Workshop Report:
Embedded Object-Oriented Production Systems (EOOPS)

Abstract

The goal of the workshop was to lay the foundations for the development of an integrated model of Object-Oriented and Production System programming. Isolated, recurring efforts to integrate these paradigms have failed to produce an acceptable unified model. Other workshops have emphasized the need to bridge the gap between OOP and AI. Because of diverse definitions of both "object" and "rules" those workshops failed to make significant progress. This one focused precisely on the integration of first-order, forward-chaining rules, with Object-Oriented Programming Languages. This workshop was the first one to bring together the most active contributors to the area of EOOPS. It was indeed very successful, mainly because the subject-matter proved to be technically well-defined, with a number of hitherto undiscussed issues, as this report intends to show.
The proceedings of the workshop are available as a Laforia internal report (#94/24), by ftp (site ftp.ibp.fr, login anonymous, under /ibp/softs/laforia/NeOpus/proceedingsEOOPS), or on the web at:
http://www-laforia.ibp.fr/~fdp/eoops.html

Principal organizer

François Pachet, Laforia-IBP, University of Paris 6, France.
E-mail: pachet@laforia.ibp.fr.

Co-organizers

Patrick Albert, ILOG S.A., France
Brian Barry, Object Technology International, Inc, Canada
Brian Donnell, NASA/Johnson Space Center
John McGehee, Texas Instruments
Dan Miranker, University of Texas at Austin

Participants

Patrick Albert, ILOG S.A., France
Vladimir Bacvanski, Aachen University of technology, Germany
Brian Barry, Object Technology International, Inc, Canada
Jacques Bouaud, CHU Paris 6, France
James Crawford, Computational Intelligence Research Lab., Univ. of Oregon
Fred A. Cummins, Object Oriented and AI services
Daniel Dvorak, AT&T Bell Laboratories
Jiri Dvorak, Swiss Scientific Computing Center, Switzerland
Charles L. Forgy, Production Systems Technologies, Inc
Dan Miranker, University of Texas at Austin
Anil Mishra, AT&T Bell Laboratories
François Pachet, Laforia-IBP, University of Paris 6, France
Jean-François Perrot, Laforia-IBP, University of Paris 6, France

Presentations

The workshop was organized around 9 presentations, followed by a general discussion on emerging common themes. The presentations were grouped as follows:

Morning: presentations in which EOOPS are seen mainly as extensions of *Rete-based* systems. These presentations were (superficially) divided into three groupe: 1) 3 presentations of C++ systems, 2) 2 presentations of Smalltalk systems, and 3) one theoretical approach.
Afternoon: three presentations using Non-Rete approaches. Followed by a general discussion on the classification of EOOPS.

Overview of the presentations

We only give here an overview of the presentations and main arguments and refer to the paper in the workshop proceedings (Cf. abstract) for more details.

Charles Forgy: The RAL/C++ system

The presentation of Charles Forgy (the inventor of the original Rete algorithm, now 18 years old, and used today in most of the available inference engines) focused on the recent developments of his system RAL/C++. RAL/C++ introduces several new mechanisms, compared to the standard OPS systems. It is to be seen as a library of subroutines for rule-based programming, rather than as a fully-fledged engine. It is presented as being on a lower level than the original OPS systems. In RAL/C++, right-hand sides of rules are similar to C++ function bodies, and may include any legal C code. Rule variables are typed by their class. Modifications to the working memory are explicitly declared as such by a *modified* statement. The novel features of RAL/C++ include a scheme to allow "recursive rules", i.e. the ability for rule sets to be called recursively from the action part of a rule.

Patrick Albert: Ilog Rules

Patrick presented the Ilog Rules system (follower of a previous system called XRete). He insisted on the importance of "not reinventing a new language", which led Ilog to build a rule system that sticks as much as possible to C++ semantics (as most Ilog products do). In Ilog Rules, the progammer must reference slots explicitly in left-hand sides of rules, in order to guarantee correct re-computation of the conflict set after modifications of the working memory. Ilog Rules addresses the notification problem by introducing a special construct (called *defimplementation*), that automatically generates accessor methods, which includes automatic notifications to the inference engine. This construct may be seen as a re-specification of those classes that are used in the rules. Contrary to Forgy's scheme (which consists in a pre-processing of rule code), Patrick advocates the duplication of information, by means of these automatic accessors. He argues that this redundancy allows to avoid possible conflicts when using external class libraries (which themselves may require the use of pre-processors). Patrick suggested the possibility of choosing between several notification models, depending on the application. In particular, some applications contain already sophisticated notification models (such as dependency models), which could be re-used for inference purposes. He also stressed on the fact that some important data model relations are not yet supported in any of the EOOPS systems so far, such as *n:m* relation (e.g. the parent/children relation).

Dan Miranker: Encapsulating rules

Dan introduced *lazy matching*, a rule compilation scheme used in his system Venus. Lazy matching is based on the remark that, in Rete, most of the computed instanciated rules are never fired. The idea is to compute only the one instanciation that will be fired, by "compiling" the conflict resolution strategy inside the instanciation algorithm. This is performed by introducing an algebra of cursors that guarantees the correctness of the algorithm. Lazy matching drastically reduces the space and time required to compute instanciations. Venus also includes a declarative scheme to express hierarchies of collections of rules, that reduces the number of premisses in rules, by factoring out common premisses.
Dan argues that lazy matching does not solve the notification problem, but substantially reduces the number of situations that require a notification.

Brian Barry: ENVY/Expert

Brian introduced the ENVY/Expert project, aimed at building a framework for embedded production systems in Smalltalk-based applications. One of the major difference with other approaches is that ENVY/Expert provides a powerful version management scheme for rules and rule bases, that is directly inherited from the ENVY/Developer's one. Brian also insisted on the decomposition of ENVY/Expert into a framework with reusable and pluggable components (the parser, the builder, implementation, inferencing, etc.) Brian engaged a discussion about debuggers and steppers, and described the musical score tracer of Envy/Expert. This triggered a discussion on the feasibility of this scheme with large (over 10000) number of rules and rule instanciations.

Jean-François Perrot: the NéOpus system

Jean-François described the NéOpus system, a continuation of the Opus system described at OOPSLA in 87. He insisted on the representation of rule bases as abstract classes, which yields a fairly natural inheritance mechanism for rule bases, that allows factoring common rule between rule bases, and translates the intuition of OOP inheritance in the world of rules. The talk concentrated on the NéOpus solution to the control problem, that consists in specifying the control strategy with a special rule base operating on particular objects, called evaluators. Rule base inheritance is extensively used to specify control rule bases. Jean-François compared it with the approach of Dan Miranker, that consists in burying the control strategy in the instanciation algorithm.
Patrick Albert asked what "instantiating a rule base" could mean in the NéOpus context. Jean-François replied that reifying rules and rule bases is not easy, and that we therefore should refrain from systematically doing so. In NéOpus, the rules are not reified, but fireable rules are, thanks to the Rete mechanism, which yields a clean definition of fireable rules as a pair (*successful token*, *final Rete node*).

A discussion on the ontological nature of rules and fireable rules followed, raised by a question by Brian Barry: "are rule bases classes or objects ?". Dan Miranker suggested that rules are syntax, and that Rete gives a semantic for their behavior.

This question was finally summarized as: should we see Rete as:

(Jean-François) an "ontological transformation" (from rules to fireable rules) *versus*
(Dan) an "operational semantics" (of rules) ?

Jean-François proposed a final argument: rules are "declarative" only when they are reified: only then can they be the subject of much more manipulation than mere procedures (which can only support compilation).

Jacques Bouaud: Behavioral match

Jacques introduced behavioral match as the key to a correct integration of objects and production rules in a compiled setting. Jacques showed that most Rete-based rule systems rely on so-called *structural match*, i.e. the matching between structures of facts and structural patterns, which is contradictory with encapsulation. Jacques defined the *behavioral match problem* as the incremental updating of alpha-memories (i.e. the set of working memory elements that satisfy a given condition). He proposed two solutions: a naive solution which consists in re-computing all alpha-memories at each cycle, and a less naive solution which relies on the systematic management of dependency dictionaries.

Brian Barry objected that this problem is not tractable in general, and is comparable to the well-known problem of building a "stripped" Smalltalk image. Only approximations are feasible in practice, and Brian has a strong intuition that the problem is not solvable (too exponential).
Patrick Albert argued that the behavioral match problem is not specific to EOOPS, and is relevant also for arbitrary procedural programs.

Jiri Dvorak: two perspectives on the EOOPS theme

Jiri presented two EOOPS systems. The first one was built using CLIPS. The main problem discussed was the interconnection between 1) CLIPS, 2) COOL, the OO language used to represent facts in CLIPS, and 3) C++, the application language. COOL is not usable directly as a programming language, and is rather to be seen as an intermediate representation layer. This yields a "language mismatch" between COOL and C++. Jiri proposed *wrappers* to wrap COOL objects on the C++ side. Several related typing and interconnection problems between both object languages were discussed.

James Crawford: Path-Based Production Rules

James presented a controversial approach to EOOPS (at least in a Forgy/Miranker setting!), with path-based production systems. He showed how the underlying assumptions of path-based systems are exactly opposite to that of Pachet&Perrot, i.e. :
- rule are expressed with access methods only instead of arbitrary methods,
- rules match one chief object (the receiver) instead of an arbitrary t-uple of objects,
- use simple direct compilation instead of complex Rete-based compilation.

James introduced the notion of "relevant change", including both slot modification and set add/remove operations. However, the notification problem is irrelevant in this setting as only slots explicitly mentioned in action parts are re-computed. James also listed categories where path-based EOOPS proved useful: 1) enforcing invariants (the "rectangl becomes square" example), 2) monitoring state changes, 3) propagate changes, 4) traditional Expert System applications, 5) others, such as GUI updates, or pointer management.

Jean-François Perrot made a point here: what we are all referring to when talking about rules and objects is a kind of "unnamed entity" that materializes the t-uple of objects matching a rule. In Rete-based systems, this entity is represented explicitly by the token, whereas in path-based rules the programmer is expected to build himself the theory of that entity, which is a reasonable assumption in a number of cases.
Charles Forgy objected that may be this is reasonable in 95 % of cases, but what about the remaining 5 % ?

Fred A. Cummins: Production System extensions to KSL

KSL is a reflexive system in which language expressions are first-class objects. Inferencing is performed with a network similar in spirit to the Rete network (although not an official descendent). Fred introduced "aspects" which are abstractions of instance variables and methods. Relationships between aspects are then expressed, and used to generate automatically methods with notification.

Discussions: towards a classification of EOOPS ?

Here is a list of the main themes discussed during the presentations:
1 - *Rete versus non Rete*,
2 - *The notification problem*, and the various ways of handling it:
- path-based: limiting rules to only access method
- explicit declaration of slots (RAL/C++)
- automatic accessors (Ilog Rules)
- generalized modified (NéOpus)
- lazy matching: limiting the number of situations requiring notification (Venus)
- behavioral match: attempt to address the problem in its generality

3 - *Efficiency*. RAL/C++ and Venus are 10 times faster than CLIPS, itself 100 times faster than the naive algorithm. When is that speed really needed ? Can we really do without it ?
4 - What is the *list of the common features* of EOOPS (e.g. taking class inheritance into account in rule variables)
5 - What are the *assumptions concerning the "openness" of the world*, especially with regards to the scope of the inference process (this is especially significant when data bases are used as working memories).

Finally, Jean-François Perrot suggested two schemes for classifying EOOPS systems according to the various topics discussed during the presentations.

First classification scheme:
Depending on applications, two main cases occur:

1 - *Rules are essential. Objects are subservient*. This is what classical AI systems such as Art or KEE deal with.

2 - *Objects are essential* (existing applications). *Rules are subservient*. Two subcases:

2.1 - Rule bases seen as "procedures" attached to particular classes.

2.2 - Rules as "constraints". This yields parallelism and control problems.

Second classification scheme:
If we take rules talking about objects, there are two main cases:

1 - The rule talks about *one chief object* and several dependent objects. This is addressed by path-based production rules. Note that the notification problem is still relevant here.

2 - The rule talks about *several unrelated objects*. The question is then: is some kind of "naive cycle" fast enough ?

2.1 - Yes. Then interpreted engines (such as Essaim) should suffice, and there is no notification problem.

2.2 - No. (Rete-like) compilation is needed. And the notification problem reappears.

Conclusion

The notification problem appeared as the main technical issue for the implementation of EOOPS, and is still not solved in its generality. There does not seem to be a consensus on the possibility or impossibility of technically solving it either, and each system imposes specific constraints on the language to solve the problem.

Future

A synthesis article is in progress. A next workshop with the same people could concentrate on methodological issues of EOOPS, as well as on the latest exciting developments of the systems presented here.