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

August 12, 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

Declaration of Authorship iii

Abstract v

Acknowledgements vii

1 Introduction 1

1.1 Problem statement . . . 1

1.2 Research questions . . . 2

1.3 Contributions . . . 3

1.4 Project context . . . 3

1.5 Thesis outline . . . 3

2 Background 5

2.1 RulesEngines . . . 5

2.1.1 What is a rules engine? . . . 5

2.1.2 What is Drools? . . . 7

An explanatory example . . . 10

2.2 Projectional Editing . . . 11

2.2.1 What is projectional editing? . . . 11

parser based editing . . . 11

projectional definition . . . 12

How projectional editing works . . . 12

2.2.2 Examples of projectional editing . . . 14

What advantages does projectional editing bring? . . . 16

What are the disadvantages of projectional editing? . . . 19

2.2.3 What are Language Workbenches? . . . 21

2.2.4 What is MPS? . . . 21

3 Method: Systematic Review **23**

4 Method: Survey **25**

4.1 Questionaire Design . . . 25

4.2 Participants . . . 26

4.3 Validity . . . 26

4.4 Pre-test . . . 26

4.5 Sampling . . . 26

4.6 Procedure . . . 27

5 Results **29**

6 Discussion **31**

6.1 Threats to Validity . . . 31

6.1.1 Construct Validity . . . 31

6.1.2 Internal Validity . . . 31

6.1.3 External Validity . . . 31

6.1.4 Reliability . . . 31

6.1.5 Repeatability vs Reproducibility . . . 31

6.1.6 Method improvement . . . 31

7 Implications to research and practice **33**

7.1 Implications to research . . . 33

7.2 Future research directions . . . 33

7.3 Implications to practice . . . 33

8 Conclusion **35**

Bibliography **37**

A Protocol Validation Checklist **41**

B Systematic Literature Review Log **43**

B.1 Search Description 43

B.2 Table description 43

C InterviewTranscripts **51**

D Questionaire Text **53**

D.1 Page 1 - Introduction 53

List of Figures

2.1 Drools components. 8

2.2 Drools Inference Loop. 9

2.3 Drools Rule Breakdown. 9

2.4 Projectional editing loop. TODO: comparison with Parsing 13

4.1 persuasive introduction. 28

List of Tables

2.1 Rules Engine products. 7

2.2 An incomplete list of projectional languages 14

2.3 Papers describing advantages. 17

2.4 Papers describing projectional editing disadvantages. 19

4.1 persuasion tactics in figure 4.1 27

A.1 Protocol Validation Checklist 41

B.1 Search Engine/Library Key 44

B.2 Systematic review log 49

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.

Our host organization, Khonraad Software Engineering, a subsidiary of Visma, 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 configuration is done in a business rule system, specifically JBoss Drools.

Drools is a language that shares an unfortunate characteristic with many other rules languages. It is verbose and can contain many rules that can interact with each other without obvious visual connection. As Forgy[2] points out, for production systems in general, "production systems have another property that makes them particularly attractive

for constructing large programs: they do not require the programmer to specify in minute detail exactly how the various parts of the program will interact". This property leads to very large and hard to reason about collections of implicitly connected rules.

Reasoning over a small number of rules is already surprisingly hard. Our host organization has many rules and, thus, reasoning about them is particularly challenging.

We have observed the difficulty that developers have trying to reason about and edit collections of Drools files. We hypothesize that developers can be presented with different views on their code that will allow them to better understand the code. The problem we wish to solve - how to improve the ability to reason about large collections of Drools rules - we believe, lends itself to the technique of projectional editing. By using projections to improve feedback whilst coding, we believe that this can reduce the representation impedance mismatch that hampers developer's reasoning.

The problem considered in this thesis is how to present rules to a developer in a way in which she can interact with [TODO: finish this thought]. As is perhaps already obvious it is not our intention to override the will of the language engineers who have spent many years developing this language and it's ecosystem. The goal of this thesis is to augment the current developer experience.

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 **??** 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 5 presents the results of our synthesis of data from the primary studies. This is followed, in chapter 6, 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 7. Finally, the conclusions are presented in chapter 8.

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. Simply put, they are where we evaluate our 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 follow the recognize-act cycle. First the match, i.e. are there any rules with a true condition, Next the do conflict resolution, to pick the most relevant of the matching rules, finally they act, which is to perform the actions described in the rule. If no items are matched then the cycle is terminated, otherwise the first step is returned to.

Rules Engines are declarative, focussing on the what of the rules not the the how of the execution. Date[3] 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[4] 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[5]. "The granddaddy" of the expert systems, MYCIN, relied heavily on rules based knowledge representation[6], 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[2], 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.

The first popular rules engine was Office Production System from 1976. In 1981 OPS5 added the Rete algorithm. CLIPS in .. JESS Drools

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.

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:

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

Rules that represent policies are easily communicated and understood. Rules retain a higher level of independence than conventional programming languages. Rules separate knowledge from its implementation logic. Rules can be changed without changing source code; thus, there is no need to recompile the application's code. Cost of production and maintenance decreases.

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. This will not change during an analysis session. The rules are the focus of this thesis and therefore we will delve into much more detail later on these.

In Forgy's[2] overview of a rete algorithm, the following steps occur.

1. Match : Evaluate the LHSs of the productions to determine which are satisfied given the current contents of working memory
2. Conflict resolution : Select one production with a satisfied LHS; if no productions have satisfied LHSs, halt the interpreter
3. Act : Perform the actions in the RHS of the selected production
4. Re-evaluate : Go To 1

Figure 2.2 show more detail of how these components interact within Drools to infer a conclusion. First a fact or facts are asserted in the working memory. The working memory contains the current state of the facts. This triggers the inference engine. The pattern matcher, using the aforementioned ReteOO algorithm will determine examine the working memory and a representation of the rules from the production memory to determine which rules are true. Matching rules will be placed on the agenda. It can be the case that many rules are concurrently true for the same fact assertion. These rules are in conflict. A conflict resolution strategy will decide which rule will fire in which order from the agenda. The first rule on the agenda will fire. If the rule modifies, retracts or asserts a fact, then the inference loop begins again. If a rule specifies to halt or there are no matching rules left on the agenda, we have inferred our conclusion.

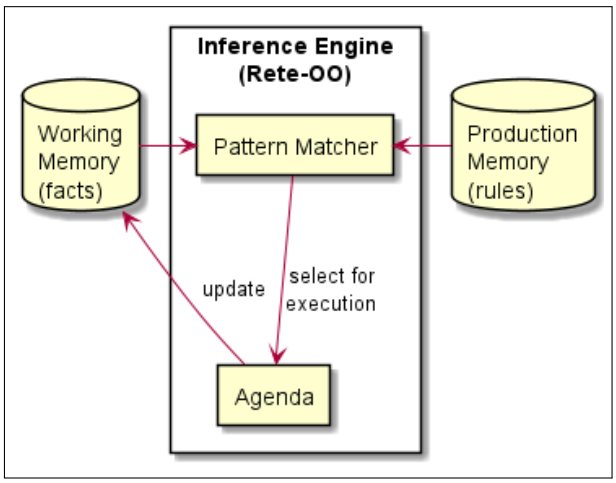

FIGURE 2.1: Drools components.

The component we will be focussing on in this paper is the rules. Rules are stored in a rules file, a text file, typically with a drl extension. During execution the rules do not change and are stored in production memory. For the sake of this paper we will skip past package, import, global, declare, function and query, which are also stored in the rule file. We will examine the anatomy of a rule.

A rule is made of 3 parts: attributes; conditions; and consequences. Attributes are an optional hints to the inference engine as to how the rule should be examined. The conditional, when, or left hand side (LHS) of the rule statement is a block of conditions that have to in aggregate return true for the asserted fact in order to be considered to be placed on the agenda. The actions, consequences, then, or Right hand side (RHS) of the rule statement contains actions to be executed, should the rule be filtered

The LHS is a predicate statement, made up of a number of patterns. variables can be bound to facts that match these patterns for use later in the LHS or for updating the working memory on the RHS. the patterns are used to evaluate against the working memory.

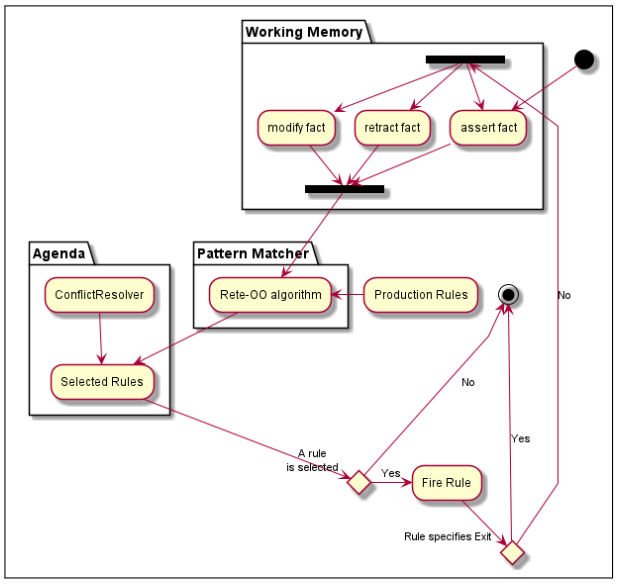

FIGURE 2.2: Drools Inference Loop.

The pattern match against the existence of facts. Patterns can also match against conditions of the properties of facts. Connectives such as not, and, and or can be applied to the patterns. The patterns apply to individual Facts rather than the group, thus can be seen as first order predicates.

There are some more advanced features in the LHS, but for this paper, these are the features we will be looking at.

Whilst the RHS can contain arbitrary code to be executed when a rule is fired, it's main purpose is to adjust the state of truth in the working memory. One can insert, modify, and retract facts in the working memory. modifying and retracting facts, must be done on fact variable references that have been created in the LHS. One can explicitly terminate the inference loop, with a halt command.

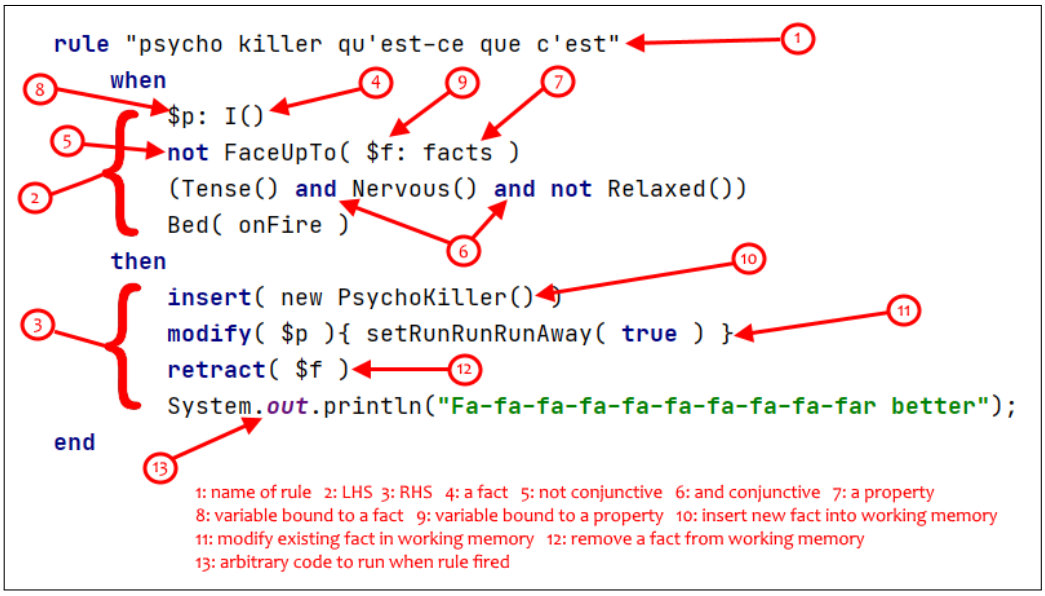

FIGURE 2.3: Drools Rule Breakdown.

An explanatory example

An example of a DRL file can be seen in listing 2.1. This has been extracted from the Drools sample code.

```
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
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?

Traditionally programmers write code with text editors, or integrated development environments (IDE), which adjust the concrete syntax and allows a parser to create the abstract syntax tree. A projectional editor, also known as structured editor or syntax-directed editor, inverts this relationship, as a developer edits the abstract syntax tree and allows the IDE to project the concrete syntax.

parser based editing

In a traditional, parser-based, development workflow, users use a text editor to represent the program. Because it is text based then the notation of the language is limited to text. A grammar is a definition of the formal syntactical rules, or concrete syntax, of a programming language. The lexer and parser are derived from the grammar. The text is passed into a text buffer where first a lexer will turn the text into tokens. A parser will validates that these tokens, the words of the language, are syntactically correct. If the tokens are not rejected then the parser will construct first a parse, or concrete syntax, tree and from there, an abstract syntax tree (AST).

An AST is a tree structure that represents the semantic meaning of the source code, stripped of all the syntactic details. The parser will do some name resolution to take care that references within the source code are represented in the tree. This turns the tree into a graph.

Compilers use the AST to do subsequent processing, such as linking, transformation, analysis, type checking, etc. Modern IDEs, in the background, also parse the code it is displaying to create an AST in order to offer relevant coding assistance. This assistance is appreciated, because without IDE help learning the concrete syntax of large languages is error prone and exploratory programming is laborious if one has to wait until compilation to discover mistakes.

projectional definition

A projectional editor does not parse any text. In its place, a developer reads and edits the AST through a projected notation. Her editing gestures immediately and directly manipulate the AST within predefined and fixed layouts.

The principle of projectional editing is familiar to those that use visual programming, like Scratch or Blockly, or graphical modelling tools, such as MetaEdit+. These tools do not parse pixels to get to their AST. They project the underlying models/programs in a view and which they store as the model/AST and not as a plain text equivalent in a traditional programming language.

Projectional editing is the generalization of this idea, with the ability to render multiple representation of the program with a wide range of notation styles.

The projection may be textual. It also may be any other notation that can represent the semantic meaning of the code, such as formulas, graphs, or images. Projections are not just the notation, but also how the user interacts with the projection. In this sense the definition of the projections and the IDE/UI overlap.

How projectional editing works

As shown in figure 2.4, a projectional editor has a model or an AST. It renders a presentation of the model as a projection. The developer performs actions on the projection. every user editing action is directly mapped on to a change in the AST.

To perform the above two things have to be defined by language engineers: The Meta-Model and the editor.

The meta-model, analogous to the abstract syntax, describes the node concepts and connection that can be used to build the, hierarchical structure that is the AST. This hierarchy can have references to nodes in other branches, so, although named a tree, it is actually a graph. The AST will be stored independently of the concrete syntax. The tree is often stored with a database, XML or a proprietary file format.

Rules of the meta-model can further be described through behaviours such as type systems or scoping rules.

Projectional editors avoid the grammars and parsers that serve as the definition of the concrete and abstract syntax in a traditional text based language. Instead of transforming the concrete to the abstract with parsing, the abstract is transformed to the concrete via a projection engine that uses projection rules. Editors are a combination of the projection rules and the gestures or actions that will create change request to the AST. They are analogous to the concrete syntax.

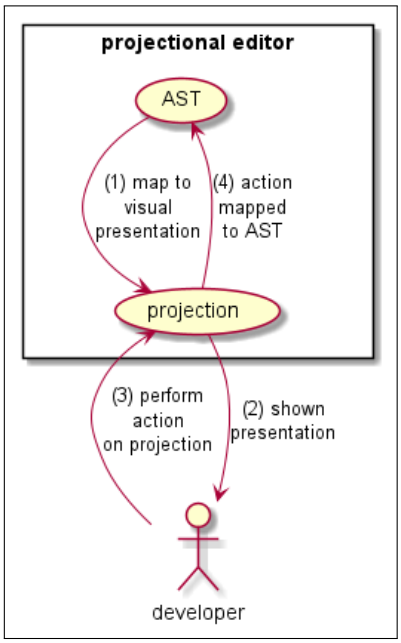

FIGURE 2.4: Projectional editing loop. TODO: comparison with Parsing

One of the actions can be typing text, however, every string is recognized as it is entered, so there is no tokenizing and the text is being entered into the templates defined in the editor. The resulting changes are displayed to the developer in a newly derived projection of the underlying AST.

The projection uses graphical elements to represent the representation. Although often appearing textual, each of the text elements are references to nodes in the AST.

Developers can only interact with the editor via the rigidly controlled code completion menus or gestures and actions.

The AST is directly built from each interaction she has with the editor. Nodes are creates as instances of the concepts defined in the meta-model. Each node has it's own unique id and points to it's defining concept. It is unambiguous. References are first class and defined by the id rather than resolved by name, as in parser based languages. Disambiguation happens at time of input, as the developer choses from limited legal inputs.

The separation of the abstract and concrete allows the language engineer to implement multiple projections of the same model, using different notations, each node of the AST taking having the design she envisions. The pattern used for projection is similar to MVC, so multiple views of the program can be visible and updateable at the same time.

Graphical modelling tools, for example for UML modelling, are specialized implementations of projectional editing. UML diagrams are not stored as pictures and whose pixels are parsed to create an AST. Instead the model is stored, often with extra information about visual layout, and the image of the UML is projected to the modeller to edit. Projectional editing generalizes this approach to projecting any notation defined by the language engineer.

2.2.2 Examples of projectional editing

Table 2.2 gives an incomplete and inconsistant history of projectional languages.

Language	Notes
MPS	inspired by a call to action for Language orientated program
	on a mission to build a product to fulfil that ideal. Meta Prog
	used to create the languages mbeddr, PEoPL, and Realaxy. H
Incremental Programming Environment[14]	A syntax-directed language independent editor from 1980. C
Intentional Domain Workbench	Inspired by Charles Simonyi's 1995 essay "The Death of Con
	the IDW was the product of the company Simonyi founded
	In 2017 Microsoft, as a part of an "acquihire" bought Intenti
GANDALF[16]	Taking over from IPE
Synthesizer Generator	
Whole Platform	
Más	
Onion	
Scratch	
Blockly	
Prune	
Eco	
Lamdu	
Blueprints (Unreal Engine)	
Interlisp-D	
deuce	
Enso	
Cedalion	
Gentleman	

TABLE 2.2: An incomplete list of projectional languages

PEoPL

At the same time, projectional language workbenches like MPS [57] and Intentional [47]

Second, Behringer, Palz, and Berger (2017) present Projectional Editing of Product Lines (PEoPL), a DSL that enables multiple projections for variability mechanisms.

Such editors have existed since the 1980s and gained widespread attention with the Intentional Programming paradigm, which used projectional editing at its core.

Projectional editing, also known as structured editing or syntax-directed editing, is not a new idea; early references go back to the 1980s and include the Incremental Programming Environment [32], GANDALF [35], and the Synthesizer Generator [39]. Work on projectional editors continues today: Intentional Programming [44, 18, 45, 14] is its most well-known incarnation. Other contemporary tools [20] are the Whole Platform [9], M´as [3], Onion, and MPS [4]

Projectional Editors from the 1980s. GANDALF [35] and the Incremental Programming Environment (IPE) [32] do not attempt to make editing textual notations efficient; for example, they lack support for linear editing of tree-structured expressions. The Synthesizer Generator [39] avoids the use of projectional editing at the fine-grained expression level, where textual input and parsing is used. While this may improve editing efficiency, it risks the advantages of projectional editing, because language composition at the expression level is limited. Another work that implements and uses a DSL within the Synthesizer Generator [37] concludes: "Program editing will be considerably slower than normal keyboard entry, although actual time spent programming non-trivial programs should be reduced due to reduced error rates."

The Intentional Domain Workbench (IDW) is the most recent implementation of the Intentional Programming paradigm [44, 18], supporting diverse notations [45, 14]. Since it is a commercial, closed-source project without widespread adoption yet, we cannot easily study it or survey its users.

All contemporary projectional editors are part of language workbenches

An early example of a projectional editor is the Incremental Programming Environment (IPE) [16]. It supports the definition of several notations for a language as well as partial projections, where parts of the AST are not shown. However, IPE did not address editor usability; to enter 2+3, users first have to enter the + and then fill in the two arguments. Another early example is GANDALF [17]; the report in [20] states that the authors experienced similar usability problems as IPE: "Program editing will be considerably slower than normal keyboard entry, although actual time spent programming non-trivial programs should be reduced due to reduced error rates." The Intentional Programming project [9, 22] has gained widespread visibility and has popularized projectional editing; the Intentional Domain Workbench (IDW) is the contemporary implementation of the approach. IDW supports diverse notations [7, 23].

Scratch [15] is an environment for learning programming. It uses a projectional editor, but does not focus on textual editing; it relies mostly on nested blocks/boxes. So does GP [18]. Textual notations, and thus grammar cells, are not relevant. Prune [2] is a projectional editor developed at Facebook. The goal is explicitly to not feel like a text editor; the hypothesis is that tree-oriented editing operations are more efficient than those known from text editors. While this is an interesting hypothesis, our considerable experience with using projectional editing in real projects has convinced us that this approach is not feasible; hence the work described in this paper.

The Synthesizer Generator [21] is a projectional editor which, at the fine-grained expression level, uses textual input and (regular, textual) parsing. While this improves usability, it destroys many of the advantages of projectional editing in the first place, because language composition and the use of non-textual notations at the expression level is limited.

Eco [10] relies on language boxes, explicitly delineated boundaries between different languages used in a single program (e.g., the user could define a box with Ctrl-Space). Each language box may use parsing or projection. This way, textual notations can be edited naturally, solving the usability issues associated with editing text in a projectional editor.

*** gothere Lamdu [5], a functional, projectional language (no paper)

. Dataflow visual programming languages, such as Blueprints in the Unreal Engine [2], are often domain-specific.

Early syntax-directed source code editors included Interlisp-D (for Lisp's limited syntax) and Emily[1] (for PL/I's rich syntax).

deuce: lightweight structured editing in sketch-n-sketch

Blockly: https://developers.google.com/blockly

An early example of a ProjE is the Incremental Programming Environment (IPE) [4]. Another early example is GANDALF [5], which generates a ProjE from a language specification.

The Synthesizer Generator [7] is also a ProjE. However, at the fine-grained expression level, textual input and parsing is used. While this improves usability, it destroys many of the advantages of projectional editing in the first place, because language composition at the expression level is limited. In fact, extension of expressions is particularly important to tightly integrate an embedded language with its host language [8].

The Intentional Programming [2,3] project has gained widespread visibility and has popularized projectional editing; the Intentional Domain Workbench (IDW) is the contemporary implementation of the approach. IDW supports diverse notations [9,10].

Language boxes [11] rely on explicitly delineating the boundaries between different languages used in a single program (e.g., the user could change the box with Ctrl-Space). Each language box may use parsing or projection.

according to LWBC 2015 the 4 porjectional LWB that took place Enso, Mas, MPS, and whole

Cedalion's

What advantages does projectional editing bring?

Projectional editing gives advantages both to the language engineer and the program developers. There is a lot of crossover and repetition between papers written on the subject of projectional editing as to the advantages it brings. To that end, what follows is a synthesis of a number of papers as to the advantages given. Rather than attributing advantages in line to their citations, a helpful reference of papers which proclaim such advantages can be found in table 2.3.

Exploratory programming As with their progenitors, syntax-directed editors, modern projectional editors help guide a developer unfamiliar with a language. The defined editors with rigid syntax and pre-defined layout mean that only specific cells within the editor can be edited. This template style means the she does not have to worry about significance of spacing or indentation. Minutiae of syntactic adornments, such as statement ending semi-colons or enclosing matched brackets, are also not interfering with her exploration of the language space.

Advantage	#	Paper(s)
Exploratory programming	5	[17–21]
Correctness-by-construction	7	[17, 22–26]
Rich notation	22	[15, 17, 19, 20, 25–42]
Mixed notation	8	[18–20, 28–32]
Multiple views	9	[17, 19, 26, 28, 29, 31, 33, 37]
Language composition	23	[15, 19, 20, 22, 23, 25, 26, 28–32, 37, 38, 41–49]
IDE functionality	4	[17, 29, 31]
Language evolution	1	[50]
Ancillary data	5	[19, 29, 42, 48, 49]

TABLE 2.3: Papers describing advantages.

When creating code the developer is only presented with legal options within the current context. As the projection is context aware, relevant actions or options can be suggested and irrelevant ones can be removed. Thus, it is easier for her to explore which options and actions the language allows her to choose. Intelligent code completion does not have to be limited to single nodes. Whole subtrees can be inserted allowing the developer to explore the larger structures of the language.

Correctness-by-construction A projectional editor, by controlling the interaction between the developer and the AST, prevents her from writing syntactically incorrect code. The whole class of syntactical errors are made impossible, with the developer relieved of having to think about special characters and layout. Typing and scoping errors are removed by only allowing validly typed and scoped options for the developer.

The developer is only able to select statements that are legal in the context of the location within the AST. Code does not have to be disambiguated, as this happens at time of entry by the developer. If there are multiple items that share the same presentation in the editor, the developer chooses the relevant item, resolving the ambiguity to what she means rather than what the parser thinks she means.

Rich notation The choice of projection is unconstrained by the restrictions of code that needs to be parsed from a textual source. This freedom opens up diverse otherwise difficult or impossible to parse notations. Examples include tabular, mathematical expressions and symbols, diagrams, trees, images, forms, prose, sub- and superscript. Any visual form or shape that can be mapped to the AST can be used to represent the program in an editor.

With these notations one can better reflect the semantics of the program domain, which should aid comprehension. Mathematics has a rich history of use of notation. When writing a DSL for the Mathematics domain, the domain experts can interact with it in the centuries old language of their domain.

Of course the projections can also be projections of text. This is often the appropriate projection type if the developer interacting with the language's domain expertise is parser-based languages.

Mixed notation Because no parsing is required, the different forms of rich notation can be combined without the need to create a unifying parser. With all notations working on the same editor infrastructure mathematic symbols can be embedded within textual projections, within tables within graphical representations. As ambiguity is not an issue for the underlying AST, then mixing different notations becomes much easier.

Multiple views With the AST being the stored artefact rather than the notation, projectional editing allows the language engineer to define multiple views on the same model optimised for different tasks. Similar to how software architecture presents different views for different stakeholders interests, the same notational diversity can be achieve with specific editors targeted to experts in the various parts of the domain. A developer can switch between different projections of a node within a larger projection, to find the one that best suits their current task.

Because the architecture of a projectional editor follows the principles of model view controller, it is possible to have multiple simultaneous views of the model. This allows the developer to update a projection that is optimised for writing and immediately see its effect in a projection optimised for understanding.

Language composition Parser-based languages can support some modularization and composition, but a projectional editor allows easy and extensive modular language extension and composition. This is a result of the nodes of an AST being disambiguated at entry rather than through a parser. If two items with the same syntax are available at the same place, then the user will choose the one that they require, and therefore the node has an explicitly chosen meaning.

The composition of independently developed languages does not suffer from the syntactic or keyword clashes they would in two grammar defined languages. Because of the lack of ambiguity, every node referencing the concept that defines it, these languages, when put together, will not have structural or syntactic issues.

Composition can involve extending an existing language or embedding other languages in a host language without modifying the definition of said language. The ease of composition and extensions leads to the advantage of being able to build larger languages out of smaller modules.

IDE functionality Developers in mature languages are used to the functionality of mature IDEs. These functionalities include syntax highlighting, intelligent code completion or suggestion, and static analysis for errors and validation. As projectional languages store the AST rather than the the concrete syntax, they require an IDE to edit. Because of this, when a language engineer designs the language she also has to design the IDE.

Because the projection is based on the AST it always knows it's context. When the editor already knows the meaning of the node it is representing, then syntax highlighting is simple. From knowing it's context, it makes it much simpler to suggest intelligent code completions.

Always having a complete AST makes it much easier to validate scope, typing and other hard to implement code validators.

Language evolution Extending a language with new capabilities and syntax is simple. If the change is syntactic then the language engineer has to update an editor. If there is a semantic change then the language engineer can write a migration in the language, to transform a node of one concept to a different type, and the developer would have to run that migration on their code.

Ancillary data Data can be added to nodes that can augment the AST. This has shown to be useful for documentation, requirements traceability and product line feature dependencies.

What are the disadvantages of projectional editing?

Whilst there are fewer papers proclaiming the disadvantages of projectional editing, we repeated the approach of the previous section. Thus, we have synthesised the disadvantages from papers in the following sections and listed citations for these ideas in the table 2.4

We do not consider that the dearth of disadvantages discussed as evidence of projectional editings superiority. We consider that those who do not find projectional editing useful do not write papers about it.

Disadvantage	#	Paper(s)
Low adoption	4	[26, 30, 39, 41]
Unnatural user experience	11	[20, 25, 26, 28, 29, 31, 37, 39, 41, 47, 50]
Abigious syntax	1	[32]
Inflexibility	2	[20, 28]
lack of integration with text ecosystem	5	[20, 28, 39, 47]
Learning curve	5	[20, 29, 30, 38, 47]
Vendor lockin	2	[29, 31, 51]

TABLE 2.4: Papers describing projectional editing disadvantages.

Lack of adoption The ideas that proceeded projectional editing - structured editor or syntax-directed editor - has been around since the early 1970's yet has failed to be adopted widely. This argument is a bit of a tautological one, as the low adoption is perhaps an outcome of the other disadvantages of projectional editing. However, low adoption can lead to a viscious circlem where lack of adoption prevents further adoption.

Inconvenient or unnatural editing Early attempts at projectional editing presented an inconvenient and unnatural user experience when coding. These usability challanges, exempified by the tedious manner of entering code as per the order of the tree, compares badly to parser based languages. A traditional text based language can be entered the way it looks, by typing the characters one after another,

This bad reputation continues, despite massive improvements in projectional editors. Whilst there is no debate that projectional editing feels different, it is questioned as to

whether this inconvenience is an intrinsic property or a result of developers, through year of experience, being used to text based programming.

Modern projectional editors, when using a textual syntax face an "uncanny valley" issue. Whilst trying to simulate a text editor, the developers start to expect all of the functinoality of the text based IDEs. This is a particular weak spot, especially with regards to granularity and restrictions of cursor movement, insertion, deletion, selection, copy and pasting, and other interacting with text

Ambigious syntax One of the selling points of projectional editing, especially when it comes to language composition, is that there can be no ambigious syntax. Whilst this may be true for the AST, it is not so for the developer as they read this code on the screen. If one combined Drools and Basic rather than Java, the developer may become confused as to which language the “Then” keyword referes when she read it.

Inflexibility As a developer one has no flexibility in code layout, perhaps for enhanced readability. This is entirely in the hands of the language engineer when determining the projection rules.

Integration with the text based world Projectional editors do not store the definition of the program in the form of a plain text implementation in the concrete syntax. Instead the AST is stored and serialized in a format not meant to be human readable, such as XML.

This leads to an issue with integration with the infrastructure that has grown arounf text based programing languages. Two notable examples are text diffing, especially where branch merging is concerned and code sharing. The diffing issue can and has been solved within projectional editing tools. This is however difficult to integrate into the workflows of software development workflows that include multiple tools.

Textual source code can be be shared simply by email or websites, this is not the case with projectional code.

Learning curve For the language engineer, the necessity to develop an editor with a good user experience is much harder work than defining a grammar for a parsed language. The learning curve for the langauge engineer is significant, as by default, she has to think also of the IDE devleopment.

For the developer, the different style of editing takes some getting used to.

Vendor lock-in The nature of projectional editing is that what one edits is a projection of the AST and therefore an IDE is needed to do the projecting, as well as language definition. Organisations thinking of going the projectional route for their DSLs may fear being locked into a specific implementation of the concept. To be able to use previously developed languages would require using the same tool set. Changing to a different toolset for language design would require a significant re-skilling effort.

2.2.3 What are Language Workbenches?

2.2.4 What is MPS?

Chapter 3

Method: Systematic Review

To answer the first Research Question, what is the current stat of Projectional Editing, we conducted a systematic literature review. Hereafter, we describe the method we undertook.

Chapter 4

Method: Survey

The validity of the the Prototype was tested using a survey. If a survey is not well designed then it could lead to invalid or irrelevant outcomes. As well as describing the design and procedure of the survey, we also outline any threats to its validity in this chapter. Our choice of survey technique is a Questionnaire.

4.1 Questionaire Design

The questionaire can be found in Appendix D. To design the survey of our prototype, we followed a number of rules derived from the works of Bryman[52] and de Vaus[53].

As advised, first we devised a clear introduction to describe the research.

We considered existing questions. With regards to projectional editing, we requested the original questionaires from three papers [20, 25, 43] pertaining to tools developed using projectional editing. From these questionaires we found [X] that we considered and decided against using any of them.

When formulating the questions we had the specific research question “Which projections can help developers to get appropriate feedback about rules?” in mind.

The pool of Drools users that we were personally in contact with was incredibly small. Thus we had to rely on responses from strangers. For this reason, we tried to make the questionaire as quick to finish as possible. This meant we looked particularly hard at removing questions that did not help us to our research goal.

We piloted the questionnaire with both ourselves and our industrial supervisor.

The instructions to each of the questions were tested for clarity, by a non-technical third party.

The only open question was one for which we wished to extract sentiment. Rather than having yes/no questions, where appropriate we applied a likert scale[54].

The design of survey monkey layout makes sure that questions do not span pages.

The socio-demographic questions (skill level and experience) were left to the end and research based questions were toward at the beginning.

We took care to rework questions that were long, ambigious, general, or leading to not

be so. We also took care to remove jargon, negative wording, and questions that asked about more than one thing.

4.2 Participants

The requirement for participants is that they have at least a little experience with using Drools. It was our hope to get a statistically significant number of participants.

4.3 Validity

Non-response bias[55] will be addressed by making the questionnaire short and easy to answer. Because of the nature of the participant selection for this survey, it will be difficult to address the bias of self selection caused by voluntary response.

Common method bias, i.e. "variance that is attributable to the measurement method rather than to the construct the measures represent"[56] can be responsible for 25% or more of variable relational influence. As we are only conducting a single survey, we won't be able to do much to prevent this, however we will take the following small precautions. Testing the survey to remove question amiguity, mood influences, and length issues. Mixing the survey order of questions will be used to mitigate the issues caused by similarity of items, proximity of items, and location of items. We will mitigate survey administration biases by administering some of these questionnaires manually and some online. We will make sure that there are no right or wrong answers and aim toward fact based questions. We will vary the scales of our likert scales and the types of questions.

The main statistical methods to address this bias, i.e. "Harman's single factor test"[56] and the"marker variable"[57] were found to be lacking in grounding[58]. Marker variable is considered approriate if used with caution. With the size of our expected returns, it may not be possible to gain a statistically significant outcome.

4.4 Pre-test

The first attempt at the survey was sent to our industrial supervisor, who has experience with Drools. We used this to remove ambigiously worded or leading questions and test that the length was truley between 5-10 minutes. This lead to the following changes: [TODO: add changes after we have tested]

4.5 Sampling

As within our own professional network we only had aquantence with (6?) Drools developers, we had to expand our reach to those we did not personally know.

Our approach was first search Stakoverflow for question askers and answerers on the subject of Drools. Our preference was to find email addresses, failing that twitter contacts.

This proved to be quite limited, especially in attempting to get contact details, 13 email addresses and 6 twitter addresses.

Our next approach was to trawl our LinkedIn connections for anyone with drools as a proclaimed skill. Whilst we had no one in our direct contacts, at one level of separation we found 204 connections. From these we could extract 54 email addresses and 40 twitter addresses, with only a small crossover with the addresses harvested from StackOverflow.

We chose not to expand to third level contacts, as we thought this would be harder to sell as to why they shoudl feel comfortable answering us.

4.6 Procedure

The questions, as described in appendix D, was uploaded to Survey monkey.

To encourage response, especially amongst only tangentally known participants, we crafted a short introduction, using techniques designed to enhance response as discussed by amongst others Cialdini[59]. As seen in figure 4.1, the tatics discussed by Cialdini, as signposted in table 4.1.

Key	Tactic
1	Short option for those with no time
2	Credentials matter
3	Recognition, (this might backfire as I hardly remember any of my Linkedin connections)
4	Consistency, they reported they have Drools experience, so they must live up to it
5a&b	Social Proof - other people have already answered
6	showing value
7	Special because of scarcity
8	Labeling - "I see you to be a good person"
9	The word "Because" has an outweighed effect
10	Compliment Expertise
11	"Every little helps"
12	point out a fault
13	own the fault
14	ask a favour
15	add inconvenience
16	Rhyming
17	hand written note

TABLE 4.1: persuasion tactics in figure 4.1

These were sent to our list of drools using strangers and we sat back and awaited response.

The response rate was (xx) or xx%.

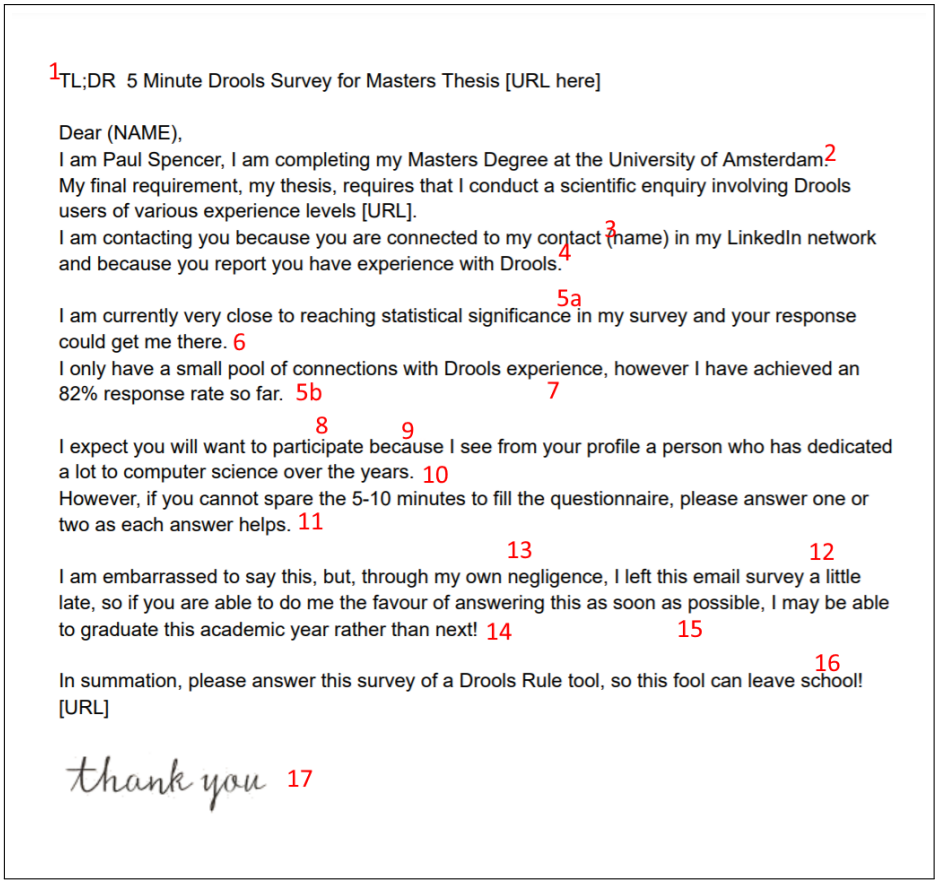

FIGURE 4.1: persuasive introduction.

Chapter 5

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 6

Discussion

6.1 Threats to Validity

6.1.1 Construct Validity

6.1.2 Internal Validity

6.1.3 External Validity

6.1.4 Reliability

6.1.5 Repeatability vs Reproducibility

6.1.6 Method improvement

Chapter 7

Implications to research and practice

7.1 Implications to research

7.2 Future research directions

7.3 Implications to practice

Chapter 8

Conclusion

We have built our projections as an aid to the understanding of Drools rules. This DSL extension includes many different ways to look at and interact with large code bases, as well as presenting options to deal with the complexity of individual rules. This means that they must be [TODO: PROPERTIES THAT AIDS UNDERSTANDING]. Our questionnaires show that we have reached that aim. Since developing our projections we have used them to model complex rules in our host organization.

Two factors lead to our success. First was the flexibility and extensibility of the MPS tool which presented the ability to develop and extend DSLs very efficiently. If we had tried this project without this tooling we would have [TODO: Finish our thoughts] Second [TODO: Finish our thoughts]

In this paper we described our work with first translating the Drools DSL into a projectional language followed by our explorations of projections. We discussed the advantages and disadvantages of the different projections we created and analysed experienced developers reactions to them.

Whilst we are convinced our projections [TODO: finish our thoughts]

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. 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.

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

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

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

[6] 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.

[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.

[13] S. Dmitriev, "Language oriented programming: The next programming paradigm," *JetBrains onboard*, vol. 1, no. 2, pp. 1–13, 2004.

[14] R. Medina-Mora and P. H. Feiler, "An incremental programming environment," *IEEE Transactions on Software Engineering*, no. 5, pp. 472–482, 1981.

[15] C. Simonyi, "The death of computer languages, the birth of intentional programming," in *NATO Science Committee Conference*, Citeseer, 1995, pp. 17–18.

[16] D. Notkin, "The gandalf project," *The Journal of systems and software*, vol. 5, no. 2, pp. 91–105, 1985.

[17] J. Klimeš, "Domain-specific language for learning programming," 2016.

[18] D. Ratiu, V. Pech, and K. Dummann, "Experiences with teaching mps in industry: Towards bringing domain specific languages closer to practitioners," in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, IEEE, 2017, pp. 83–92.

[19] M. Völter and E. Visser, "Language extension and composition with language workbenches," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2010, pp. 301–304.

[20] M. Voelter, J. Siegmund, T. Berger, and B. Kolb, "Towards user-friendly projectional editors," in *International Conference on Software Language Engineering*, Springer, 2014, pp. 41–61.

[21] M. Hosseinkord, G. Dulai, N. Osmani, and C. K. Anand, "Code and structure editing for teaching: A case study in using bibliometrics to guide computer science research," *arXiv preprint arXiv:2107.09038*, 2021.

[22] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. Kats, E. Visser, G. Wachsmuth, *et al.*, *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.

[23] D. Ratiu, M. Gario, and H. Schoenhaar, "Fasten: An open extensible framework to experiment with formal specification approaches," in *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*, IEEE, 2019, pp. 41–50.

[24] D. Ratiu, B. Schaetz, M. Voelter, and B. Kolb, "Language engineering as an enabler for incrementally defined formal analyses," in *2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, IEEE, 2012, pp. 9–15.

[25] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert, and J. Siegmund, "Efficiency of projectional editing: A controlled experiment," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 763–774.

[26] P. Vysok, P. Parizek, and V. Pech, "Ingrid: Creating languages in mps from antlr grammars," 2018.

[27] V. Pech, "Jetbrains mps: Why modern language workbenches matter," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 1–22.

[28] M. Voelter and S. Lisson, "Supporting diverse notations in mps'projectional editor.," in *GEMOC MoDELS*, 2014, pp. 7–16.

[29] M. Voelter and K. Solomatov, "Language modularization and composition with projectional language workbenches illustrated with mps," *Software Language Engineering, SLE*, vol. 16, no. 3, 2010.

[30] M. Voelter, A. v. Deursen, B. Kolb, and S. Eberle, "Using c language extensions for developing embedded software: A case study," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2015, pp. 655–674.

[31] M. Voelter, "Embedded software development with projectional language workbenches," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2010, pp. 32–46.

[32] S. M. Guttormsen, A. Prinz, and T. Gjøsæter, "Consistent projectional text editors.," in *MODELSWARD*, 2017, pp. 515–522.

[33] M. Voelter and B. Merkle, "Domain specific: A binary decision?" In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, 2010, pp. 1–6.

[34] A. Wortmann and M. Beet, "Domain specific languages for efficient satellite control software development," *ESASP*, vol. 736, p. 2, 2016.

[35] M. Voelter, D. Ratiu, B. Kolb, and B. Schaetz, "Mbeddr: Instantiating a language workbench in the embedded software domain," *Automated Software Engineering*, vol. 20, no. 3, pp. 339–390, 2013.

[36] C. Simonyi, M. Christerson, and S. Clifford, "Intentional software," in *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006, pp. 451–464.

[37] M. Voelter, T. Szabó, S. Lisson, B. Kolb, S. Erdweg, and T. Berger, "Efficient development of consistent projectional editors using grammar cells," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, 2016, pp. 28–40.

[38] V. Pech, A. Shatalin, and M. Voelter, "Jetbrains mps as a tool for extending java," in *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*, 2013, pp. 165–168.

[39] M. Voelter, J. Warmer, and B. Kolb, "Projecting a modular future," *IEEE Software*, vol. 32, no. 5, pp. 46–52, 2014.

[40] D. Ratiu, H. Nehls, and J. Michel, "Taming the software development complexity with domain specific languages," *Modellierung 2018*, 2018.

[41] M. Voelter, Z. Molotnikov, and B. Kolb, "Towards improving software security using language engineering and mbeddr c," in *Proceedings of the Workshop on Domain-Specific Modeling*, 2015, pp. 55–62.

[42] M. Voelter, B. Kolb, K. Birken, F. Tomassetti, P. Alff, L. Wiart, A. Wortmann, and A. Nordmann, "Using language workbenches and domain-specific languages for safety-critical software development," *Software & Systems Modeling*, vol. 18, no. 4, pp. 2507–2530, 2019.

[43] S. Meacham, V. Pech, and D. Nauck, "Adaptivevle: An integrated framework for personalized online education using mps jetbrains domain-specific modeling environment," *IEEE Access*, vol. 8, pp. 184 621–184 632, 2020.

[44] D. Pavletic, S. A. Raza, M. Voelter, B. Kolb, and T. Kehrer, "Extensible debuggers for extensible languages," *GI/ACM WS on Software Reengineering*, 2013.

[45] M. Voelter, "Language and ide modularization and composition with mps," in *International Summer School on Generative and Transformational Techniques in Software Engineering*, Springer, 2011, pp. 383–430.

[46] D. Ratiu, M. Voelter, Z. Molotnikov, and B. Schaetz, "Implementing modular domain specific languages and analyses," in *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation*, 2012, pp. 35–40.

[47] M. Voelter, D. Ratiu, B. Schaetz, and B. Kolb, "Mbeddr: An extensible c-based programming language and ide for embedded systems," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, 2012, pp. 121–140.

[48] M. Voelter and E. Visser, "Product line engineering using domain-specific languages," in *2011 15th International Software Product Line Conference*, IEEE, 2011, pp. 70–79.

[49] M. Voelter, D. Ratiu, and F. Tomassetti, "Requirements as first-class citizens: Integrating requirements directly with implementation artifacts," in *Proceedings of ACES-MB Workshop*, Citeseer, 2013.

[50] E. Schindler, K. Schindler, F. Tomassetti, and A. Sutii, "Language workbench challenge 2016: The jetbrains meta programming system," in *LWC SLE 2016 Language Workbench Challenge, Splash2016, 39 October-4 November 2016, Amsterdam, The Netherlands*, 2016.

[51] F. Tomassetti and V. Zaytsev, "Reflections on the lack of adoption of domain specific languages.," in *STAF Workshops*, 2020, pp. 85–94.

[52] A. Bryman, *Social research methods*. Oxford university press, 2016.

[53] D. De Vaus and D. de Vaus, *Surveys in social research*. Routledge, 2013.

[54] R. Likert, "A technique for the measurement of attitudes.," *Archives of psychology*, 1932.

[55] J. S. Armstrong and T. S. Overton, "Estimating nonresponse bias in mail surveys," *Journal of marketing research*, vol. 14, no. 3, pp. 396–402, 1977.

[56] P. M. Podsakoff, S. B. MacKenzie, J.-Y. Lee, and N. P. Podsakoff, "Common method biases in behavioral research: A critical review of the literature and recommended remedies.," *Journal of applied psychology*, vol. 88, no. 5, p. 879, 2003.

[57] M. K. Lindell and D. J. Whitney, "Accounting for common method variance in cross-sectional research designs.," *Journal of applied psychology*, vol. 86, no. 1, p. 114, 2001.

[58] G. Gorrell, N. Ford, A. Madden, P. Holdridge, and B. Eaglestone, "Countering method bias in questionnaire-based user studies," *Journal of Documentation*, 2011.

[59] N. J. Goldstein, S. J. Martin, and R. Cialdini, *Yes: 50 scientifically proven ways to be persuasive*. Simon and Schuster, 2008.

Appendix A

Protocol Validation Checklist

The protocol will be validated using the checklist in table A.1, which was adapted from Kitchenhams book EVIDENCE-BASED SOFTWARE ENGINEERING AND SYSTEMATIC REVIEWS[**Kitchenham_20**

Components	Questions
Background	Is the motivation for the review clearly stated and reasonable?
Research questions	Do these address a topic of interest to practitioners and/or researchers? Are they clearly stated?
Search strategy	Is the strategy justified and is it likely to find the right primary studies without the reviewers having to check or read a large number of irrelevant papers? Has the strategy been validated?
Study selection	Are the inclusion/exclusion criteria clearly defined and related to research questions? Is a validation process specified? Is there a process for handling marginal and uncertain papers? Is there a process for managing multiple reports of individual studies?
Quality of primary studies	Is it clear that the outcomes will be used in the later stages of the review? Is a validation process specified? Are criteria for assessing quality provided and justified and appropriate to the anticipated primary study types?
Data extraction	Does the data to be extracted properly address the research questions? Are the methods of recording the data appropriate for the types of data to be extracted? Is a validation process specified? Are there mechanisms for iteration where data is qualitative and categories are not (or cannot be) fully defined in advance of the extraction? Will the process enable the research questions to be answered?
Data aggregation and synthesis	Are the methods proposed for qualitative and quantitative data appropriate? Has consideration been given to combining results across multiple study types? Is the approach to aggregation and synthesis justified concerning appropriate literature?
Reporting	Has this been considered? has sufficient attention been paid to the completeness, general interest, validation, traceability, and the limitations of the review?
Review management	Are the tools that will be used for managing papers, studies, and data specified and appropriate? Is the management of the many-to-many relationship between papers and studies addressed?

TABLE A.1: Protocol Validation Checklist

Appendix B

Systematic Literature Review Log

B.1 Search Description

The papers we found with our automatic search can be found in table B.2. There were 100 unique papers found with the search strings across all of the venues when we used the search string and date restrictions. First we excluded those that were not in English or were unavailable to us. This reduced the count to 94. Next we downloaded each of the papers.

Next we skimmed all of these papers to remove any that were obviously unrelated to projectional editing. This reduced the count to 69.

two of the papers were referring to the same study, which reduced the count to 67.

We then excluded all grey literature, i.e. masters projects, proposals and phd thesies, and also books. This brought us down to 51 papers.

At this point we began our quality assesment. xxx papers fell below our quality threshold, which left us with xxx papers to assess to determine the current state of projectional editing. [TODO: fill in the quality numbers]

B.2 Table description

Table B.2 shows the log of the Systematic literature review.

The First column “Paper Title” is the name of the paper as given by the search engine.

The second column “Lib”, indicates the library or search engine through which it was found. The Libraries are Identified by the Keys in table B.1.

The third and fourth columns show inclusion and exclusion reasons. As inclusions only rely on one question, “does this paper discuss projectional editing”, the affirmative is indicated by a tick. The exclusion column includes the reason for the exclusion.

The fifth column, QA, gives the the Quality Assesment Criteria score. This score is calculated As described in the Methods section.

Ths final column “ref”, gives a link to the citations in the separate bibliography for the systematic literature review, found at the end of the Appendicies.

Key	Search engine/library	original #	final cut #
1	Google Scholar	82	?
2	IEEExplores	5	?
3	ACM	18	?
4	BASE	7	?
5	CORE	7	?
6	Web of Science	1	0
7	Microsoft Academic	10	?
8	SCOPUS	10	?
9	Semantic Scholar	10	?
10	SpringerLink	22	?
11	Wiley Online	1	?

TABLE B.1: Search Engine/Library Key

Paper Title	lib	in	exclusion	QA	ref
“Filmar, assistir e problematizar” – contribuições à aprendizagem de cálculos	9		not english		
20. Internationales Stuttgarter Symposium	10		book		
A Domain-Specific Language for Payroll Calculations: a Case Study at DATEV	1	✓			
A Domain-Specific Language for Payroll Calculations: An Experience Report from DATEV	1,10	✓	Duplicate		
A Framework for Modernizing Domain-Specific Languages	1	✓			
A Framework for Projectional Multi-variant Model Editors.	1,8	✓	gray		
A Generic Projectional Editor for EMF Models.	1,7,8,9	✓			
A language-driven Development framework for simulation components to generate simulated environments	1	✓	gray		
A Model-Driven Approach Towards Automatic Migration to Microservices	10	✓			
A survey of Model Driven Engineering in robotics	1	✓			
A Survey on the Design Space of End-User Oriented Languages for Specifying Robotic Missions	1,10	✓			
A survey on the formalisation of system requirements and their validation	1	✓			
A text-based syntax completion method using LR parsing	1				
Activities and costs of re-engineering cloned variants into an integrated platform	1				
AdaptiveVLE: An Integrated Framework for Personalized Online Education Using MPS JetBrains Domain-Specific Modeling Environment	1,2	✓			
Adding Interactive Visual Syntax to Textual Code	3	✓			
An approach to generate text-based IDEs for syntax completion based on syntax specification	1	✓			
An MPS implementation for SimpliC	1	✓			
Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment	1	✓			
Block-based syntax from context-free grammars	1,3,5	✓			
Bridging the worlds of textual and projectional language workbenches	1	✓	gray		
Classification Algorithms Framework (CAF) to Enable Intelligent Systems Using JetBrains MPS Domain-Specific Languages Environment	2,5	✓			

Paper Title	lib	in	exclusion	QA	ref
Code and Structure Editing for Teaching: A Case Study in using Bibliometrics to Guide Computer Science Research	1,9	✓			
ComPOS - a Domain-Specific Language for Composing Internet-of-Things Systems	1,4	✓	gray		
Concepts of variation control systems	1	✓			
Concise, Type-Safe, and Efficient Structural Diffing	3				
Constructing optimized constraint-preserving application conditions for model transformation rules	1				
Design & Evaluation of an Accessible High-Level Language for Advanced Cryptography	1	✓	gray		
Domain-specific languages for modeling and simulation	1				
Domain-Specific Languages in Practice	10	✓			
DSL Based Approach for Building Model-Driven Questionnaires	1,10	✓			
DSS-Based Ontology Alignment in Solid Reference System Configuration	10		unavailable		
Editing Software as Strategy Value	1				
Efficient editing in a tree-oriented projectional editor	1,3,7,8,9	✓			
Efficient generation of graphical model views via lazy model-to-text transformation	1,4	✓			
Efficient usage of abstract scenarios for the development of highly-automated driving functions	1,10	✓			
Enabling language engineering for the masses	1,3	✓			
Engineering Gameful Applications with MPS	1,10	✓			
Enhancing development and consistency of UML models and model executions with USE studio	1,3,7,8,9	✓			
Enterprise Information Systems	10		book		
Example-driven software language engineering	1,3	✓			
Exploring Visual Primitives for Authoring Source Code	1		gray		
FASTEN: An Extensible Platform to Experiment with Rigorous Modeling of Safety-Critical Systems	1,10	✓			
FeatureCoPP: unfolding preprocessor variability	1,3				

Paper Title	lib	in	exclusion	QA	ref
FeatureVista: Interactive Feature Visualization	1				
Filling Typed Holes with Live GUIs	3	✓			
First-class concepts: reifying architectural knowledge beyond the dominant decomposition	1,3	✓			
FORMREQ 2020	1,2,8		book		
Gentleman: a light-weight web-based projectional editor generator	1,3,4,7,8,9	✓			
GPP, the Generic Preprocessor	1				
Improving the usability of the domain-specific language editors using artificial intelligence	1	✓	gray		
Incremental Flow Analysis through Computational Dependency Reification	1,2	✓			
Incrementalizing Static Analyses in Datalog	1	✓	gray		
Integrating the Common Variability Language with Multilanguage Annotations for Web Engineering	1				
Integrating UML and ALF: An Approach to Overcome the Code Generation Dilemma in Model-Driven Software Engineering	10	✓			
Javardise: a structured code editor for programming pedagogy in Java	1	✓			
JetBrains MPS as Core DSL Technology for Developing Professional Digital Printers	1,10	✓			
JetBrains MPS: Why Modern Language Workbenches Matter	1,7,10	✓			
Learning Data Analysis with MetaR	1,10	✓			
Lipschitz-like property relative to a set and the generalized Mordukhovich criterion	6				
Macros for Domain-Specific Languages	3				
Mechanizing metatheory interactively	1	✓	gray		
Migrating Insurance Calculation Rule Descriptions from Word to MPS	1,10	✓			
Model Driven Software Engineering Meta-Workbenches: An XTools Approach	1,5	✓			
Model-based safety assessment with SysML and component fault trees: application and lessons learned	1,10	✓			
Model-Driven Development for Spring Boot Microservices	1,5	✓	gray		
n Challenges for Software Language Engineering	1				
On preserving variability consistency in multiple models	1,3				

Paper Title	lib	in	exclusion	QA	ref
On the Need for a Formally Complete and Standardized Language Mapping between C++ and UML.	1				
On the Understandability of Language Constructs to Structure the State and Behavior in Abstract State Machine Specifications: A Controlled Experiment	1				
On the use of product-line variants as experimental subjects for clone-and-own research: a case study	1				
PAMOJA: A component framework for grammar-aware engineering	1	✓			
Programming Robots for Activities of Everyday Life	1	✓	gray		
Programming tools for intelligent systems	1	✓	gray		
Projecting Textual Languages	1,10	✓			
Rule-based and user feedback-driven decision support system for transforming automatically-generated alignments into information-integration alignments	5		unavailable		
Semi-Automatische Deduktion von Feature-Lokalisierung während der Softwareentwicklung: Masterarbeit	5		not english		
Should Variation Be Encoded Explicitly in Databases?	1				
SLang: A Domain-specific Language for Survey Questionnaires	1	✓			
SpecEdit: Projectional Editing for TLA+ Specifications	1,4,7	✓			
Specifying Software Languages: Grammars, Projectional Editors, and Unconventional Approaches	1,2,4,5,7,8,9	✓			
Teaching Language Engineering Using MPS	10	✓			
Teaching MPS: Experiences from Industry and Academia	1,10	✓			
Teasy framework: uma solução para testes automatizados em aplicações web	1	✓			
The Art of Bootstrapping	10	✓			
The state of adoption and the challenges of systematic variability management in industry	1				
Toward a domain-specific language for scientific workflow-based applications on multicloud system	1,11				
Towards a Universal Variability Language	1	✓	gray		

Paper Title	lib	in	exclusion	QA	ref
Towards Multi-editor Support for Domain-Specific Languages Utilizing the Language Server Protocol	10	✓			
Towards Ontology-based Domain Specific Language for Internet of Things	1,3	✓			
Towards projectional editing for model-based SPLs	3,4,7,8,9	✓			
Tychonis: A model-based approach to define and search for geometric events in space	1				
Type-Directed Program Transformations for the Working Functional Programmer	1	✓			
Understanding Variability-Aware Analysis in Low-Maturity Variant-Rich Systems	1				
Untangling Mechanized Proofs	3				
Variability representations in class models: An empirical assessment	1,3				
Visual design for a tree-oriented projectional editor	1,3,4,7,8,9	✓	Duplicate		
What do practitioners expect from the meta-modeling tools? A survey	1,7,8,9	✓			
Cyrillic named paper 1	1		not english		
Cyrillic named paper 2	1		not english		

TABLE B.2: Systematic review log

Appendix C

InterviewTranscripts

Write your appendix content here.

Appendix D

Questionaire Text

Note: This questionnaire was presented on Survey Monkey and thus the text here is a best approximation of their paging system. pictures of these can be seen in [TODO: take Survey monkey pictures]

D.1 Page 1 - Introduction

This survey should take [X] minutes to complete. It is part of a Masters Project by Paul Spencer at the University of Amsterdam.

The purpose is to determine whether projectional editing can be used to aid the comprehensibility of Business rules from the point of view of a software developer. We are using Drools as our example business rules language.

You were selected as you asked or answered a Drools question on StackOverflow, listed Drools as a skill on your linkedin profile or were referred this survey by someone who did. It is therefore assumed you are aware of what Drools is.

Projectional editing is a form of writing computer programs directly rather than in text and having that parsed to create the program. It allows you to directly edit the program from different projections or views upon the code. In this survey we will present you a few of these views.

‘(Reliability) Does the language protect you against errors? How would you rate it on a scale of 0 to 10 (0 not at all reliable and 10 reliable at an excellent level)?

(Functional Suitability) Does the language contain all the functionality required for the data science part? For example, does it include the following: Uploading of the data, Choice of algorithm, Presenting the results. If you answered no, what should be included?

‘(Usability) How much would you rate (in a scale of 0 to 10) the language as regards to its usability? If you rated the usability over 5, can you please give a couple of good points that you found? If you rated the usability under 5, can you please add suggestions for improvements?