# ENVY®/Expert: An embedded reasoning system in Smalltalk

a position paper for the
OOPSLA '94 Embedded Object-Oriented Production Systems Workshop

Jeff McAffer, Ted O'Grady and Brian Barry
Object Technology International
{jeff, ted, brian}@oti.on.ca

Until recently, most expert system or production system technology was available only in so-called Expert System Shells (ESSs) which were stand-alone, self-contained systems. They were designed to be all encompassing environments in which users defined and used knowledge. As such, they could offer only a very narrow range of interaction and integration with other programs.

Systems like CLIPS [NASA] addressed this problem in part by defining what is essentially a user callable reasoning library and allowing the library to call user-defined functions or programs. Still, there was a relatively strong separation between the CLIPS knowledge space and the user program's knowledge space. CLIPS knowledge and user program knowledge could not be directly shared and interchanged.

This was not so much a deficiency in CLIPS but in its implementation environment -- that is, an inherently non-object-oriented system with object extensions (C/C++). We believe that the full and complete integration of production systems (PSs) into logically consistent object-oriented environments will lead to a marked increase in the usability and in fact usefulness of PSs in end-user applications.

There are many ways in which object-oriented (OO) technology and production systems compliment each other. not be a set form or structure to what knowledge (e.g., rules and facts[1] ) can be asserted. Any object can be a fact and regular code can be rule antecedents and conclusions.

The everything-is-an-object point of view enables the running of inference engines to be the subject matter of a PS just as easily as say medical diagnosis. This gives us meta-level control over the reasoning process and requires no special meta-mechanisms (other than openness). Inference engines become essentially context holders and execution descriptors for rules. As such, they are dynamically adaptable and interchangeable.

---

[1] The term fact is problematic in the context of object-oriented reasoning systems but is used here to avoid the wanton construction of new terms. See the section on terminology below.

In theory, typical frame-based systems The use of objects brings to the table a generality of assertions. That is, there need can represent complex knowledge structures. In practice, these systems are not sophisticated enough to allow users to create and maintain these structures. Pure OO systems on the other hand, excel in this area. As a result, PS users can write rules in terms of complex, semantically rich objects.

The flow of benefits is not one way. OO systems are often used in complex dynamic environments where the behaviours are best described in a declarative, heuristic manner. Integrated PS/OO environments enable the description of these behaviours within a consistent domain space.

The goal of our work is to provide modular and embeddable PS reasoning capabilities so closely integrated to the underlying OO environment mechanism as to be virtually indiscernible.

# 1. ENVY/Expert

Work on **ENVY**/Expert [Barry and McAffer 92] started several years ago with a CLIPS-like reasoning system implemented in Smalltalk. In the design of the system we paid much attention to the partitioning of the various system components to enhance embeddability. The result was a highly modular or pluggable inferencing architecture. While application (i.e. normal Smalltalk) objects and code could readily use and be used in the reasoning process, the implementation lacked full integration with the Smalltalk execution system. Rules were maintained in a separate format and required something akin to interpretation to be run.

The next version of **ENVY**/Expert was completely different in that it focused mainly on the integration of rule execution with the underlying virtual machine at the expense of modularity and embeddability. This version was heavily influenced by NéOpus [Paçhet 92] and used similar implementation techniques.

Our recent work on **ENVY**/Expert has been directed at the integration of the two earlier versions with the aim of building an embedded OO production system having the following three properties:

1. Close integration with the underlying OO mechanisms.

2. Clear partitioning of the PS components to isolate knowledge acquisition, knowledge construction and reasoning.

3. Open and clear definition of inferencing infrastructure and mechanisms to promote inference engine interoperability.

## 1.1 Terminology

We have explicitly tried to avoid creating new terms but rather explicitly state our interpretation of existing terms. Fact and knowledge are of particular interest here.

Fact: Any object which is asserted in the context of some inference engine. While this definition is largely the same as others, we explicitly avoid saying anything about truth, but rather speak of existence -- some object, the fact, exists (is asserted) in some context.

Knowledge: Facts and rules. Many previous uses of this term have addressed only one or the other. We feel that this has been largely a result of implementation influences. Clearly both facts and rules are objects which represent knowledge.

Further to these viewpoints and since some reasoning strategies (e.g., Rete [Forgy 82]) do not require explicitly separate rule and fact bases, we can eliminate the notions of rule and fact bases as separate entities and unify them as knowledge bases. This simplifies the design and enhances modularity and interoperability.

## 2. Integration

The current version of **ENVY**/Expert is completely integrated with the Smalltalk environment. Like NéOpus, rules are represented by Smalltalk methods which are executed directly by the Smalltalk virtual machine. The knowledge description language is very similar to Smalltalk having only minor modifications for argument typing and demarking premises and conclusions etc. We defer detailed discussion of the implementation and refer readers to documents related to the NéOpus system.

| | |
|---|---|
| Language | Smalltalk parser. Generates a parse tree. |
| Specification | Smalltalk CompiledMethod without bytecodes. Represents the logical method. Contains higher (than Implementation) level information (e.g., selector). Focal point for maintaining correspondence between user (Language) and machine (Implementation) levels. |
| Builder | Smalltalk compiler code generator. Generates, an executable form in terms of the Implementation. |
| Implementation | Smalltalk's bytecodes and primitive mechanisms. |
| Inferencing | Smalltalk Virtual Machine or process thread. Maintains context as execution jumps from method to method (rule to rule). |
| Management | Smalltalk's classes, categories, source files and version management facilities. |

Figure 1: Analogy between **ENVY**/Expert and Smalltalk system components

## 3. Partitioning

In an effort to increase embeddability, modularity and interoperability, **ENVY**/Expert is divided into six discrete subcomponents: Language, Specification, Builder, Implementation, Inferencing and Management. By clearly defining the roles and interfaces of each component we greatly increase the capacity of the system as a whole. Figure 1 illustrates how these components fit together (and the logic for their existence), by drawing analogies with the Smalltalk system itself.

### 3.1 Language

The Language component defines the syntax of user knowledge specification. It is also responsible for converting textual specifications into some standard (within **ENVY**/Expert) representation for further processing. Having Language as an explicitly separate component allows for its easy replacement and thus the adaptation of the system to different user views of knowledge and its expression.

### 3.2 Specification

The Specification component represents an intermediate step between the Language and the Implementation. By providing this step we isolate the language from the implementation. The parsed form of the language is converted to generic knowledge objects such as Rule and Fact. These objects are abstract in the sense that they are not executable. They are the primary objects of user manipulation.

### 3.3 Builder

The Builder is closely linked with the Implementation. Its role is to convert implementation independent specifications into objects which are executable under a particular implementation. While Builders are Implementation specific, their correspondence need not be one-to-one. Many Implementations may be built using the same techniques and objects but simply execute differently.

Besides *pluggability*, one of the main reason for separating out this element is that it is not necessary for the execution of rule-static PSs. When embedding a fixed and known set of rules in a user application, we need not include the Builder objects.

### 3.4 Implementation

The Implementation subcomponent is responsible for defining an executable representation of knowledge. Though there is not necessarily a direct correlation, the objects in the Implementation are analogous to those in the Specification but structured or tuned for a particular inferencing technique. For example, in straightforward RETE style reasoners, a path through the RETE network nodes is equivalent to the antecedents of the Rule object corresponding to the left/termination node.

### 3.5 Inferencing

The Inferencing subsystem defines the objects required for actually executing a particular inferencing scheme. These consist largely of an inference engine responsible for directing rule execution and maintaining the context in which they execute. Inherent in this are objects such as ConflictSets and Activations. Again, while Implementation and Inferencing are closely related, there need not be a 1:1 correspondence. A complex inference engine might consider rules from many different rulebases each having a different implementation.

### 3.6 Management

The Management component manages the objects at the Specification level. Since the Specification objects are the primary point of manipulation by users, issues such as the storage of rule source, version management and the organization of rules and facts into logical groups or knowledge bases is of primary importance.

## 4. Interoperability

The particular breakdown of subsystems we have chosen encapsulates the PS functionality into discrete components. These components have well-defined interfaces allowing us to substitute or plug-in new components and thus extend or modify the behaviour and capability of the production system. For example, if a user's application knowledge is best expressed using certain knowledge language, we need only create a parser for that language. As long as it can produce a suitably formatted parse tree, no other part of the system will require changes. Similarly, the execution mechanism of a defined rulebase can be changed without affecting its rule source or specification.

## 5. Conclusions

We have briefly introduced the design and implementation of a production system (**ENVY**/Expert) embedded in and integrated with a comprehensive object-oriented programming environment (Smalltalk). **ENVY**/Expert's inferencing technology is strongly based on Paçhet's NéOpus system. The implementation architecture presents a generalized framework for integrating and embedding production systems into the Smalltalk environment and OO environments in general. Using **ENVY**/Expert, programmers can embed high-quality reasoning techniques in their applications.

## 6. References

[Barry and McAffer 92] "**ENVY**/Expert: A rule-based programming system for Smalltalk", *Proceedings of the Canadian AI Conference*, May 1992.

[Forgy 82] C.L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem". *Artificial Intelligence* 19, pp. 17-32, 1982.

[NASA] *CLIPS Reference Manual*, Software Technology Branch, Lyndon B. Johnson Space Center.

[Paçhet 92] Paçhet, F., *Représentation de connaissances par objets et règles: le système NéOpus*, Report 92/30, LAFORIA, Institut Blaise Pascal, Paris.