UNIVERSITEIT VAN AMSTERDAM

MASTERS PROJECT

---

# Representation Mismatch Reduction for Development in Rules-Based Business Engines

---

*Author:*
Paul SPENCER

*Supervisor:*
Dr. Clemens GRELCK

*A thesis submitted in fulfillment of the requirements for the degree of Master of Software Engineering*

*in the*

Graduate School of Informatics
Faculty of Science

July 20, 2021

UNIVERSITY OF AMSTERDAM

# Declaration of Authorship

I, Paul SPENCER, declare that this thesis titled, "Representation Mismatch Reduction for Development in Rules-Based Business Engines" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my work.
- I have acknowledged all of the main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSITEIT VAN AMSTERDAM

# *Abstract*

Graduate School of Informatics

Faculty of Science

Master of Software Engineering

**Representation Mismatch Reduction for Development in Rules-Based Business Engines**

by Paul SPENCER

*Context*: Declarative rules engine languages, such as Drools, can become difficult to reason about when there are many rules.

*Objective*: This project investigates how different projections of the code can ease the comprehensibility of the code.

*Method*: We created an implementation of the Drools language using the MPS language workbench and made innovative projections of large ASTs.

*Results*:

*Keywords*: projectional editing; Rules Engines; MPS; Drools

*Paper type*: Research paper

# *Acknowledgements*

We would like to acknowledge the School of The Gradate school of Informatics in the Faculty of Science at The University of Amsterdam for their guidance, specifically Dr. Clemens Grelck, who has been a supportive, understanding, and available academic adviser.

We received inspiration from the Strumenta Languages engineering community. Specifically we would like to thank Federico Tomasetti, who shared with me his model of a rules engine in MPS.

Also we would like to thank Václav Pech from JetBrains for the course he created and the time he spent with me explaining MPS. Further, Sergej Koščejev from JetBrains helped us with specific MPS issues during his Office hours.

Other prolific output that terrifically helped our research and development was the Heavy Meta YouTube series from Kolja Dummann and the dozens of papers and books from Markus Voelter, both currently working at Itemis A.G.

[TODO: add mark proctor if he helps out]

Our greatest thanks go out to Toine Khonraad, an alum of this course, who provided me with moral and monetary support, as well as wisdom and friendship that aided in the completion of this, my fourth attempt at getting this project behind me. Without his constant mantra of simplify, simplify, simplify, we would still be implementing the Drools languages now without having made a single projection.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

> The limits of my language mean the limits of my world.
>
> *Logico-Tractatus Philosophicus*
> *Ludwig Wittgenstein*

## 1.1 Problem statement

Miller's Law[1] states that an average human can hold in his short-term memory 5-9 objects. This is often an argument for more succinct code. The argument being anything that is not immediately in the developers vision has to be stored in her memory. With it being impractical to reason about code that she cannot recall, then the fewer relevant items to her reasoning that are out of view the easier it is to reason about the code.

[TODO: complete the problem statement]

## 1.2 Research questions

To reason about a large code base of rules engine code effectively, a different presentation is needed. This presentation should allow a clearer organization whilst remaining interactive. We can formulate the following research questions based on the discussion in the preceding sections.

The research question we wish to answer is:

- **Main research question:** "How can projectional editors and DSLs be combined to address feedback mechanisms for developers in the context of reasoning about rules in a rule-based business engine?"

This question requires knowing if it is possible with current tooling, thus we would like to answer the question:

- **RQ 1:** "What is the current state of language workbenches supporting projectional editing?"

Finally, we specifically would like to know how we can improve the ability to reason about the business rules engine, so we ask the question:

- **RQ 2:** "Which projections can help developers to get appropriate feedback about rules?"

## 1.3 Contributions

This thesis proposes a code representation of business rules in a concise and readable format that could solve comprehensibility issues resulting from large code bases of business rules. The implementation behind the approach relies on language engineering and projectional editing. An implementation has been developed as a stand alone opensource solution on a limited demonstration version of Drools. The underlying Drools implementation can be used as a base language for model to model generation by the wider MPS ecosystem.

## 1.4 Project context

This investigation was hosted by Khonraad Software Engineering, a subsidiary of Visma. Khonraad provides mission-critical services focussed on the automation of workflows at the cross-section of local government and healthcare. Specifically, Khonraad facilitates the mental health care and coercion laws in the Netherlands - WVGGZ, WZD, and WTH - which provide agencies the ability to intervene in domestic violence, psychiatric disorders, and illnesses.

Khonraad's system facilitates reporting and communication between municipalities, police, judiciary, lawyers, mental health care, and many social care institutions. The system has 15,000 users and is available 24/7.

Configuration and administration use complex matrices of compliance mechanisms, access user rights and communication settings. The sensitivity of the personal data, being both medical and criminal, means security is of utmost importance. The security against data loss, preventing unlawful disclosure and guaranteeing availability, especially during crisis situations, is crucial. Demonstration of the correctness of the, often changing, configuration is a major concern in the company.

This work environment allows us to work on an existing project, where the tangible success will have an impact on the lives of those in critical need. Khonraad has it's own implementations in the Drools language, that have evolved over the iterations of the laws. The evolution of the code base over the years means that the real-life issues we came across are not just thought experiments.

## 1.5 Thesis outline

We start in chapter 2 with the required background information on projectional editing and rules engines. In chapter 3 we present the research questions. Further, the chapter

describes the protocol that we use for search strategy, selecting our studies, extracting data from them, and synthesizing the results. Chapter 4 presents the results of our synthesis of data from the primary studies. This is followed, in chapter 5, by a discussion of both the validity of the work and the implications of the findings. We discuss the implications of this study in chapter 6. Finally, the conclusions are presented in chapter 7.

# Chapter 2

# Background

This chapter gives the background information required on rules engines and projectional editing. It presents the specific case of rules engine that we will be using for our investigation: Drools. Further, it briefly examines the base tool type for creating Domain-specific languages: Language work benches. Finally, it presents the specific projectional editing tool we will be using: JetBrains MPS.

## 2.1 RulesEngines

### 2.1.1 What is a rules engine?

In this section we will describe what a rules engine is and a little of its history.

The Aristotelian doctrine of essentialism declares that a thing has properties that are essential and properties that are accidental. If one takes away accidental properties, then the thing remains the thing. If one takes away essential properties, the thing is no longer the thing. If the thing is a business application, then its essential properties are its business rules.

Simply put, business rules are the principles or regulations by which an organization carries out the tasks needed to achieve their goals. When properly defined these rules can be encoded into statements that defines or constrains some aspect of the business organizational behaviour. A rule consists of a condition and an action. When the condition is satisfied then the action is performed. More formally, business rules can be seen as the implication in the basic logical principle of Modus Ponens.

When described like this, one could me forgiven for thinking is this not just an if-then logic that is frequently used in traditional programming. One would not be wrong, however in traditional programming, representing all the combinatorial outcomes can become complex. In the typical application architecture, rules are distributed in the source code or database. Each additional rule leads to more fragility.

Documentation describing these rules may be found in the design documentation or user manuals. However, as applications evolve documentation gets out of sync with codebase. Once this desynchronization occurs, to know what the rules that govern the application, one has to navigate the codebase and decode the rules from their, often scattered, locations.

A rules engine is also known as a Business Rules Engine, a Business Rules Management System or a Production Rules System. The goal of a rules engine is the abstraction of business rules into encoded and packaged logic that defines the tasks of an organization with the accompanying tools that evaluate and execute these rules. Rules engines match rules against facts and infer conclusions. If we return to the Modus Ponens comparison:

$$\begin{array}{l} p \\ \underline{p \rightarrow q} \\ \therefore q \end{array}$$

If the premise $p$ holds. And the implication $p \rightarrow q$ holds then the conclusion $q$ holds. In terms of a rule engine and business rules this could be seen as:

1. the rules engine gathers the data for the premise: $p$
2. it examines the business rules as the implications: $p \rightarrow q$
3. it executes the conclusion: $q$

Rules Engines are declarative, focussing on the what of the rules not the the how of the execution. Date[2] describes rules engine as to "specify business process declaratively, via business rules and get the system to compile those rules in to the necessary procedural (and executable) code." Fowler[3] describes rules engine as follows: " ... providing an alternative computational model. Instead of the usual imperative model, which consists of commands in sequence with conditionals and loops, a rules engine is based on a Production Rule System. This is a set of production rules, each of which has a condition and an action ...".

Rule engines arose from the expert systems of the late 70s and early 80s. Expert systems initially had three main techniques for knowledge representation: Rules, frames and logic[4]. "The granddaddy" of the expert systems, MYCIN, relied heavily on rules based knowledge representation[5], rather than long inference chains. MYCIN was used to identify bacteria and recommend antibiotic prescriptions. MYCIN and its progenitor, DENDRAL, spawned a whole family of Clinical Decision Support Systems that pushed the rules engine technology until the early 1980's. Research into rules engines died out in the 1980s as it fell out of fashion.

Early in their existence, the rules engines hit a limiting factor because the matching algorithms they used suffered from the utility problem, i.e. the match cost increased linearly with the number of rules being examined/ This problem was solved by Charles Forgy's efficient pattern matching Rete algorithm[6], and its successors. This algorithm works by modelling the rules as a network of nodes where each node type works as a filter. A fact will be filtered through this network. The pre-calculation of this network is what provides the performance characteristics.

In general, rules engines are forward chaining. This means to test if [TODO: Explain forward chaining with logic symbols]

[TODO: ADD MORE HISTORY HERE]

Moving forward to current times, there are a few rules engines currently in use. Some of the more commonly used ones are shown in table 2.1

Some of the advantages of using a rules engine include:

| Product | | Developer | licence type |
|---|---|---|---|
| CLIPS | [7] | NASA | open source |
| Drools | [8] | JBoss/RedHat | open source |
| BizTalk Business Rule Engine | [9] | Microsoft | proprietary |
| WebSphere ILOG JRules | [10] | IBM | proprietary |
| OpenRules | [11] | OpenRules | open source |

TABLE 2.1: Rules Engine products.

- The separation of knowledge from it's implementation logic
- Business logic can be externalized
- Rules can be human readable

In summary a rules engine, is the executor of a rules based program, consisting of discreet declarative rules which model a part of the business domain.

### 2.1.2 What is Drools?

JBoss Rules, or as it is more commonly known, Drools, is the leading opensource rules engine written in Java. In this paper when we use the name "Drool" we are referring to the "Drools Expert" which is the rule engine module of the Drools Suite. Drools started in 2001, but rose to prominence with it's 2005 2.0 release. It is an advanced inference engine using an enhanced version of the Rete algorithm, called ReteOO[12], adapted to an object-oriented interface specifically for Java. Designed to accept pluggable language implementations, it can also work with Python and .Net. It is considered one of the most developed and supported rules platforms.

For rules to be executed there are 4 major components as demonstrated in figure 2.1. The production memory contains the rules. The rules are the focus of this thesis and therefore we will delve into much more detail later on these. The working memory contains the facts. The pattern matcher, using the aforementioned ReteOO algorithm will determine the possible rules to fire. Rules that match will be placed on the agenda to be executed. A conflict resolution strategy will decide which rule will fire. If a rule specifies to halt or there are no matching rules left on the agenda, then the program extends.

#### An explanatory example

An example of a DRL file can be seen in listing 2.1.

```
1   package org.drools.examples.honestpolitician
2
3   import org.drools.examples.honestpolitician.Politician;
4   import org.drools.examples.honest politician.Hope;
5
6   rule "We have an honest Politician"
7       salience 10
8       when
9           exists( Politician( honest == true ) )
10      then
```

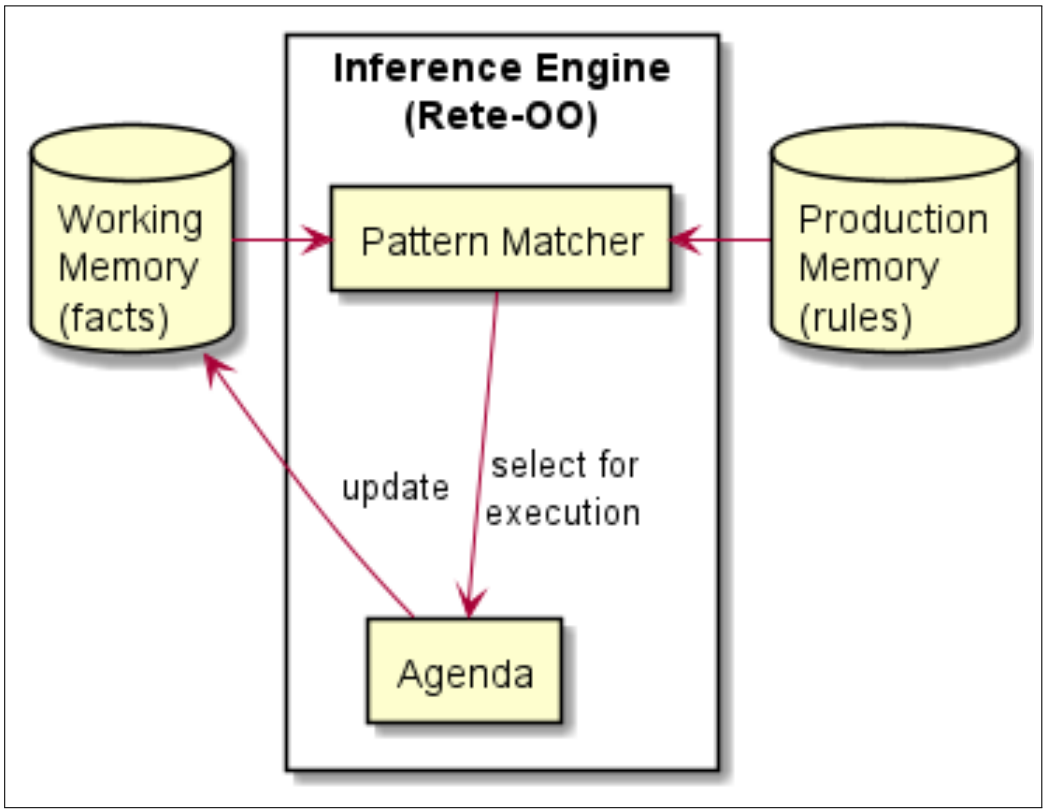


FIGURE 2.1: Drools components.

```
11              insertLogical( new Hope() );
12      end
13
14      rule "Hope Lives"
15          salience 10
16          when
17              exists( Hope() )
18          then
19              System.out.println("Hurrah!!! Democracy Lives");
20      end
21
22      rule "Hope is Dead"
23          when
24              not( Hope() )
25          then
26              System.out.println( "We are all Doomed!!! Democracy is Dead
                    " );
27      end
28
29      rule "Corrupt the Honest"
30          when
31              $p : Politician( honest == true )
32              exists( Hope() )
33          then
34              System.out.println( "I'm an evil corporation and I have
                    corrupted " + $p.getName() );
35              modify( $p ) {
36                  setHonest( false )
37              }
38      end
```

LISTING 2.1: Example Drools file.

Listing 2.1 gives the Drools engine instructions on what actions to take when something changes in the working memory. What this toy example does is reacts to when an honest politician is added to the working memory, prints a message celebrating the existence of said politician, corrupts her, gloats in a message and then prints a message of despair. The code in listing 2.1 does the following:

1. on line 1 the package statement identifies the rule file
2. on lines 3 and 4 the import statements describes which facts can be used
3. the "We have an honest Politician" rule on line 6 does the following:
   (a) using salience on line 7 it sets that this rule is to be run before rules with a lower salience
   (b) on line 10 it checks the working memory for Politician facts with the honest property equal to true
   (c) on line 12, if found then Hope facts will be inserted into the working memory
4. the "Hope Lives" rule on line 15 does the following:
   (a) line 18 check if any Hope facts exist
   (b) on line 20, if found, it prints a message
5. the "Hope is Dead" rule on line 23 does the following:
   (a) checks if no Hope facts exist on line 25
   (b) if none are found, on line 27, it prints a message
6. the "Corrupt the Honest" rule on line 30 does the following:
   (a) line 32 checks for any Politician facts with the honest property equal to true, and sets them to the variable $p
   (b) line 33 checks if any Hope facts exist
   (c) if both hope and politicians are found on line 35 it prints a message including the $p variables name
   (d) on line 36 to 38 it modifies the fact in working memory represented by $p to change it's honest property

## 2.2 Projectional Editing

### 2.2.1 What is projectional editing?

### 2.2.2 what are Language Workbenches?

### 2.2.3 What is MPS?

## Chapter 3

# Method

# Chapter 4

# Results

the purpose of abstraction is not to be vague but to create a new semantic level in which one can be absolutly precise.

*Logico-Tractatus Philosophicus*
*Edsger W. Dijkstra*

# Chapter 5

# Discussion

## 5.1 Threats to Validity

### 5.1.1 Construct Validity

### 5.1.2 Internal Validity

### 5.1.3 External Validity

### 5.1.4 Reliability

### 5.1.5 Repeatability vs Reproducibility

### 5.1.6 Method improvement

# Chapter 6

# Implications to research and practice

## 6.1 Implications to research

## 6.2 Future research directions

## 6.3 Implications to practice

## Chapter 7

# Conclusion

## Appendix A

# InterviewTranscripts

Write your appendix content here.

# Bibliography


[1] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information.," *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[2] C. J. Date, *What not how: the business rules approach to application development*. Addison-Wesley Professional, 2000.

[3] M. Fowler, *Should i use a rules engine?* https://martinfowler.com/bliki/RulesEngine.html, Accessed: 2021-07-18, 2009.

[4] P. Jackson, "Introduction to expert systems," 1986.

[5] E. H. Shortliffe, "Mycin: A rule-based computer program for advising physicians regarding antimicrobial therapy selection.," STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE, Tech. Rep., 1974.

[6] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," in *Readings in Artificial Intelligence and Databases*, Elsevier, 1989, pp. 547–559.

[7] *CLIPS product page*, http://www.clipsrules.net/, Accessed: 2021-07-17.

[8] *Drools product page*, https://www.drools.org/, Accessed: 2021-07-17.

[9] *BizTalk product page*, https://docs.microsoft.com/en-gb/biztalk/, Accessed: 2021-07-17.

[10] *IBM WebSphere JRules product page*, https://www.ibm.com/docs/en/iis/11.7?topic=applications-websphere-ilog-jrules, Accessed: 2021-07-17.

[11] *OpenRules product page*, https://openrules.com/, Accessed: 2021-07-17.

[12] D. Sottara, P. Mello, and M. Proctor, "A configurable rete-oo engine for reasoning with different types of imperfect information," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 11, pp. 1535–1548, 2010.