining). The condition of the `deposit` rule states that the C file must be checked out by the current user (i.e., the user executing the `deposit` command). The "?old.reservation_status = Available" predicate in the effect is marked as consistency, by its enclosure between square brackets "[...]"; predicates enclosed between parentheses "(...)" are automation predicates, allowing backward compatibility with old rule sets. This guarantees that the deposit will take effect only if the implications of this predicate can be carried out, in particular, to execute the `touchup` rule on the enclosing module. Note that there is an error in the comment for the `touchup` rule: this rule is not actually triggered when a C file's status becomes `NotCompiled`. The `no_chain` directives in the `deposit` rule disallow forward chaining to automatically trigger `deposit` whenever a C file is reserved, and disallow backward chaining to automatically trigger `reserve` when a user attempted to deposit a C file that is not checked out to him or her. Both of these were considered undesirable behavior for practical programming.

There are three main goals for our proposed work on RBPS. The first is to abstract away from the MARVEL Strategy Language, in which the rule-based process model for MARVEL environments are written. This is simple from a syntactic perspective; we have already gone through many iterations on the yacc grammar used by the MARVEL parser. The semantics will be reasonably straightforward for languages that map into our spectrum from consistency preservation to automation. Our idea is for an RBPS library to include an extensible rule-based process modeling ‘‘assembly language’’ into which the new classes of rules could be translated.

The second major goal is strongly tied to the first: we would generalize MARVEL's Opportunist module, or chaining engine, into a parameterizable Process Server. This would be the ''abstract machine'' for the ''assembly language'' above. Due to the need for flexibility in parameterization, this ''abstract machine'' would necessarily be interpretive in nature and thus relatively inefficient. However, the RBPS library might contain a small number of optimized rule network representations and corresponding hard-coded chaining engines for selected points on the assistance model spectrum, including pure automation, pure consistency maintenance, and the maximalist form of integration we have implemented for MARVEL 3.0.

The final and perhaps most challenging goal is to devise interfaces between RBPS and the other components of environments, and codify a corresponding methodology for employing RBPS. Consider the possibility of inserting RBPS into an existing environment. This would involve breaking certain communication links within that environment, and inserting the Process Server in the middle. For example, if the original behavior of the environment was centered on a command dispatch loop, then commands that could be mapped to rules would be rerouted through the Process Server and the results of those commands would be returned through or otherwise communicated to the Process Server; this is how MARVEL works. An alternative possibility is that the individual commands do not map to individual rules, but instead it is desirable for lower level tool operations on the objectbase to map to rules; then the Process Server might be projected onto a method call facility. The subroutines in the RBPS library would have to be sufficiently flexible to handle a variety of schemes for executing its ''abstract machine''.

This work is still at the drawing board stage, but we expect to firm up our ideas over the next year, and then begin extracting initial versions of our components from the MARVEL implementation.

Non-profit institutions and industrial sponsors may contact the author for information about licensing MARVEL 3.0. The Programming Systems Laboratory is supported by National Science Foundation grants CCR-9106368, CCR-9000930 and CCR-8858029, by grants from AT&T, BNR, DEC and SRA, by the New York State Center for Advanced Technology in Computers and Information Systems and by the NSF Engineering Research Center for Telecommunications Research.

References:

[1] Naser S. Barghouti and Gail E. Kaiser. Modeling Concurrency in Rule-Based Development Environments. *IEEE Expert* 5(6):15-27, December, 1990.

[2] Naser S. Barghouti. *Concurrency Control in Rule-Based Software Development Environments*. PhD thesis, Columbia University, November, 1991.

[3] Naser S. Barghouti and Gail E. Kaiser. Scaling Up Rule-Based Development Environments. In A. van Lamsweerde and A. Fugetta (editor), *3rd European Software Engineering Conference*, pages 380-395. Springer-Verlag, Milano, Italy, October, 1991.

[4] Israel Z. Ben-Shaul. An Object Management System for Multi-User Programming Environments. Master's thesis, Columbia University, April, 1991.

[5] Donald Cohen. Automatic Compilation of Logical Specifications into Efficient Programs. In *5th National Conference on Artificial Intelligence*, pages 20-25. AAAI, Philadelphia, PA, August, 1986.

[6] Mark A. Gisi and Gail E. Kaiser. Extending A Tool Integration Language. In *1st International Conference on the Software Process*, pages 218-227. Redondo Beach CA, October, 1991.

[7] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti and Israel Z. Ben-Shaul. Rule Chaining in MARVEL: Dynamic Binding of Parameters. In *6th Knowledge-Based Software Engineering Conference*, pages 276-287. Rome Laboratory, Syracuse NY, September, 1991.

[8] Gail E. Kaiser, Peter H. Feiler and Steven S. Popovich. Intelligent Assistance for Software Development and Maintenance. *IEEE Software* 5(3):40-49, May, 1988.

[9] Gail E. Kaiser, Naser S. Barghouti and Michael H. Sokolsky. Experience with Process Modeling in the Marvel Software Development Environment Kernel. In Bruce Shriver (editor), *23rd Annual Hawaii International Conference on System Sciences*, pages 131-140. Kona HI, January, 1990.

[10] Naftaly H. Minsky and David Rozenshtein. A Software Development Environment for Law-Governed Systems. In Peter Henderson (editor), *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 65-75. ACM Press, Boston MA, November, 1988. Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.