Expert System Tools Exploration

March 10, 2009

Authors
Sean Lydon
Daniel Nelson
Emily Schwarz

Class
CPE 481 – Knowledge-Based Systems
Introduction

When we started the first programming assignments in Knowledge-Based Systems class at Cal Poly, we realized that both CLIPS and JESS (the current tools at the time) were not supported very well with either installation documentation or tutorials. They also didn't easily provide some desired features, so we started this project to search for and evaluate other existing other Expert System Development Tools (ESDTs) that might provide increased function and usability. These ESDTs could be in the form of either compiled or interpreted languages or complete environments that inputted application specific criteria to create a rule base and an inference engine.

From our analysis of ESDTs, we developed interesting teaching materials and this summary document. The teaching materials include presentations and example implementations which are directly applicable to the Knowledge-Based Systems class. Our findings are possibly causing a change in the tool availability for this class.

Methodology

We started by doing many, many Google searches, and this lead us to lists of possible ESDTs. Interestingly, most of the search results ended up at university websites for teaching Artificial Intelligence Expert Systems. We compiled an extensive list, and more than a couple of the results we found were one of the following: a research project, a Business Intelligence platform, or a Tech support generation platform. We then evaluated all possibilities on the list according to the following criteria:

Availability
Cost
Installation Documentation
Tutorial Documentation
Supportive Community

The results from this initial dig can be found in our results section. We also recognized that it
would be good to keep track of the evaluation and some history tid-bits for each of the ESDT's in a more verbose manner, because it is very interesting and it usually demonstrates the motivation for a particular implementation. This is compiled in the verbose evaluation section.

The initial evaluation narrowed down our list down quite a bit. The actual selection criteria was heavily weighed for applicability to the Knowledge-Based Systems class. From this list we did a further comparative analysis according to the following criteria:

- Usability
- Installation Process
- Applicability

From these results, we narrowed our results to two promising candidates. For these we actually attempted to do the first programming assignment (a family tree knowledge base) in each environment. From these experiments we developed teaching materials in the form of presentation slides, which were presented to the Knowledge-based Systems class.

**Results**

The following are our results in tabular form for each of our stages in the exploration. Table 1 contains the evaluation of the initial thirty-two ESDTs. Each of the evaluation criteria are rated on a sliding scale from 1 – 5, with 5 being excellent and 1 being poor. Wherever there is a “n/a” it means that the tool was not publicly available, so testing and/or research did not continue.

<table>
<thead>
<tr>
<th>ESDT</th>
<th>Availability</th>
<th>Cost</th>
<th>Inst. Docs.</th>
<th>Tutorials</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algernon-J</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Babylon</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DEC Reasoner</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Drools</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Euler Proof Mechanism</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Expect</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>F-Logic</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Initial Results
<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOCL</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G2</td>
<td>5</td>
<td>1</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Hank</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>InfoSapient</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E2glite</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>JADEX</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>JAM</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTP</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Jason</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Jeops</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>MIKE</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Manderx Int. Engine</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>OPS83 &amp; OPS5</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OpenCYC</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Poplog</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Prolog</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Protege</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>PyExpect</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ruleby</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>KAES</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SnePS</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Soar</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>TyRuBa</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XpertRule</td>
<td>5</td>
<td>1</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Tmycin</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

From the above table we selected the best ten to do a further analysis. Table 2 contains the results of the more inclusive tests. Again, these results are weighted from 1 to 5, with 5 being excellent and 1 being poor. The parts where “n/a” is used because the installation process was too difficult.
Table 2: Second Round Results

<table>
<thead>
<tr>
<th>ESDT</th>
<th>Usability</th>
<th>Installation Process</th>
<th>Applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algernon-J</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DEC Reasoner</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Drools</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>E2glite</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Jeops</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>OpenCyc</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Prolog</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Ruleby</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>SnePS</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Soar</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Verbose Evaluation

This section is devoted to the verbose evaluation of each of the ESTDs that we analyzed in the initial dig. It is extremely interesting to see the history, features, and motivation for the development of each solution, so here they are in block form. Most entries also have an URL for further reading.

Algernon / Algernon-J

http://algernon-j.sourceforge.net/download/

Algernon was developed at the University of Texas in the early 1990s. It interfaces with Lisp, so it has a similar syntax. It is also very similar to Protégé, and was even incorporated into it as a plug-in later in its lifetime. The original system appeared to stop active development in the late 1990's. It was ported to Java around 2005 for further development, when many other applications were also getting ported to Java.

Babylon

Babylon was implemented and embedded in Common Lisp. There is a published book from 1992 titled, Babylon: An Open and Portable Development Environment for Expert Systems.
Unfortunately, the book is no longer available from the German National Research Center for Computer Sciences, however Carnegie Mellon University seems to still maintain a copy. Other than this book, there appears to be very little documentation available on the Internet, and all research with this platform seems to have ended in the mid 1990s.

**Discrete Event Calculus Reasoner**
http://decreasoner.sourceforge.net/

The Discrete Event Calculus Reasoner performs automated commonsense reasoning, but it is meant to be used in a timed linear system. All rules use predicate logic and are specified in lambda type syntax. It is requires Python to operate. The documentation is extensively available with tutorials, and there is even a book available called, *Commonsense Reasoning*. This platform requires Python to operate. Although this is a very interesting development tool, the “commonsense” framework was not able to fit into our needs.

**Drools**
http://jboss.org/drools/

Drools is a business rule management system written in Java. It is open source and has an amazing GUI component. It's syntax is similar to Jess or CLIPS, but then it can also have Java code integrated for easily printing things, or objects for easy memory management. It has some useful graphical tools for: rule generation, setting salience flow, and decision tables. Then the whole expert system shell, is a plugin to eclipse, and it is exportable to a Jboss server Application Server (Tomcat). It has an attractive rule engine, and its forward chaining is simple. There is current work happening on Drools, and it has an attractive interface.

**Euler Proof Mechanism**
http://www.agfa.com/w3c/euler/

The Euler proof mechanism is implemented in Java, C#, Python, Javascript, and Prolog. It supports logic based proofs, and it is still maintained and supported. It also has some attractive
interfacing with SQL.

**Expect**

Expect has a book, *EXPECT: Intelligent support for knowledge base refinement*, but it does not really have much online documentation for running it as an ESDT. Most of the documentation found on the Internet is around using a Unix tool for automation, called expect.

**F-Logic**

F-Logic is a knowledge representation and ontology language, like OWL. It is a declarative language to describe object-oriented and frame-based languages. Further analysis of this tool determined that it will not fit our needs.

**FOCL**

http://www.ics.uci.edu/~mlearn/FOCL.html

FOCL is an interesting expert system shell developed in LISP. It has several different interfaces, and the applicability of each determines its function. For instance, the Macintosh interface allows for building rule bases, but the Windows version is only useful for machine learning. This ESDT has been out of development since the mid 1990s.

**G2**

G2 is a real-time business rules engine, but it is not free. This means that no matter how good it might be, it would never be applicable to the Knowledge-Based Systems class.

**Hank**

Hank is a cognitive modeling environment for non-programmers. Production rules take the form of fact card and instruction cards. It was last updated in 2003, and it was designed for use with undergraduate cognitive psychology course at a university. Like most research projects, there is not
much commonly available documentation on this system.

**InfoSapient**
http://info-sapient.sourceforge.net/

InfoSapient is a rules engine for Java. It was last updated in 2002, and currently has support for fuzzy logic. There are some JavaDocs detailing its usage, but not much else.

**E2glite**
http://www.expertise2go.com/webesie/e2gdoc/

E2glite a knowledge base implemented in Java. It utilizes a text interface for laying out the flow of web page JavaScript forms. It a cool tool for developing an on-line tech support center, but not for this class.

**JADEX**
http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/tutorial/index.single.html

This tool seems very powerful for the development of intelligent agents, rather than knowledge based systems. There is very little documentation, little more than a simple tutorial exists for it, and it doesn't seem as simple as Jess or CLIPS.

**JAM**
http://www.marcush.net/IRS/Jam/Jam-man-01Nov01-draft.htm

JAM is also more focused around intelligent agents and learning based agents. It has less emphasis placed on the knowledge based system portion.

**JTP**
http://ksl.stanford.edu/software/JTP/

This is a knowledge-based platform put out by Stanford. There is a distinct lack of documentation though.

**Jason**
http://jason.sourceforge.net/

Again this is more an intelligent agent system designed around the Beliefs-Desires-Intentions
(BDI) paradigm, like JADEX and JAM. The verdict is that this will not work for the requirements of this class.

**Jeops**  
http://www.di.ufpe.br/~jeops/manual/  

Jeops is an exciting looking tool with an attractive interface. It also looks like it will function very nicely because it has a close integration with Java. Any Java function can be used as a rule. There are some problems with its availability as the current download site is down. Lastly, there does not exist a ton of documentation on this product.

**MIKE**  
http://kmi.open.ac.uk/people/marc/mike_text.html  

Unable to find resource

**Manderx Inference Engine**  
Unable to find resource

**OPS83 & OPS5**  
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?OPS5  

OPS5 is a rule-based programming language, but it is rather old and unsupported. It was last updated in 1995, and there exists very little documentation.

**OpenCYC**  

OpenCYC looks like a viable language, but there is currently no downloadable version. The authors proposed a new version a while ago, but there has been no visible activity for at least two years. On a related note there is a ResearchCyc, which is available with a license. This doesn't quite fit into our testing model so we are leaving it out for now, especially since we don't know exactly how useful it would be.
**Poplog**

http://www.poplog.org

Looks like it is just a general purpose programming language, not really related to Knowledge-Based Systems.

**Prolog**

Prolog is very powerful, is widely recognized, and freely available. The best implementation for our purposes is SWI-Prolog, because there are a large number of tutorials that refer to it. After basic experimentation, this is one of our final tools, because of its widespread acceptance and support. The particular implementation we should use is SWI-Prolog because it is free, fast, and is used by a number of other organizations interested in teaching ProLog. The tool is also multi-platform, allowing for easy Windows and Linux development. This programming environment is exactly what this exploration was intent upon finding.

**Protege**

http://protege.stanford.edu/

Protege is freely available via the Stanford site. They say it is used by private, commercial, and governmental groups for various knowledge ontological means. It seems like a nice interface for doing OWL, and documentation is rather good.

**PyExpect**

http://www.noah.org/wiki/Pexpect

This platform is also known as Pexpect. It is a python module based on Don Libes' Expect, and Expect is a tool for monitoring/automating other applications for either general use or testing. Although it looks fun, it is not for use in Knowledge-Based Systems.

**Ruleby**

http://ruleby.org/wiki/Ruleby
Ruleby uses a Ruby Domain Specific Language. There is some documentation, but it seems to have a limited following. Some features seem underdeveloped. Ruleby has no interface of its own and runs only in Ruby code files that contain “include Ruleby.” It uses Ruby syntax to write the rules in a class that inherits from “Rulebook.” The language supports forward chaining but not backwards chaining. The example programs are not contained in the Ruleby .gem and must be downloaded separately. Overall the language supports no exceptional features (does not even have its own interface) nor are there many resources for it on-line. We would recommend Ruleby only to a developer who is already using Ruby and wishes to add some rule based functionality to an already existing project.

**KAES**  
http://kaes.anthrosciences.net/  
Looks like an underdeveloped graduate student project.

**SnePS**  
http://www.cse.buffalo.edu/sneps/  
SNePS was created by and is currently maintained by the University of New York Buffalo. Its installation was cryptic and painful. Even though there are many journal articles that discuss it, most of this documentation is written at a very high level. SnePS has its own interface, a text console. There may also be a GUI component, but it wasn't able to be enabled during testing. There is a lot of fancy (research) documentation but very little simple straightforward “how-tos” or lists of features from a high level perspective. The rule syntax is similar to CLIPS but looks to be much more flexible. This is far from a slick beginner’s tool or a commercial tool. This is purely academic and very dense.

**Soar**  
http://sitemaker.umich.edu/soar/home  
Soar is currently available and developed, and it has a (graphical) debugger and editor. The debugger allows you to see the working memory and is nicer than the interface for CLIPS. It comes
with a lot of documentation, including very basic step-by-step tutorials. It has built in learning mechanisms but they can be turned off. The faq warns that Soar is harder to learn than Jess for simple applications. Its complex syntax makes the learning curve greater.

**TyRuBa**

http://tyruba.sourceforge.net/

Was generated as part of a Ph. D. thesis for generating Java code.

**XpertRule**

http://www.xpertrule.com

Enterprise rapid application development environment. Not publicly available, and it looks very limited.

**Tmycin**

Tmycin is not really available anymore. It only exists now as a research project at Stanford and the University of Texas.

**Prolog**

Prolog was developed as a joint venture between researchers at the University of Aix-Marseille and the University of Edinburgh in 1972. It initially gained the majority of its popularity in Europe and Britain, and was featured in the ICOT Fifth Generation Computer Systems Initiative. Prolog was designed as a theorem proving framework, revolving around Horn clauses. Much development has been made on Prolog since, and both interpreters and compilers are available.

There are many tools and tutorials available for Prolog due to its popularity and common use in rule-based systems. It was selected for the final phase because of its ease of use, free availability, active development, and large user base (resulting in good documentation).

Once we understood the fundamentals of Prolog, developing programs for it were quick and easy. Unlike previous examples of Expert Systems such as CLIPS or Jess, Prolog is a reverse-chaining
system, which, when queried, tries to work its way backwards from the query to the facts that the system has. Even with this, queries can be sufficiently generic through wildcards to allow complex conclusions.

Several free implementations of Prolog are available. The one used in most tutorials is SWI-Prolog; it's available for most operating systems and environments for free (but not GPL). There is also a GNU compiler available for Prolog that generates native code for computers. Because of the wide support for Prolog and the availability of actively developed tools, we chose to use Prolog as one of our final suggestions for this class.

We feel that AI/Expert System classes at Cal Poly would benefit from having at least a brief overview of the Prolog language, and perhaps even give the option to complete some of the programming exercises in Prolog.

SOAR

Soar was first developed at Carnegie Mellon in 1987. Soar has been in active development ever since. It's stated purpose is to be a system that embodies all the features needed of an intelligent agent. This includes several types of learning and reasoning. The heart of Soar is a rule based system that also has the ability to learn, however the latest release version Soar 9.0 has added a lot of additional functionality on top of that.

Soar comes with an extensive tutorial and nice debugging and programming environments. It was selected as one of our final systems to investigate because of its: ability to use rules (applicable to class); nice user interfaces (good for use in class); its active development (current tool with support); the presence of documentation (good for use in class).

However during our final phase when tried to make a family tree knowledge base the problems with soar in a teaching environment became clear. There is no documentation on how to use Soar as an expert system. While the tutorial does contain some information on creating rules it quickly moves on
to other things. The tutorial also explains how to write rules as pairs of propose and operator rules. This allows the right hand side of a rule to be extracted so that multiple rules do not have duplicate right hand sides. This system is built into the Soar interface and design methodology. The tutorials are all designed with this system in mind, and thus it would be difficult to write an expert system without following it. This complicates teaching it, as it would be one more thing for the students to learn. There is also no resources from any other school that has tried to teach Soar.

But much more importantly, we could not find anyway to communicate with a Soar program through a simple console interface. While Soar programs can be run in the debugger, there is no way to interact with them directly, such as prompting the user for information. Soar programs often interconnect with other programs, but this is done through xml and message passing.

It was these two issues that made us recommend that Soar not be used for the expert systems class.

**Teaching Assistance**

Once finished narrowing down which ESDTs we really wanted to look into, we tried making a family tree knowledge base in each. This effect succeeded for Prolog but not for Soar. This will serve as an entry level example for any further teaching of Prolog. In addition to this we also created presentations for both SOAR and Prolog. These presentations are to orientate students with the development environment and to show some of the history and motivation behind each solution.

**Conclusion**

Through this project our team was able to come up with several important conclusions and observations. One of the first of these is that although there are a great number of expert systems development tools are available there are only a handful of quality. Many exist only as obscure and forgotten research projects.
Our next observation was that although some languages had more features or better interfaces the was those systems that had the most documentation and user support were usually the best. More over we got a chance to observe the wide range of tools for the perspective of students. Some systems, such as Soar may be very powerful, but they are not appropriate for most classroom experiences due to their complex nature. Other systems such as Prolog are more simple to use and ultimately make better teaching tools.

Finally we conclude that Prolog is an excellent candidate for instruction and we recommend that it at least be introduced in the csc/cpe 481 because of its widespread use and importance in the expert systems community. We also suggest that students be introduced to Prolog early so that they have the option of completing homework assignments in it.

We foresee this document to be used in the future by students in csc/cpe 481 and other AI classes that are looking to find information on expert systems. We have also provided power point presentations on Prolog and Soar that can be used to introduce these topics to a class. Finally we produced a family knowledge base in Prolog which can be used by students as an example if Prolog becomes one of the expert systems taught in class.