Expert System for Error Analysis

Rule Based Reasoning Applied on Log Information and Dump Reports from Mobile Phones

Daniel Gustavsson & Daniel Molin

June 14, 2010

Introduction

This short article reflects the work of developing a rule-based expert system for ST-Ericsson in Lund, Sweden.

During the development of mobile platforms, it is important to continuously test the product to ensure functionality and quality. For this purpose ST-Ericsson has developed an automatic test application called JATT (Java Automated Test Tool). It outputs different files, for example logs and reports, which have to be manually analyzed if a test failed. Depending on the problem this can be quiet hard and sometimes require that an expert gets involved. To help the user find and analyze the errors automatically, a rule-based expert system has been developed. We call it ADLA, which stands for Automatic Dump and Log Analyzer.

Rule engine

The initial plan, to write the entire expert system from scratch, was quickly discarded. It would probably have consumed most of the project time and thereby also affected the number of features and competence of the final system. Because of this, it was instead decided to use a third-party rule engine, which is the reasoning part of an expert system. The reasoning is performed by combining simple rules and facts. The rules describe what condition must be met for its consequence to be executed. This way a complex result can be achieved through several small steps.

Drools[1] is an open-source rule engine developed in Java by JBoss Community, which is run by Red Hat. It was adapted to fit our requirements and now plays a significant role in the reasoning part of our expert system, ADLA.

Problems

Here are some problems we faced during the development; the first one was how to identify the result. When the rule engine is done reasoning, it is time to decide what facts should be in the result presented to the user. The problem is to separate the interesting facts from the other facts used when reasoning.

Another problem was how to manage without a user in the automatic system. The system can not ask the user questions, e.g. when errors occur, without interrupting the entire system. The problem gets even harder since we want to avoid the use of log files for presenting errors to the user. This is because the purpose of the system is to analyze files so the user does not have to.

[1] http://www.jboss.org/drools/drools-expert.html (verified 2010-06-09)

Result

ADLA is separated into a reasoning part and an interactive part. This way the result can be viewed without running the reasoning part every time.

We designed the interactive part based on the JUNG[2] framework, where the result from the reasoning and the paths taken to reach it are displayed as a graph, see figure 1. This way the users themselves can identify the interesting parts and also get the reasoning behind them. The graph was also the solution for presenting errors to the user. Each error found is connected to the rule where it originated.

To aid the users in identifying the result a number of features have been added. The nodes can be right-clicked to bring up additional information about them. Below the graph the different node types are explained. The area is dynamically updated to always show the node types currently used in the graph, helping the user to interpret it.

To avoid having too many nodes cluttering the graph, making it harder to overview for the user, only the important nodes are shown by default. With the checkbox "Show all nodes", the user has the ability to toggle all nodes on. This is useful to find out if some specific fact was present in ADLA, even though it was not important.

Conclusion

ADLA makes it easier for the users to identify errors from the automatic tests run by JATT. This also decrease the workload on the experts and makes knowledge more available to all co-workers.

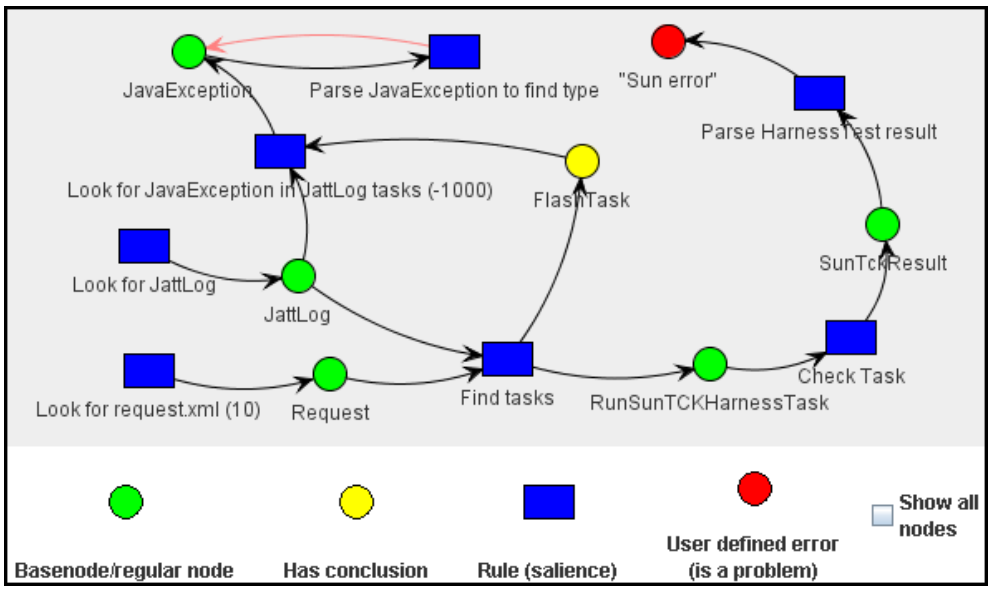

Figure 1: The graphical part of ADLA. All nodes in the graph, facts and rules, are connected to each other with arrows to visualize the flow in the system. An arrow pointing from a rule means it created the fact, if the arrow is red it was updated instead. An arrow pointing to a rule means it required the fact to run.

[2]Java Universal Network/Graph, http://jung.sourceforge.net (verified 2010-06-11)