オブジェクト指向システムでの推論

Reasoning in object-oriented systems

Jeff McAffer *

Summary.

The increasing demand to use object-oriented techniques and rule-based reasoning in real industrial applications is not met by today's technology. Monolithic reasoning shells have inferior object models and object-oriented systems have inferior reasoning tools. We present a collection of Smalltalk reasoning classes which are completely integrated with the underlying object system. We detail how such a toolset can be applied in end-user applications, meta-level processing and advanced language features research. **ENVY**/*Expert*, our component toolkit, is shown to be comparable to existing reasoning tools at the reasoning level but far more advanced in object and meta-object facilities.

1 Introduction

The increasing demand to use object-oriented techniques and rule-based reasoning in real industrial applications is not met by today's systems. Rule-based reasoners are monolithic, closed and are not well integrated into object-oriented tools. In this paper we report on the development of **ENVY**[1]/*Expert*, an object-oriented inferencing toolkit designed to rectify this situation. Our system is implemented in Smalltalk and allows rule-based reasoning to be seamlessly integrated into real industrial applications as well as used in advanced language research. It has capabilities equivalent to toolkits such as NASA's CLIPS [NASA 91] but is fully object-oriented and more open, adaptable and flexible.

In the following sections we discuss the applicability and benefits of object-oriented programming to rule-based reasoning and the uses of reasoning in

* Object Technology International and Department of Information Science, The University of Tokyo. `jeff@is.s.u-tokyo.ac.jp`

[1] **ENVY** is a registered trademark of Object Technology International Inc.

object-oriented system research and implementation. An outline of the system design and some implementation experiences are also given. We relate our design concepts and decisions to real-world problems in real-time and embedded computing, research issues in computational reflection as well as the control and analysis of massively parallel computations. A brief comparison of **ENVY**/*Expert* to CLIPS and a discussion of our future plans is also given.

2 Application-level reasoning

Traditional expert systems are rule-based and represent factual knowledge using *patterns* and *frames*. A pattern is an array of literal values (e.g., numbers and symbols) and variables which are matched based on their physical location. Patterns are insufficient for complex reasoning tasks as they can represent only simple data configurations and have no sharing or abstraction capabilities.

The concept of a frame for knowledge representation was introduced in the 1970's by Minsky. A frame is a collection of *slots* which contain literal values and other frames. Slots themselves can have features such as constraints, default values, etc. Frames are similar to objects but do not offer generalized behaviour specification. Objects are a superset of the common factual knowledge representation techniques and offer the following benefits.

- Application specific objects are reasoned about directly without either having to encapsulate them in, or re-constitute them as, frames.
- The full power of behaviour specification techniques (e.g., inheritance and delegation) can be exploited in the reasoning mechanism, the knowledge representation and the application.
- Applications can contain many distinct lightweight expert systems.
- Inferencing tools are dynamically modifiable. This is essential for open, flexible, reflective reasoning.
- Objects enable a flexible model of concurrency within expert systems.
- Modularized knowledge domains improve reuse and maintainability.
- Performance improvement through selective use of knowledge domains.
- Sophisticated, complex and functional facts can be represented.

ENVY/*Expert* seamlessly integrates declaritive rule-based and object-oriented processing. Rule activations invoke Smalltalk message sends and Smalltalk message sends can result in rule activations and fact assertions. Rule conditions are conjunctions of arbitrary Smalltalk message results.

This opens a whole range of application areas previously closed to object-oriented technology. For example, **ENVY**/*Expert* is currently being used by the Canadian Space Agency to build a real-time embedded satellite power system health monitor. The monitor requires both reasoning and conventional computational capabilities. Without integrated Smalltalk reasoning tools, it would be implemented partially in CLIPS and partially in Smalltalk. Fundamental incompatibilities between the object models and computing paradigms would force object conversion overhead, inflexibility and design compromise. Code and knowledge maintenance would be significantly impaired.

Reasoning technology is also applicable to our research related to the monitoring and analysis of massively parallel system behaviour. The highly dynamic and variable nature of these systems requires flexible and adaptable monitoring and analysis tools. Reasoning technology is used here to implement:

- Intelligent probes which actively seek out information or behaviour.
- Expert systems to identify behaviours and interactions
- Declaritive rule definition of the global control of the monitoring system.

3 Meta-level processing

In addition to expanding the end-user domain horizons for reasoning technology, object-oriented reasoning is applicable at the system meta-level. They add a new, declarative, dimension to meta-reasoning and meta-programming.

Meta-reasoning is control of, or reasoning about, reasoning. For example, an expert system whose domain is the running of expert systems is a meta-reasoner. The rule shown in Figure 1 contains meta-knowledge about the chaining behaviour of bidirectional inference engines. If the engine spends too much time searching the solution space (i.e., chaining) in one direction, this rule will reverse its direction. Note that the conditions for determining how far is "too far" are quite independent of the actual application domain and so this rule can be applied to many different inference engines in many different situations.

ENVY/*Expert*'s open architecture directly supports reflection by reifying the inferencing tools and components into first-class objects and enabling any number of them to be used concurrently. Since rules are compositions of regular Smalltalk message sends and the inferencing tools themselves are Smalltalk objects, the entire reasoning process is available for observation and manipulation. Figure 2 shows how this idea can be extended to a number of meta-levels.

```
chainingControl

    | engine {InferenceEngine} |
    engine chainLength > 75.
=>
    engine reverseChainingDirection
```

Figure 1. Chaining control meta-rule

In the bottom right corner of the figure is the end-user application domain, say, signal processing. Inference engine 1 manipulates the knowledge in this domain thus here we say that the application is the *target* of inference engine 1 and signal processing is its domain. Implicit is the fact that its associated knowledge bases contain signal processing knowledge. At the next level up, we see a similar structure but this time, it is inference engine 1 which is the target of inference engine 2. The domain of level 2 is the reasoning process itself. This level is the meta-level of inference engine 1 (i.e., level 1). It would contain rules such as that shown in Figure 1 which relate directly to level 1's reasoning. Meta-level 3 (in the upper-left corner) targets meta-level 2 and deals in the domain of general system control and coordination (e.g., communications routing and message passing style).

In this particular example there is a linear one-to-one relationship between the meta-levels. While logically this always holds, it need not be the case physically. For example, the control domain of level 3 is equally applicable to inference engine 1 or the end-user application itself. One meta-level reasoner can control many inferencing tools. Also, the actual decision mechanisms used at each level can be vastly different (i.e., any combination of expert systems, regular Smalltalk objects and user designed schemes). As shown in the figure, the reasoning process (i.e., **ENVY**/*Expert*) is just another application domain for **ENVY**/*Expert*.

We use meta-reasoning facilities in our object-based concurrent programming research. Object-oriented and reflective (i.e., meta-level) computing are of great interest to people building languages for massively parallel systems. In these systems, global control of individual active objects is undesirable or impossible. Control (e.g., load balancing, scheduling) should be handled locally by delegation to individual or groups of objects. We propose that control be described in whole or part using rule-based declaritive knowledge operating at the meta-level. Rules can be used to define anything from synchronization and timing mechanisms to scheduling to inheritance. Object and

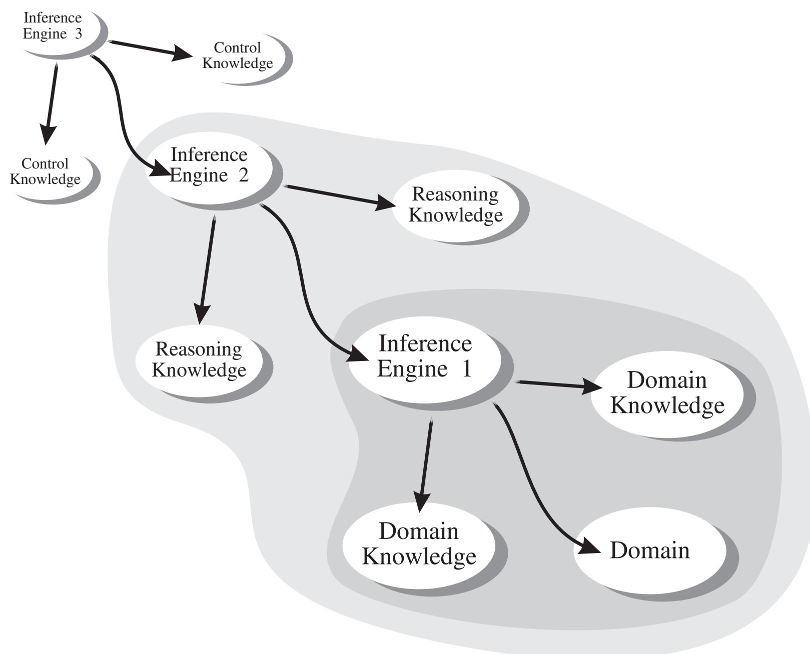

Figure 2. Example meta reasoning structure

group-wide reflection mechanisms [Ichisugi 91, Watanabe 91] provide access to object behaviour and our reasoning tools provide control decision capabilities based on user-defined rules. This configuration enables experimention with object behaviours and interactions by varying the declarative control rules for individual objects. This is independent of the object's domain processing.

4 ENVY/*Expert*

ENVY/*Expert* is an inferencing toolkit written in and fully integrated with Smalltalk. The overall approach was to define a set of generic objects and protocols which are typical to a wide range of reasoning techniques. These are then extended and combined to form particular inferencing tools. Figure 3 shows each of the four main subsystems along with the objects and interfaces they define.

The Language subsystem defines the textual syntax and the parsed structure of all the knowledge constructs represented and processed in **ENVY**/*Expert*. Parsing knowledge text results in a standard format parse tree which is input to a knowledge builder. Language syntax is independent of inferencing technology and thus the same knowledge specification can be used with

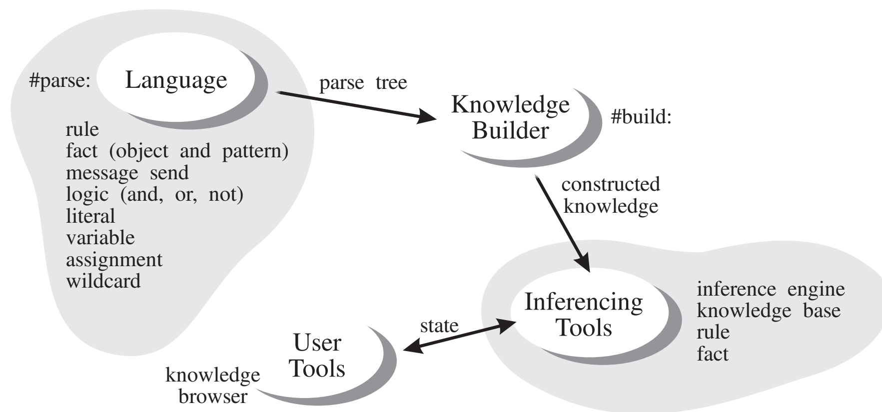

Figure 3. ENVY/*Expert* system architecture

several different inferencing tools and vice versa.

The Knowledge Builder translates from the parse tree of Language objects to the fact and rule representation of a particular Inferencing Tool. Internally it defines a #build method for each of the applicable language objects. Knowledge is built by traversing the parse tree asking nodes to build themselves. As the internal representation of knowledge varies between inferencing mechanisms, it is typical that builders be specific to particular inferencing tools.

User Tools defines general browsers for manipulating knowledge bases, facts, rules and inference engines. They provide functionality equivalent to the Smalltalk class browser, inspector and debugger.

Inferencing Tools is the runtime system for **ENVY**/*Expert*. Figure 4 shows its main components and their essential protocol (housekeeping methods are not included). Facts are arbitrary objects. Rules have an antecedent or set of conditions and a set of conclusions or actions. Both consist of arbitrary Smalltalk message sends and fact specifications. Knowledge Bases keep track of the currently *activated* rules. In forward-chaining, a rule is *activated* when all of its conditions are satisfied (i.e., return true) while in backward-chaining, a rule is activated when one or more of its actions *can* result in one of the current goals.

The basic execution model is very simple. The inference engine repeatedly asks its knowledge bases (it may have many) for their #nextActivation. An activation is a rule and the context (i.e., set of facts and variable bindings) in which the rule was satisfied. After determining which activation to run, the inference engine sends #executeFor: to the rule and passes in the activation context. Execution of a rule in forward-chaining mode evaluates (sends the messages in) the rule's conclusions while in backward-chaining mode the rule's antecedent condition messages are resolved as goals. Both antecedents and conclusions are made of message sends and can therefore #assert: and #retract: facts, #add: or #remove: rules, and perform arbitrary processing.

Meta-level control of the inferencing process manipulates the characteristics of the inference engine and knowledge bases. For example, the order in which the engine asks its knowledge bases for activations, the activation conflict resolution strategies and chaining direction are all subject to meta-level control.

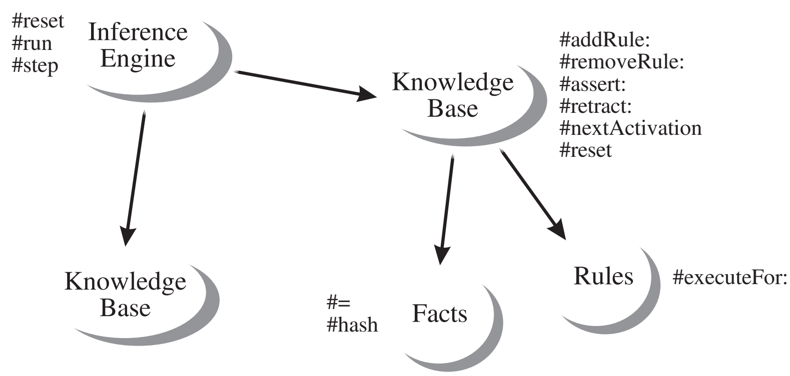

Figure 4. **ENVY**/*Expert* runtime architecture

4.1 Implementation experience

ENVY/*Expert* is very open, flexible and modular. Extending a particular component of **ENVY**/*Expert* is done in the typical object-oriented way by adding new classes, subclassing base component classes and providing new method definitions for the base protocol where necessary.

For example, we have implemented a forward-chaining inference engine based on the Rete pattern matcher [Forgy 82] and using a CLIPS-like knowledge definition language. Below are the steps we followed in building the Rete inferencing system from the base **ENVY**/*Expert* components. Future inferencing systems can reuse the Rete components and say, just replace the Language or add a bi-directional inference engine. Either would require addtional code but relatively few changes to the current system.

- Implemented a CLIPS language parser and made minor changes to #parse:. No parse tree node types were added or changed.
- Designed and implemented the objects particular to the Rete algorithm (e.g., pattern matching network nodes) and defined #build methods to

transform parse trees into these Rete knowledge representation objects. Knowledge Builder>>#build: was also redefined.

- Subclassed most of the Inferencing Tools base classes to add state required by the Rete pattern matcher. Made minor changes to Inference Engine>>#step, and #assert:, #retract:, #nextActivation and #reset in Knowledge Base.

During the implementation we encountered a number of unexpected issues in areas such as; fact equality vs. identity, hash functions, and access to the Smalltalk virtual machine. Object equality versus identity was a problem both for reasons of consistency and efficiency. It is generally accepted that rules express value-based knowledge. Therefore, fact bases and pattern matchers should be based on equality rather than identity. However, arbitrarily complex object equality tests should be avoided for efficiency. We resolve this in **ENVY**/*Expert*, by asserting facts into fact bases using equality tests. They are then passed to the pattern matcher. The fact base insertion eliminates duplicates allows the pattern matcher to use only identity tests.

Using hashed sets for the fact base requires special attention to fact object hash functions. For instance, in one benchmark it was found that 60% of the execution time was spent looking up facts in the fact base. The problem was that all the facts were two element Arrays and Array>>#hash simply returned the receiver's size. Hashed lookup became a linear search due to collisions. Fixing the hash function eliminated almost all of this search time and thus doubled execution speed. Future class libraries should expand the parameterization concept seen in SortedCollection and allow set hash functions and containment tests to be user-configurable.

The ability to directly invoke arbitrary compiled methods without method lookup would also be useful. Currently, the Rete pattern matcher tests are constructed with different classes of objects each of which knows how to do a particular action. On receiving the #execute message, it performs its operations. If we could invoke arbitrary methods directly then these objects could be eliminated and the tests compiled directly to Smalltalk methods. This would result in faster and cleaner inferencing code.

4.2 Comparison to CLIPS

We estimate that the current version of **ENVY**/*Expert* has approximately 60-70% of the CLIPS capability not including our extra features. CLIPS is implemented in C and totals approximately 3 MBytes of source code including comments. This represents many person-years of work in design, coding and optimization. **ENVY**/*Expert* on the other hand, is written in Smalltalk and is about 350 KBytes of source code including comments. The entire design and implementation required 6 person-months. No serious optimization work has been done to date.

To compare execution speed, we used some CLIPS examples which exercise basic fact assertion, simple condition testing and rule execution. In these tests, CLIPS ran five times faster than **ENVY**/*Expert*. This is reasonable performance considering our lack of optimization effort. Also, **ENVY**/*Expert* is targeted at object-oriented reasoning while CLIPS is aimed at pattern-based reasoning. Since the benchmark uses only pattern-based facts (i.e., no objects), CLIPS' better performance is understandable. We expect CLIPS performance to decline with increased use of objects and **ENVY**/*Expert*'s to remain relatively constant but have yet to construct reasonable object-based benchmarks to confirm this. Readers should note that CLIPS performance declined with successive benchmark runs and the system eventually crashed due to memory leakage.

5 Future work

ENVY/*Expert* will continue to be a valuable tool in our object-oriented concurrency research and we look forward to it being applied in a broad range of problem domains from intelligent assistants to process monitoring and control. We are also investigating a variety of new inferencing and reasoning techniques to meet coming needs. These include case-based reasoning, truth maintenance and backward and bidirectional chaining inferencing. We are currently looking at replacing the CLIPS-like language with something like that shown in Figure 1 which is more familiar to Smalltalk programmers.

参考文献

[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.

[Ichisugi 91] Yuuji Ichisugi, *et al*, "An object-oriented concurrent reflective architecture for distributed computing environments", In *Proceedings of 29th Annual Allerton Conference on Communication, Control and Computing*, Allerton Illinois, 1991.

[NASA 91] *CLIPS Reference Manual*, Software Technology Branch, Lyndon B. Johnson Space Center, 1991.

[Watanabe 91] Takuo Watanabe and Akinori Yonezawa, "An actor-based meta-level architecture for group-wide reflection". In *The proceedings of the Foundations of Object-Oriented Languages Workshop*, Lecture Notes in Computer Science 489, Springer-Verlag, pp. 405–425.