Pebbles and Jess: A Marriage of Convenience?

Atif Alvi
Computer Laboratory, University of Cambridge
atif.alvi@cl.cam.ac.uk

Abstract

We study the feasibility of using Jess (Java Expert System Shell) as a building block in our rule-based architecture for home environment [1]. Jess is an attractive candidate: it is a rule engine written in Java; can process a large number of rules quickly; and is a stable software product. The outstanding but resolvable issues for this integration are: Jess's appetite for memory; distributing Jess; rule validation; allowing communication or conversion between application programs and rules; and making Jess plug 'n' play with respect to devices.

1. Introduction

We are developing a functionally monolithic rule-based controller for the home environment where a changing population of devices is governed by a rule-set of variable cardinality. More than one system design is under consideration and this writeup refers to the framework described in [1]. The major components of this design are registry, rulebase, rule validator and execution engine. We are looking for ways to implement these components suitably.

Jess is an expert system shell and scripting language written entirely in Java. It is a programmer's library in Java that serves as an interpreter for another language called the Jess language. The Jess language is very similar to the language defined by the CLIPS (C Language Integrated Production System) expert system shell [5], which in turn is a highly specialized form of LISP. Jess supports the development of rule-based expert systems which can be tightly coupled to code written in Java. At its heart is the Rete algorithm [2] — a time-efficient method for matching objects and patterns — that can be used to find all the rules with satisfied conditions (the *if*'s) after an event and then to fire those rules (the *then*'s).

Instead of writing a system from scratch, we seek to intelligently blend already available software components that have gone through extensive development cycles to be assured of their reliability and performance in such a sensitive context as one's own home.

In Section 2 we discuss the overall requirements for Pebbles. Section 3 overviews Jess' capabilities and Section 4 analyzes the issues concerning Jess deployment in Pebbles. We conclude in Section 5.

2. Pebbles Requirements

The Pebbles system is meant to control a multitude of embedded devices in a home environment. [1] achieves this by having a rule-based controller that comprises: registry (holds device details and current worldview), rulebase (stores rules), rule validator (checks for consistency among rules), and execution engine (responds to events by taking appropriate actions). Communication with the external world is done through an asynchronous eventing protocol, such as GENA, and an XML-style RPC protocol, such as XML RPC or SOAP. Predicate calculus is used to define rules and imperative style user applications either need to be compiled into universally quantified rules or they need to be executed remotely.

As Pebbles faces real life and real time scenarios, its reaction time should be quick in the presence of a large number of devices, applications and rules. In addition, suitable prioritizing and conflict resolution strategies must be at hand to model real situations and consequent actions.

3. Jess Capabilities

With Jess come all the portability and security advantages of Java along with the ability to write Java applications or applets. However, this also introduces Java's slower execution speed, but this is more than compensated for by the fast Rete algorithm it employs. The analytical time complexity of a rule firing is at worst $O(W^{2P-1})$ and at best $O(1)$, where W is the size of the working memory (global database holding all the data or facts) and P is the number of patterns (*if*'s) in a rule [2]. If we take the size of the memory taken up by rules (rule memory) as a complexity measure, then the time for one rule firing is at worst $O(R)$, where R is the total number of rules in rule memory (the best case is $O(\log_2 R)$). Results from a practical study are also encouraging where Jess was used to track the movements of 20 people with approximately 2 events/sec (sample frequency) generated. The response time was 0.20 sec on an 800MHz Intel Pentium III machine with 128MB RAM. This was almost ten times faster than the BAT tracking system in the Technology Group (LCE). In the same study, the speed was estimated to be linear in the number of facts and quadratic in the number of conjunctions in the rules.

Jess offers a range of possibilities in terms of the amount of Java code and pure Jess language scripts that the user wants to employ. At one end of the spectrum there can be pure Jess language scripts and at the other end pure Java code; usually, a mix of both is utilized.

Jess uses first order logic and supports both forward and backward chaining, unlike Prolog that uses only backward chaining. Backward chaining lends the system a 'goal seeking' behaviour whereas forward chaining looks for satisfied rules on its own after a fact is known.

Jess can also be used in a multithreaded environment. The *jess.Rete* class internally synchronizes itself using several synchronization locks.

4. Embedding Jess in Pebbles

Jess can be integrated into Pebbles with slight changes in the internals of the architecture proposed in [1]. The rule base and execution engine are already integrated into Jess. It also acts as a registry by registering devices and other entities as Java objects. The rule validator will still have to be written, modified for use with the Jess scripting language. RPC (SOAP) and eventing (GENA) can easily interact with Jess.

The problem of distributing the rulebase for efficient execution as the number of rules increases can be tackled by using, say, CORBA to enable different cooperating incarnations of Jess residing on different servers. However, this would not be necessary until the number of devices or rules grows very large because Jess works efficiently with several hundred objects with each object typically having between ten and one hundred attribute-value pairs.

A compiler for converting imperative-styled user applications into the Jess scripting language style of rules will have to be written. Some user applications can use the *javax.rules* API to hook directly into Jess.

The idea of assigning priorities to different rules is supported in Jess [4] in the form of *salience* of a rule. Salience values can be integers, global variables, or function calls. In the case of a tie in the salience values of rules, two different conflict resolution strategies are currently available (CLIPS has seven [5]): *depth (LIFO)* and *breadth (FIFO).* In either case, if several rules are activated simultaneously (i.e. by the same fact-assertion event) the order in which these several rules fire is unspecified, implementation-dependent and subject to change. More built-in strategies may be added to Jess in future. One can (perhaps) also implement one's own strategies in Jess [4].

To discover new devices in a plug 'n' play manner, Java code will have to be written that conforms to the UPnP standard for discovery and description.

Also, complete rule removal seems unsupported at this time although rules can be modified to anything in Jess.

5. Conclusions and Future Work

The analysis in this report shows that Jess, with a manageable amount of work, can be incorporated

into the Pebbles architecture. Further investigation needs to be carried out into the issues of distributing Jess and writing a compiler for converting application programs into Jess rules. Moreover, the tightness and ease of validation of Jess rules has to be compared with the pure logic approach of [1].

6. References

[1] D. J. Greaves and T. Omitola. Ideas for a Predicate Calculus, Rule-Based Controller. *Unfinished Draft*, 2005.

[2] C. L. Forgy. Rete: A Fast Algorithm for the ManyPattern/Many Object Pattern Match Problem. *Artificial Intellignce*, 19 (1982), pp. 17-37.

[3] E. Katsiri. Middleware Support for Context-Awareness in Sensor-Driven Systems. *Ph.D. Dissertation*, Computer Laboratory, University of Cambridge, *to be submitted*.

[4] E. J. Friedman-Hill. Jess, The Rule Engine for Java Platform. http://herzberg.ca.sandia.gov/jess/docs/70/, Version 7.0a5, Draft, February 2nd 2005

[5] CLIPS Reference Manual, Volume I, Basic Programming Guide. www.ghg.net/clips/download/documentation/bpg.pdf, Version 6.23, January 31st 2005.