Chapter XIII
The Debugging of Rule Bases

Valentin Zacharias
Forschungszentrum Informatik (FZI), Germany

ABSTRACT

This chapter presents an overview of the issues affecting and the tools used for the debugging of rule bases. It describes the challenges in debugging rules, presents a classification of the debugging methods developed in academia and the tools currently used in practice. This chapter explains the main debugging paradigms for rule based systems: Procedural Debugging, Explanations, Why-Not Explanations, Algorithmic Debugging, Explorative Debugging, Automatic Theory Revision and Automatic Knowledge Refinement.

INTRODUCTION

The creation of rule bases, like the creation of any software system, is an error prone process. In order for a rule base to function satisfactory, (most of) the errors have to be found and need to be corrected. The most common way to find errors is by testing the rule base, discovering bugs in the way these tests are processed and then to find and correct the faults causing them - debugging. The debugging process is an important part of any non-trivial manual rule base development, indeed, a recent survey of 50 developers of rule based systems found that the difficulty of debugging was the most important issue hindering a rule base's development.

This big role and difficulty of debugging stands in marked contrast to the often professed believe that rule languages are a simpler and more natural way to build computer systems - implying a lesser importance and difficulty of debugging. This idea of rule languages as a simpler way to program rests on three observations:

- Rule languages are (mostly) declarative languages that free the developer from worrying about how something is computed. This should decrease the complexity for the developer.
- The If-Then structure of rules resembles the way humans naturally communicate a large part of knowledge.
- The basic structure of a rule is very simple and easy to understand.

However, in particular in relation to debugging, this promise of simplicity remains elusive. The same survey mentioned above found that a large majority of developers of rule based systems felt that it was easier to debug a conventional (object oriented or procedural) program than a rule base.

In a study of three rule base development processes (Zacharias 2008b) we identified six reasons for this apparent mismatch between the promise of simplicity and the difficulty of rule base development; reasons that explain why the simple and well understood rules are nevertheless hard to develop and to debug.

- **The One Rule Fallacy:** Because one rule is relatively simple and because the interaction between rules is handled automatically by the inference engine, it is often assumed that a rule base as a whole is automatically simple to create. However, the inference engine obviously combines the rules only based on how the user has specified the rules; and it is here - in the creation of rules in a way that they can work together - that most errors get made. Examples for errors affecting the interaction of rules are the use of different attributes to represent the same thing or two rules being based on incompatible notions of a concept. Hence a part of the gap between the expected simplicity of rule base creation and the reality can be explained by naive assumptions about rule base creation. Rule based systems hold the promise to allow the automatic recombination of rules to tackle

Figure 1. Issues Hindering the Development of Rule Based Systems - based on the answers of 76 developers of rule based systems. The participants of an online survey were asked to classify each of the issues whether it was 'Not an Issue', an 'Annoyance' or whether it 'Hindered Development'. The votes for 'Not an Issue ' were multiplied by zero, those for 'Annoyance' by one and those for 'Hindered Development' by two, the sum of all votes for an issue then determined its ranking in this list (Zacharias 2008a).