Overview

Our additional feature is the further limiting of our list to those expert systems most promising. The bugs might be those systems which were denoted as 'YES' before, but are now eliminated.

We have identified the main topics to cover in our final documentation. These are slides for alternative expert systems focusing heavily on our final choice, to be listed in the week 8 documentation. We will provide copies of examples and tutorials with references.

This version is currently on track and in line with requirements as documented in the proposal.

User Feedback

At this point, no user feedback is available. The andriod team should have done an evaluation by now, but they haven't. Prof. Kurfess said they had some problems, but they did not offer any feedback whatsoever. We did, however, discuss some of our goals and ideas with Prof. Kurfess, and he still seemed excited about the project. His main concerns were: we had deliverables, we provided a good evaluation criteria for our evaluation team, we didn't evaluate just on pertinence to this class, and we were actually working hard and enjoying the project.

We are still discussing our findings with our peers, as they are in the same position as the eventual students receiving this material, so they will hopefully be well equipped to understand the concepts presented.

System Design

We have dug deeper in our selected Expert System Shells for whether it will satisfy our evaluation criteria. We did so by downloading the Expert System Shells, installing them, and running through some examples. The following lists our findings.

Algernon / Algernon-J

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

This is basically a LISP shell that interacts with the Algernon inference engine via 'tell' and 'ask' commands of patterns and facts. I was thinking of staying away from LISP type engines because of the complex syntax structure that it requires, but if we do go forward with a LISP shell, then we will need to select only one of the few we are evaluating. The installation requires some packages (protege) that we decided during the last milestone not to pursue.

Pursue: NO

Discrete Event Calculus Reasoner
This calculus solver is meant to be in a timed linear system. Events fire in order, and they are done. All rules use predicate logic and are specified in lambda type syntax. The learning curve on this would be high, and not really needed for this class. It's fact memory is rather limited as well.

**Pursue: NO**

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

This tool 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). This might be very nice given most students tried to create few complex rules in CLIPS/Jess in Assignment 1 instead of many simple rules as most did in Assignment 2. This might be too limited for the need of generating new fact fields on the fly, but it is nice for when the complete knowledge structure is known.

**Pursue: YES**

**E2glite**
I don't have Java plugin for my browser and platform. Working on that now, but this kinda prevents me from accessing the tool. The site seemed interesting enough, with a very helpful tutorial, and a nice rule description language. Unfortunately the rule description language isn't as powerful as Jess or CLIPS. The generic UI is nicer, though, when it works in the client's web browser. Working with this tool in 481 would be inappropriate, I feel, but in a simpler class, like if 101 was Knowledge-based, this would provide a helpful starting point for students.

**Pursue: NO**

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

This tool looks really cool and useful, and has close integration with Java. Any Java function can be used as a rule. I'd like to download it and give it a shot, but the website is down. If I get some time over the next couple weeks, I'll play with it. Unfortunately, no decision can be reached at this time

**Pursue: NO**

**OpenCyc**
This looked like it was a great tool at one point, but unfortunately, it is out of date. They don't provide any version for download at all, leaving behind an unmaintained website and software repository with promises of a “new improved version.” The last time the website was modified was 2 years ago. I am of the opinion that if this tool were available, it’d be the best option out of all of these, but unfortunately, it’s not, so it's not the best choice here. Of potential interest is ResearchCyc, which is available with a license, so perhaps it would be best to go to the department with this information, if no other expert system tool proves viable. Also, I get the feeling that the main focus of Cyc is that it has predefined rules and facts for general purpose, and not so much the expert system engine.

**Pursue: NO**

**Prolog**
Prolog looks very powerful, it's well 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 we need.

Pursue: YES

Ruleby
http://ruleby.org/wiki/Ruleby
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 online. I 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.

Pursue: NO

SnePS
http://www.cse.buffalo.edu/sneps/
SNePS was created and is maintained by the University of New York Buffalo. The installation was cryptic and painful. Most of the documentation is written at a very high level and there are many journal articles that discuss it. There is one nice tutorial though for beginners
http://www.cse.buffalo.edu/sneps/Tutorial/tutorialHTML.html. It has its own interface, a text console. It appears not to interface with the LISP programs. It appears to be very powerful. There may be a GUI component, but I was unable to get it to work. There are a lot of “it appears” in this review because 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.

Pursue: NO

Soar
http://sitemaker.umich.edu/soar/home
Soar 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. The Soar community is active and Soar is in use in multiple systems. I feel that Soar is an acceptable candidate to compare with our current tools because it has a nice interface and a powerful system. What is left to evaluate is if it has all the features that are needed in a 481 class and how easy the syntax can be learned.

Pursue: YES

Prototype and Implementation
Our additional feature is the further limiting of our list to those expert systems most promising. The bugs might be those systems which were denoted as 'YES' before, but are now eliminated.

We have identified the main topics to cover in our final documentation. These are slides for alternative
expert systems focusing heavily on our final choice, to be listed in the week 8 documentation. We will provide copies of examples and tutorials with references.

**Evaluation Issues**
Verbally, through Professor Kurfess, we learned the evaluation team needs more specific instructions on being able to evaluate our work. Our source directory now contains at least a document for each of the Expert System Shells tested containing installation requirements, installation instructions, and examples of code. It would be good for the Evaluation team to look at these examples of code and let us know if we are along the correct course for being successful in this project.