Projecting Rules: Improving Comprehension of Business Rules with Projectional Editing

Paul SPENCER
spencerpj@gmail.com

November 30, 2021, 122 pages

Academic supervisor: Dr. Clemens GRELCK, c.grelck@uva.nl
Daily supervisor: Toine KHONRAAD
Host organisation: Khonraad / Visma, https://www.visma.com/

UNIVERSITEIT VAN AMSTERDAM
FACULTEIT DER NATUURWETENSCHAPPEN, WISKUNDE EN INFORMATICA
MASTER SOFTWARE ENGINEERING
http://www.software-engineering-amsterdam.nl

Abstract

Rules engine languages are languages used to gather together and execute the rules of an application. Declarative rules languages, such as Drools, can become difficult to reason about when there are many rules. This project investigates whether projectional editing is a reasonable solution to this issue. If so, how can different projections of the Abstract Syntax Tree ease the comprehensibility of the code? We conducted a systematic literature review to ascertain the relevancy of projectional editing. We implemented the Drools language using the MPS language workbench and made innovative projections of Drools ASTs. We validated our projections with a survey. Projectional editing is still niche, though it makes some industrial and educational headway, particularly with JetBrains MPS. Projections can be helpful. Contributions of this project include an overview of the current state of projectional editing, an alternative base language for model-to-model transformations in MPS, and projections that provides a more compact manner to enter and view Drools rules.

Acknowledgements

We want to acknowledge The Graduate 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 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 us explaining MPS. Further, Sergej Koščejev from JetBrains helped us with specific MPS issues.

Further, we thank Michel Mercera, who, among other things, provided us with helpful guidance on conducting and analysing surveys.

Our greatest thanks go out to Toine Khonraad, an alumnus of this program, who provided us with the support, wisdom and friendship that aided in completing this project. Without his constant mantra of “simplify, simplify, simplify”, we would still be implementing the Drools language without making a single projection.

Contents

1

Introduction

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

1.1 Problem Statement

Rules engines are a place for gathering and executing the rules of a business application. Rules contain premises and implications. If the premise of the rules are met then the implications are executed. A rules engine decides which of the rules to execute.

Drools is one of the leading rules engines. Drools implements the rules engine paradigm using a version of the Rete algorithm. The rules are written in a declarative DSL, which will be the subject of this project. The language, shares an unfortunate characteristic with other rules languages. Namely, it is verbose and can contain many rules that can interact without an apparent visual connection. As Forgy[1] points out, for rules languages in general, "[they] have another property that makes them particularly attractive for constructing large programs: they do not require the developer to specify in minute detail exactly how the various parts of the program will interact". This property leads to very large and difficult to reason about collections of implicitly connected rules.

We found that reasoning over a small number of rules is already surprisingly hard. Our host organisation, Khonraad, a provider of mission-critical cloud software to the municipalities of the Netherlands, has many rules and, thus, reasoning about them is particularly challenging.

One approach to tackle comprehensibility could be to consider Miller's Law[2]. This law states that an average human can hold in his short-term memory 5-9 objects, which is often an argument for succinct code. The argument is that the developer must store anything that is not immediately in her vision in her memory. With it being impractical to reason about code that she cannot recall, the fewer relevant items to her reasoning that are out of view, the easier it is to reason about the code.

Through both the experience of the host supervisor and in conversation with the developers at Khonraad, we have observed the difficulty developers have to reason about and edit extensive collections of Drools rules. We hypothesise that when we present developers with different views on their code, they can better understand it. The problem we wish to solve - how to improve the ability to reason about sizeable collections of Drools rules - we believe lends itself to the technique of projectional editing.

Projectional editing is a technology implementation that allows users to directly edit the abstract syntax tree through representations of the tree called projections. This requires no parsing. The lack of parsing means that code can be represented without the restrictions of having to be textual. This property means that code can be edited using unparsable notations, such as mathematical, tabular or graphical notations.

Language Engineers can use language workbenches (LWB) to design languages. JetBrains MPS is a LWB to create projectional languages, in their own IDE. By using projections, developed in MPS, to improve feedback whilst coding, we believe that this can reduce the representation impedance mismatch that hampers the developer's reasoning.

To reason about a large code base of rules engine code effectively, perhaps a different presentation of that code is needed. The presentations we create should allow a more precise organisation of the code whilst re-

maining interactive.

1.2 Project Context

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

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

Configuration and administration require 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 crises, is crucial. Demonstration of the correctness of the configuration is a significant concern in the company.

The work environment at Khonraad allows us to work on an existing project, where success will impact the lives of those in critical need. The software that facilitates Khonraad's services uses Drools code. This code has evolved and grown together with the changes in the laws. Working in this environment means that we are also dealing with real-life issues and not just thought experiments.

1.3 Research Questions

We can formulate the following research question based on the discussion in the preceding sections being:

- **Main research question:** "Is projectional editing a viable and effective solution to issues developers have with reasoning about Drools business rules?"

This question requires knowing if it is even possible or practical with currently available tooling. As researchers, we would also like to know if this is a compelling area to research. Thus, we would like to answer the question:

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

Next, we specifically would like to know what we can create to improve the developer's ability to reason about the business rules engine, so we ask the question:

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

Finally, we would like to know how the current Drools developer community would respond to projectional editing, particularly the projections we create.

- **RQ 3:** "Do projections of Drools business rules lead to a greater understanding of those rules?"

1.4 Research Method

To answer the research questions from Section 1.3, we pursued three approaches.

Our chosen path to answering RQ 1 was to conduct a systematic literature review (SLR). The SLR aims to answer this question by interrogating current research with the sub-questions, "Is there any current research in the area of projectional editing?", "Which tools are currently being used for research?" and "What is the sentiment in papers currently discussing projectional editing?". We describe how we went about this in Section 3.1.

This SLR showed that whilst still niche, projectional editing is maturing, with one successful language workbench dominating the commercialisation of this field.

We attacked RQ 2 by implementing limited versions of the Drools language using the MPS language workbench and creating projections on top of this. We first created a minimal pilot version, called Really Simple Drools, which validated that this approach would work. Next, we created Drools-Lite, which had the look and feel of traditional Drools and enough functionality that we could present it to experienced Drools users. We describe this design science research (DSR) approach in detail in Section 4.1.

The outcome of this research was several projections that we felt would improve the understanding of larger collections of Drools rules.

Finally, we tackled RQ 3 with a survey. We presented the results of RQ2 to experienced industrial and academic Drools users, as laid out in Section 5.1. We compared some of our projections with textual projections similar to the code they were used to.

The survey showed that whilst there was interest in the approach and that some projections were more understandable than others, textual presentations, at least for this population, were still considered easier to understand.

1.5 Contributions

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

In summary, the concrete contributions of this project are:

- ***The Projectional Editing Status Quo*** in our SLR, we provide an overview of the current state of research in the field of Projectional Editing
- ***An alternative Base Language*** with a bit of extra work, the Drools-Lite language can be used as an alternative base language for model-to-model transformations in MPS.
- ***A new way to enter rules*** our implementation allows Drools developers a more compact manner to enter rules.

1.6 Thesis Outline

We start in Chapter 2 with the required background information on rules engines and projectional editing. Chapter 3 describes our efforts to answer our first research question using a systematic literature review. Chapter 4 describes the approaches and outcomes of answering the second research question using design science research. Chapter 5 brings us to research question 3 and the survey we used to explore it. We then investigate related work in Chapter 6. Finally, we present the conclusions of our research in Chapter 7.

2

Background

This chapter gives the background information that should help a reader get situated in this area of research. We begin, in Section 2.1.1, by clarifying what a rules engine is and the specific case of the one that we will be using for our investigation: Drools. Next, in Section 2.2 we delve into the world of projectional editing. Finally, in Section 2.3, we present the specific projectional editing tool we will be using: JetBrains MPS.

2.1 Rules Engines

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.

To paraphrase Quine[3], the Aristotelian doctrine of essentialism declares that a thing has essential properties and properties that are accidental. If one takes away the accidental properties of a thing, then the thing remains the thing. In contrast, if one takes away the thing's essential properties, the thing is no longer the thing. If the "thing" we refer to is a business application, its essential properties are its business rules.

Simply put, business rules are the principles or regulations by which an organisation carries out the tasks needed to achieve its goals. When adequately defined, it is possible to encode these rules into statements that define or constrain some business organisational behaviour. A rule consists of a condition and an action. When the condition is satisfied, then the action is performed. More formally, business rules follow the logical principle of Modus Ponens.

One could imagine this as just the "if-then" logic frequently used in traditional programming when described like this. One would not be wrong. However, representing all the combinatorial outcomes of an extensive collection of rules in traditional programming can become complex. Each additional rule, in this traditional style, increases complexity and decreases maintainability. In essence, it adds to the fragility of the codebase. Additionally, developers tend to distribute rules throughout the source code or database of the application.

We find descriptions of these rules in the design documentation or user manuals. However, as applications evolve, documentation gets out of sync with the codebase. Once desynchronisation occurs, to know the rules governing the application, one has to navigate the codebase and decode the rules from often scattered locations.

A rules engine is also known as a Business Rules Engine, a Business Rules Management System or a Production Rules System.

Rules Engines are declarative, focussing on the what of the rules, not the how of the execution. Date[4] describes a rules engine role as "to specify business process declaratively, via business rules and get the system to compile those rules into the necessary procedural (and executable) code." Fowler[5] describes a 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 ...".

The goal of a rules engine is to abstract business rules into encoded and packaged logic that defines the tasks of an organisation 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. Returning 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 would be:

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 recognise-act cycle. First, the match, i.e., are there any rules with a true condition? Next, they carry out conflict resolution, pick the most relevant matching rules. They then perform the actions described in the rule. Then back to the matching step. If they make no more matches, they terminate the cycle.

Some of the advantages of using a rules engine include:

- The separation of knowledge from its implementation logic.
- The externalisation of business logic.
- 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 A short history of the rules engine

Rule engines arose from the expert systems of the late 70s and early 80s. Expert systems initially had three primary techniques for knowledge representation: Rules, frames, and logic[6]. "The granddaddy" of the expert systems, MYCIN, relied heavily on rules-based knowledge representation[7] 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 1980s. 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 examined rules. Charles Forgy's efficient pattern matching Rete algorithm[1] and its successors solved this problem. This algorithm works by modelling the rules as a network of nodes where each node type works as a filter. A fact flows through the filters of this network. The pre-calculation of this network is what provides the performance characteristics.

The first popular rules engine was Office Production System (OPS) from 1976. In 1981 OPS5 added the Rete algorithm. Currently, there are a few rules engines in use. We show some of the more commonly used ones in Table 2.1.

Product		Developer	licence type
CLIPS	[8]	NASA	open source
Drools	[9]	JBoss/RedHat	open source
BizTalk Business Rule Engine	[10]	Microsoft	proprietary
WebSphere ILOG JRules	[11]	IBM	proprietary
OpenRules	[12]	OpenRules	open source

Table 2.1: Rules engine products

CLIPS is a very widely used expert system tool. Inspired by OPS, it was initially released in 1986 by NASA. It uses a mixture of rules and objects, written in C, to model human expertise.

We describe Drools in detail in Section 2.1.3, so we will skip that description here.

BizTalk is Microsoft's middleware offering for the .Net environment. Amongst its features is a Business Rule engine using Forgy's Rete algorithm. It has a simple graphical tool for exploring and building rules and policies, which is what they call their collections of rules. Microsoft is attempting to migrate customers from BizTalk to Azure App Service Hybrid Connections, which does not have a Rete based rules engine.

ILOG is a cross-platform Business Rules Management System now owned by IBM and rebranded as IBM Operational Decision Management. Rules are collected as policies in the rules repository. It is now part of IBMs WebSphere platform and is still actively updated.

OpenRules is an open-source Business Rules and Decision Management System. OpenRules, Inc. who offer paid support, training, and consulting services, currently maintain it. It allows the input of rules files using spreadsheets such as Excel and Google Sheets.

2.1.3 What is Drools?

JBoss Rules, or as it is more commonly known, Drools, is Java's leading opensource rules engine. When we use the name "Drools", 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 its 2005 2.0 release. It is an advanced inference engine using an enhanced version of the Rete algorithm, called ReteOO[13], 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.

To execute rules, Drools has four major components, as demonstrated in figure 2.1. The production memory contains the rules, and this will not change during an analysis session. The rules are the focus of this thesis. Thus, we will examine these in more detail later. The working memory contains the facts. A pattern matcher will match facts against rules and place them on the agenda. The agenda executes the rules and, if necessary, updates the facts in the working memory.

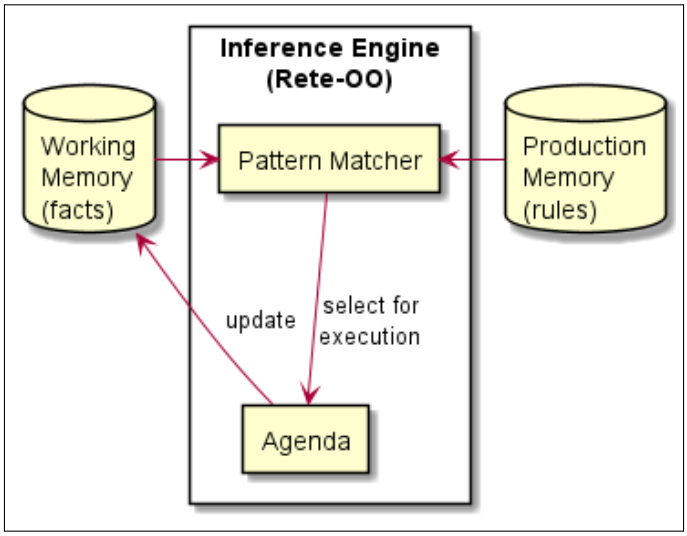


Figure 2.1: Drools components

In Forgy's[1] overview of a rete algorithm, the following steps occur.

1. Match: Evaluate the Left-hand sides (LHS) 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; halt the interpreter if no productions have satisfied LHSs.
3. Act: Perform the actions in the Right-hand side (RHS) of the selected production.
4. Re-evaluate: Go To 1.

Figure 2.2 shows more detail of how these components interact within Drools to infer a conclusion. First, Drools asserts facts in the working memory. The working memory contains the current state of the facts, which triggers the inference engine. Using the Rete-OO algorithm, the pattern matcher will examine both the working memory and a representation of the rules from the production memory to determine which rules are true. Drools will then put the rules that match on the agenda. It can be the case that many rules are concurrently true for the same fact assertion - these rules 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. We have inferred our conclusion if either a rule specifies to halt or no matching rules remain on the agenda.

The component we will be focussing on is the rules. A rules file containing the rules is a text file, typically with a .drl extension. The rules do not change during execution. Drools stores the rules in the production memory.

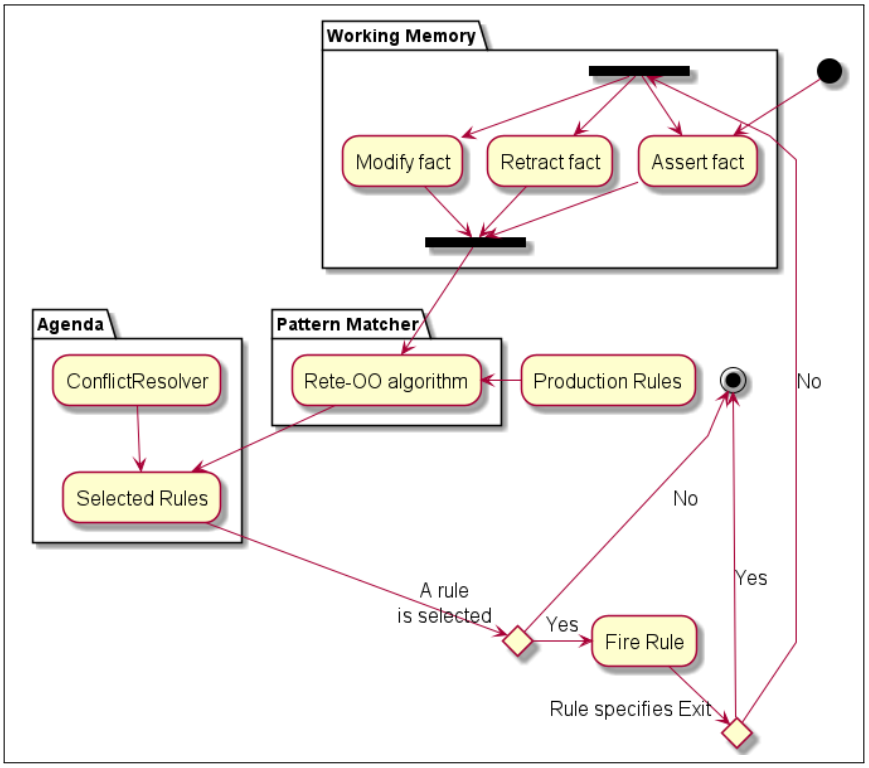


Figure 2.2: Drools inference loop

We will not examine the rule file components `package`, `import`, `global`, `declare`, `function`, and `query`. Explaining these components are not core to understanding how rules work.

We will now examine the anatomy of a rule, as shown in Figure 2.3 on the following page. A rule consists of 3 parts: attributes, conditions, and consequences. Attributes are optional hints to the inference engine as to how to examine a rule. The conditional, "when", or left-hand side (LHS) of the rule statement is a block of conditions subject to logical conjunctions. If true, then the rule is placed on the agenda. The actions, consequences, "then", or right-hand side (RHS) of the rule statement contains actions to be executed when the rule is selected.

The LHS is a predicate statement made up of some patterns. The patterns evaluate facts from the working memory. The pattern can match against the existence of facts or facts with matching property conditions. Logical operators, such as `not`, `and`, and `or` can combine patterns. The patterns apply to individual facts rather than the group, thus can be seen as first-order predicates.

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.

Drools offers more options for the LHS. We have limited our scope to the features described thus far.

The RHS can contain arbitrary Java code that will execute when a rule is selected. However, its primary 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 bound in the LHS. One can explicitly terminate the inference loop with a halt command.

2.1.3.1 An Explanatory Example

Listing 2.1 shows an example of a .drl file taken from the Drools sample code.

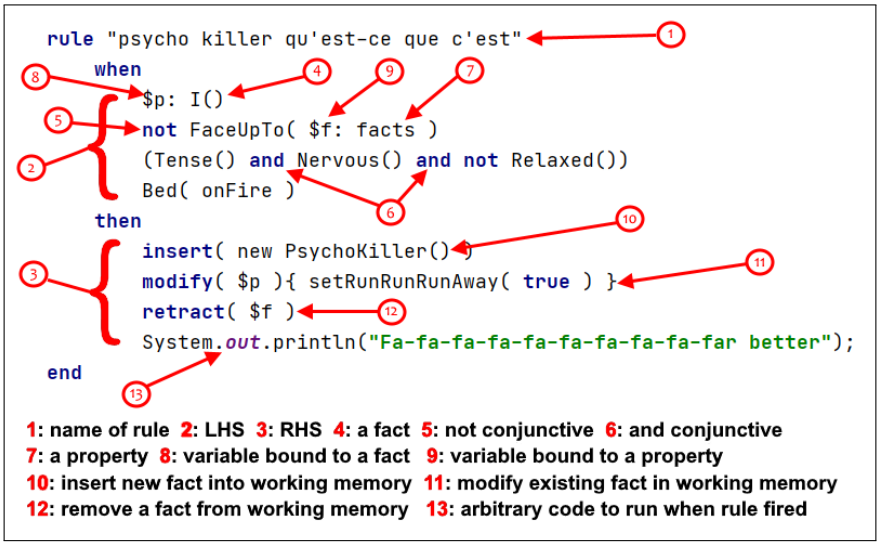


Figure 2.3: Drools rule breakdown

```
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 the working memory. This toy example reacts to when the working memory has an honest politician added. It prints a message celebrating the existence of said politician. It then corrupts her and gloats in a message. Finally, it 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 describe which facts are available for use.
3. The "We have an honest Politician" rule on line 6 does the following:
   (a) On line 7, using the `salience` attribute, the rule is set to be run before other rules with a lower `salience`.
   (b) On line 9, the rule checks working memory for `Politician` facts with the `honest` property equal to `true`.
   (c) On line 11, if found, a `Hope` fact is inserted into the working memory.
4. The "Hope Lives" rule on line 14 does the following:
   (a) Line 17 check if any `Hope` facts exist.
   (b) On line 19, if found, it prints a message.
5. The "Hope is Dead" rule on line 22 does the following:
   (a) On line 24, it checks that no `Hope` facts exist.
   (b) On line 26, if it finds no facts, then it prints a message.
6. The "Corrupt the Honest" rule on line 29 does the following:
   (a) Line 31 checks for any `Politician` facts with the `honest` property equal to `true` and sets them to the variable `$p`.
   (b) Line 32 checks if any `Hope` facts exist.
   (c) If both `Hope` and `Politician` facts are found, on line 34, it prints a message including the `$p` variables name.
   (d) On lines 35 to 37, it modifies the fact in working memory bound to the variable `$p` to change its `honest` property.

2.2 Projectional Editing

2.2.1 What is Projectional Editing?

When talking about projectional editing, we are mostly talking in the domain of metaprogramming. Usually, when we talk about software development, it is the programmer or developer who creates the program and the user who uses it. In metaprogramming, we are in the domain of the development of languages. Here the term "developer" could refer to both the creator and the user. We will distinguish these two roles by referring to the creators of the languages as "language engineers". Thus, when we refer to "developers", we mean users of the language that is the product of metaprogramming.

Traditionally developers 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 inverts this relationship, as a developer edits the abstract syntax tree and allows the IDE to project the concrete syntax.

2.2.1.1 Parser Based Editing

A program is defined using text and edited with a text editor in a traditional parser-based development workflow. A grammar is a definition of a programming language's formal syntactical rules or concrete syntax. One derives the lexer and parser from the grammar. The lexer will turn the text, via a text buffer, into tokens. A parser then validates that these tokens, the words of the language, are syntactically correct. The parser will often then construct a concrete syntax tree and an abstract syntax tree (AST), though not necessarily both.

A concrete syntax tree is the output of a text parsed complying with syntax of the grammar. It contains many tokens and keywords necessary to be unambiguously parsable that are without any semantic meaning.

An AST is a structure that represents the semantic meaning of the source code, stripped of all the syntactic details. The parser will carry out some of the name resolutions needed to ensure that the tree represents the references expressed within the source code. These references turn the tree into a graph.

Compilers use the AST to do subsequent processing, such as transformation, linking, code generation,

analysis, and type checking. Modern IDEs, in the background, also parse the code it is displaying to create an AST that it uses to offer relevant coding assistance. This assistance is valuable. Without help from the IDE, learning the concrete syntax of non-trivial languages is demanding. Exploratory programming is laborious if one must wait until compilation to discover mistakes.

2.2.1.2 Projectional Definition

In the projectional editing paradigm, a semantic model represents the program. This model requires projectional editing tools to be read and edited.

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

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 generate their AST. Instead, they project the underlying models/programs in a view. They store the model/AST in a custom format rather than its plain text equivalent in a traditional programming language.

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

The projection may sometimes seem like a text editor. However, this is just acrobatics by the language engineer designing an editor to help developers from traditional text-based languages feel comfortable. The text is just another type of projection of the AST. 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.

2.2.1.3 What is it Not?

Projectional editing does not have clearly defined boundaries. We exclude the following types of tools that sometimes get associated with projectional editing.

A Venn diagram of Model-Based Software Engineering (MBSE) and projectional editing would have a significant overlap. Here we will not be looking at tools that build code from UML or other MBSE or Model-Driven Engineering (MDE) tools.

Another area mistakenly grouped with projectional editing is Low-code software development environments. These, however, are only tangentially related.

Most confusing is “projectional editing” when referring to a methodology of product line differentiation in code bases. In addition to having the same name, one of the top products for this product line technique is called PEoPL. PEoPL uses MPS for development. Hence, this product is a projectional editor (the paradigm) for product line projectional editing (the methodology).

2.2.1.4 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 maps directly to a change in the AST.

To perform the above two items 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 connections used to build the hierarchical structure that is the AST. This hierarchy can have references to nodes in other branches, so it is a graph although named a tree. conceptually this can be seen in Figure 2.5. Each AST must have a root node. The nodes, represented by the ovals, can have child nodes. The child relationship is represented by the downward solid edges. Nodes can also reference other nodes in the AST, here represented by the dashed edges with arrows showing the direction of the reference.

The AST is stored independently of the concrete syntax, often using a database, XML, or a proprietary file format. Rules of the meta-model, such as type systems or scoping rules, must also be described.

Projectional editors avoid the grammars and parsers that define the concrete and abstract syntax in a traditional text-based language. In text-based languages, parsing transforms the concrete into the abstract. In projectional editing, the abstract is transformed to concrete using a projection engine that uses projection rules.

Editors combine the projection rules and the gestures or actions to create a change request to the AST. They are analogous to a concrete syntax.

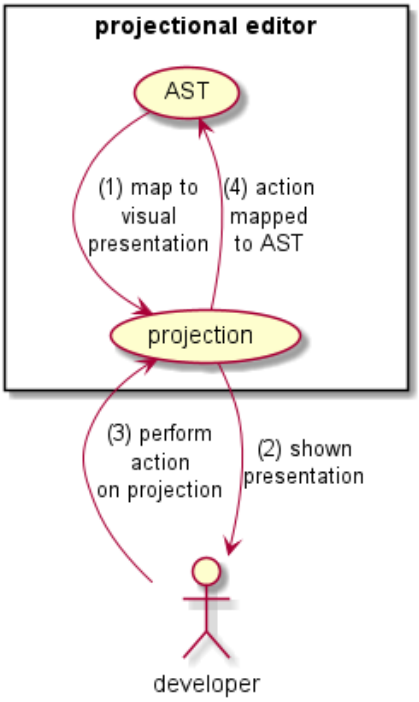


Figure 2.4: Projectional editing loop

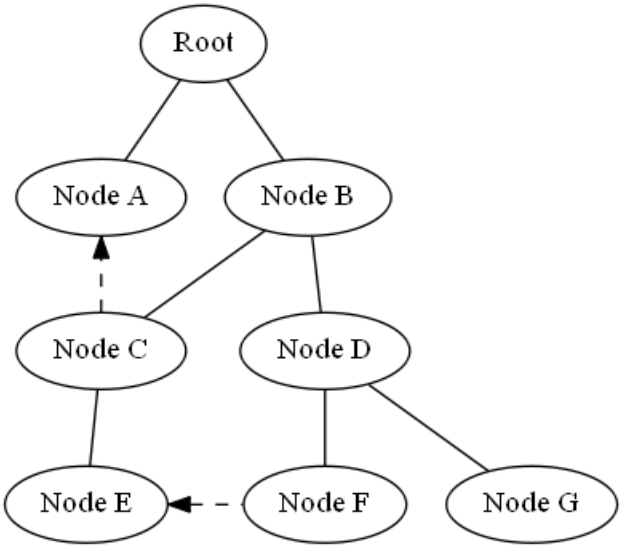


Figure 2.5: Conceptual AST

One of these actions can be typing text. However, every string is recognised when entered. Therefore, there is no tokenising. Text enters into the templates defined by the editor, and a newly derived projection displays the adjusted underlying AST to the developer.

The projection uses graphical elements to represent the model. 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. She builds the AST directly from each interaction she has with the editor.

Nodes are instances of the concepts defined in the meta-model. Each node has a unique id and points to its 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 the time of input, as the developer chooses 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. The implementation used for projections follows the model-view-controller (MVC) pattern. Therefore, multiple views of the model can be visible and updateable simultaneously.

Graphical modelling tools, for example, tools for UML modelling, could be seen as specialised implementations of projectional editing. These modelling tools do not store pictures of the UML diagrams and then parse them to create an AST. Instead, they store the model, often with extra information about the visual layout, and the image of the UML is projected to the modeller to edit. Projectional editing generalises this approach to projecting any notation defined by the language engineer.

2.2.2 History of Projectional Editing

Here follows a short history of projectional languages.

In the '70s and early '80s, researchers created several applications for research into the realm of structured editors. Some examples were: MENTOR[14], Incremental Programming Environment[15], GANDALF[16], Cornell Program Synthesizer[17], and Synthesizer Generator[18]. These language-based program editors could force syntactically correct programs. This syntactic guidance is achieved through the explicit knowledge of the language. These were the precursors to the modern projectional editors. They worked by providing templates for each abstract computational unit of the language. First, one would choose the concept and then fill out the placeholders.

These tools were not good at editing textual notations, which led to a poor user experience. When they

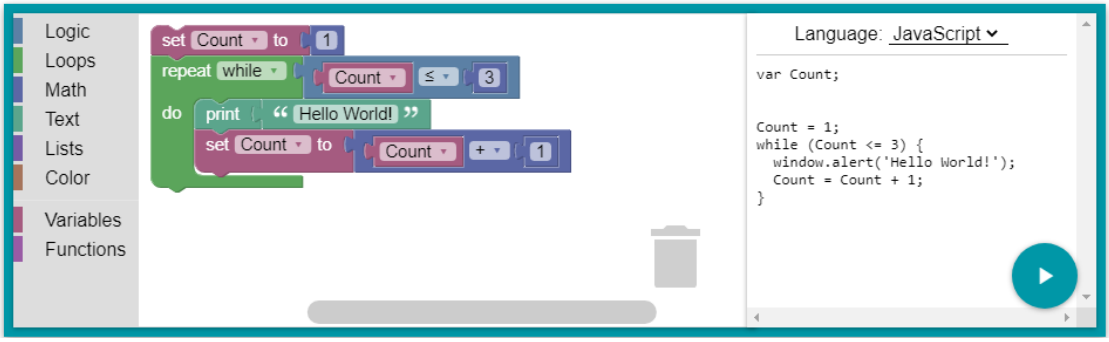
attempted to fix the poor user experience, for example, in the Synthesizer Generator, they reintroduced parsing to parts of the editor, which took away many advantages of the AST's direct editing.

In the late '90s and early '00s, the first forays into commercialising a more generalised version of structured editors, the projectional editors, began. The first of these was Intentional Domain Workbench (IDW), inspired by Charles Simonyi's 1995 essay "The Death of Computer Languages, The Birth of Intentional Programming"[19]. IDW was the product of the company Simonyi founded in 2002 - Intentional Software. The Intentional Programming paradigm spotlighted the projectional editing domain, taking it out of the universities and into practice. Unfortunately, as it was a closed sourced and expensive product, not many papers were written about it. Ultimately, in 2017 as part of an "acquihire", Microsoft bought Intentional Software for its employees and let the product die.

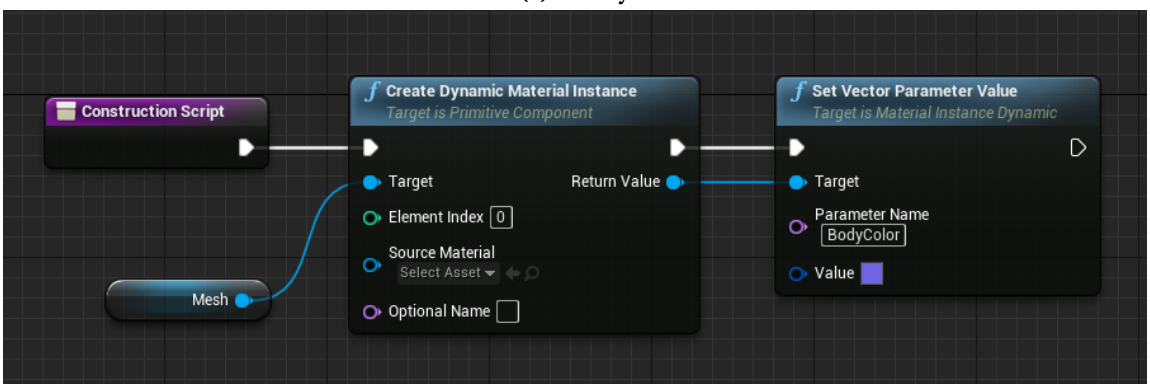
Inspired by a call to action for language orientated programming[20], JetBrains embarked in 2004 on a mission to build a product to fulfil that ideal. Meta Programming System (MPS) was the outcome of that journey. Language engineers created the languages mbeddr, PEoPL, and Realaxy using the MPS platform. It currently has an active community of developers and projects both in academia and in the commercial world. We chose this tool to be the basis of our projectional editing experiments. We will talk about it at greater length in Section 2.3.

The last decade has produced a few smaller projectional workbenches. There are a few open-source, small team projectional projects. In 2013 several projectional language workbenches joined MPS in the Language Workbench challenge[21]. These included Más, a web-based projectional editor, which is no longer with us[22]. Whole Platform[23] is a projectional language workbench plug-in for Eclipse. Cedalion[24] provides another projectional IDE, specialising in internal DSLs.

More recently, there have been some new products that intersect the projectional domain. Deuce[25] and Gentleman[26] are two projectional editors that have recently emerged from academia. The final two mentions in our incomplete history are a little out of left field. Google's Blockly[27] is a tool for making structural editor languages, but only in a block format. Blockly can create languages similar in style to the Scratch language. Blueprint visual scripting, a part of the Unreal Engine, is a visual programming language for building concepts such as levels or game assets. Examples of Blockly and Blueprint can be seen in Figure 2.6a and Figure 2.6b, respectively.



(a) Blockly



(b) Blueprint

Figure 2.6: Unconventional projectional editors

2.2.3 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 projectional editing regarding its advantages. To that end, what follows is a synthesis of several papers as to the advantages that projectional editing claims, in no order. Rather than attributing each advantage to each paper, we have made a reference table, Table 2.2, of papers proclaiming said advantage.

Advantage	#	Paper(s)
Exploratory programming	5	[28–32]
Correctness-by-construction	7	[28, 33–37]
Rich notation	22	[19, 28, 30, 31, 36–53]
Mixed notation	8	[29–31, 39–43]
Multiple views	9	[28, 30, 37, 39, 40, 42, 44, 48]
Language composition	23	[19, 30, 31, 33, 34, 36, 37, 39–43, 48, 49, 52–60]
IDE functionality	3	[28, 40, 42]
Language evolution	1	[61]
Ancillary data	5	[30, 40, 53, 59, 60]

Table 2.2: Papers describing advantages

Exploratory programming As with their progenitors, syntax-directed editors, modern projectional editors help guide a developer unfamiliar with a language. With their rigid syntax and predefined layout, the editors only allow editing within specific cells of the editor. This template style means that the developer does not have to worry about the 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.

When creating code, the editor only presents the developer with legal options within the current context. As the projection is context-aware, with relevant actions and options suggested and irrelevant ones removed. Therefore, it is easier for the developer to explore what the language allows her to choose. Intelligent code completion does not have to be limited to single nodes. Inserting whole subtrees allows the developer to explore the larger structures of the language.

Correctness-by-construction A projectional editor prevents her from writing syntactically incorrect code by controlling the interaction between the developer and the AST. The whole class of syntactical errors is 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 can only 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 the time of entry by the developer. If multiple items share the same notation in the editor, the developer chooses the relevant item, thus resolving the ambiguity to what she means rather than what the parser thinks she means.

Rich notation Constraints associated with textual parsing do not affect the choice of created projections. This freedom allows otherwise tricky 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 map onto the AST can 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. Textual projections are often the appropriate projection type. This suitability is especially true if the domain expertise of the developer is parser-based languages.

Mixed notation As no parsing is required and ambiguity is not an issue for the underlying AST, it is straightforward to combine different forms of rich notation. With all notations working on the same editor infrastruc-

ture, embedding mathematic symbols within textual projections, within tables within graphical representations is a simple coding pattern.

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. In the practice of software architecture, one presents different views to different stakeholders based on their interests. Similarly, projectional editing can present experts with various domain expertise views on the model that reflect their needs. A developer can switch between node projections within an enclosing 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. These multiple views allow the developer to update a projection optimised for writing and immediately see its effect in a projection optimised for understanding.

Language composition Parser-based languages can support some modularisation and composition, but a projectional editor allows easy and extensive modular language extension and composition. This ability results from the disambiguation of the nodes of an AST at the time of entry. If two items with the same syntax are available at the same place, the user will choose the one 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.

Language composition can involve extending an existing language or embedding other languages in a host language without modifying its definition. The ease of composition and extensions allows building more significant 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 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. A projection always knows its context because it comes from the AST. When the editor already knows the meaning of the node it represents, syntax highlighting is simple. Knowing its context 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 Parsing complicates the evolution of languages. For example, adding a new reserved word is difficult without breaking existing code. Extending a language with new capabilities and syntax in projectional editing 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 added to nodes can augment the AST. This data is helpful for tasks such as documentation, requirements traceability and product line feature dependencies.

2.2.4 What are the Disadvantages of Projectional Editing?

Whilst fewer papers proclaim 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 Table 2.3.

Disadvantage	#	Paper(s)
Low adoption	4	[37, 41, 50, 52]
Unnatural user experience	11	[31, 36, 37, 39, 40, 42, 48, 50, 52, 58, 61]
Ambiguous syntax	1	[43]
Inflexibility	2	[31, 39]
Lack of integration with text ecosystem	5	[31, 39, 50, 58]
Learning curve	5	[31, 40, 41, 49, 58, 62]
Vendor lock-in	2	[40, 42, 63]

Table 2.3: Papers describing projectional editing disadvantages

We do not consider that the dearth of disadvantages discussed as evidence of projectional editing's superiority. Our best guess is that those who do not find projectional editing beneficial do not write papers about it.

Lack of adoption The ideas that proceeded projectional editing - the structured or syntax-directed editor - have been around since the early 1970s yet have 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 self-reinforcing process, 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 challenges, exemplified by the tedious manner of entering code as per the tree's order, compare poorly to parser-based languages.

This poor reputation continues, despite massive improvements in projectional editors. Whilst there is no debate that projectional editing feels different, some question whether this inconvenience is an intrinsic property or a result of developers, through years of experience, being used to text-based programming.

Modern projectional editors, when using a textual projection, face an "uncanny valley" issue. Whilst trying to simulate a text editor, the developers start to expect all the functionality of the text-based IDEs. This expectation is an especially weak trait regarding granularity and restrictions of cursor movement, insertion, deletion, selection, copy and paste, and other interactions with the text.

Ambiguous syntax One of the selling points of projectional editing, especially concerning language composition, is that there can be no ambiguous syntax. While there is no ambiguity in the code, there can be ambiguity in the reader's mind. This ambiguity occurs as the same notation can be projected from different nodes. For example, if one combined Drools with Basic, the developer might become confused about which language the "`Then`" keyword was coming from. Consequently, writing ambiguity is replaced by reading ambiguity.

Inflexibility A developer using a projectional editor has no flexibility in code layout. They may feel they require this for enhanced readability. The flexibility of the layout is entirely in the hands of the language engineer when she determines 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 serialised in a format optimised for the computer rather than the human reader.

This different format of program storage leads to an issue with integration with the text-based ecosystem. This ecosystem is extensive, as text-based coding has been popular since the 60s. Two notable examples are code sharing and text diffing for branch merging. The diffing issue within projectional editing tools is solved. However, as codebases often span multiple programming languages and tools, the difficulty of integrating projectional diffing into the software development workflows is still a real problem.

Textual source code can be shared simply by email or on websites. This sharing, however, is not easy 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 language engineer is significant, as, by default, she has to think also of the IDE development.

For the developer, especially one with an extensive text-based experience, the different editing style 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 and language definition. The fear of getting locked into a specific concept implementation can negatively impact evaluations of projectional editing by organisations. To be able to use previously developed languages would require using the same toolset. Changing to a different toolset for language design would require a significant re-skilling effort.

2.3 What is MPS?

Language workbenches (LWB) are tools to help language engineers create languages, particularly domain-specific languages (DSL). Fowler[64] popularised the term LWB in a 2005 article.

Meta Programming System (MPS) is an open-source LWB that assists in the creation of projectional languages. It started in 2003 by JetBrains and was introduced to the world in Dmitriev's 2004 Paper "Language Orientated Programming: The Next Programming Paradigm"[20].

As discussed in Section 2.2.1, when creating a projectional language, one must define the language and how one interacts with it. In MPS, the language engineer defines languages, including their interactions. Developers create programs using these languages. The language engineer can extend languages. The developer can mix the languages she uses.

The following is an overview of how MPS implements the ideal of a projectional language. It is also the structure of this section:

- Abstract syntax
  - Structure
  - Behaviors[1]
  - Constraints
  - Type system
- Concrete syntax
  - Editors
  - Intentions
- Generators
  - Model-to-Model
  - Model-to-Text

MPS defines the different aspects of the language definitions with small, declarative DSLs. These are bundled together into what they term Aspects.

2.3.1 Abstract Syntax: Structure

Structure is what determines the abstract syntax of a language. The most important item available in a Structure Aspect is the Concept. Instances of concepts are called nodes. With these nodes, the developers construct their programs. When referring to a program in MPS, we are talking about its stored abstract syntax tree (AST).

In principle, a concept contains three types of things:

1. Properties: these primitives are integer, boolean, string, or enum items and are similar to leaf values.
2. Children: these are other concepts, or collections of them, similar to subtrees.
3. References: these are relationships with other nodes in the AST. These turn the tree into a graph.

[1] When referring to its use in the MPS workbench, consistent with their use, we use the American spelling - Behavior.

Concepts follow some object orientated (OO) traits, such as subtype, being abstract, and implementing interfaces.

One of these Concepts must be a root node. Otherwise, there is nowhere for a program to start.

Other items available in the Structure Aspect are the Interface Concept, the Enumeration, the Constrained Data Type, and the Primitive Datatype. [2]

Thus, the Structure aspect defines how the AST can be structured.

Figure 2.7 shows a concept with three children that implements two interfaces. The first line names the concept and which concept it extends. By default, all concepts extend `BaseConcept`. The next two lines shows the interfaces it implements. This concept is not rootable, and therefore cannot be instantiated as a file in a solution. the alias and short description override what will be shown in menus instead of `RuleStatement` and its fully qualified name. This concept has no custom properties or references. It has three children. The attributes child contains one and only one `RuleAttribute` node. Outcomes also contains only one child node, this one of type `StatementList`. The conditions child has a collection of `AbstractCondition` nodes.

```
concept RuleStatement extends    BaseConcept
                      implements INamedConcept
                                 IFileLevelStatement

  instance can be root: false
  alias: rule
  short description: rule

  properties:
  << ... >>

  children:
  attributes : RuleAttributes[1]
  conditions : AbstractCondition[0..n]
  outcomes   : StatementList[1]

  references:
  << ... >>
```

Figure 2.7: Concept example

2.3.2 Abstract Syntax: Behaviors

OO design usually bundles together data and methods that can act on that data. Concepts are analogous to the data part of this equation. Behavior fills the role of the methods in the OO analogy, defining the functionality called from instantiated nodes and static methods called from the Concept. The Constructor is a specialised method in a Behavior, filling the same role as a constructor in OO.

The methods have public, private, or protected visibility. If the Concept to which the Behavior refers is abstract, the Behavior itself can contain abstract methods. Abstraction, variable visibility, and inheritance allow a sort of polymorphism. If a virtual method is declared, then it can be called polymorphically.

Figure 2.8 shows a constructor added to a Concept to initialise its children. It has a method to allow other nodes to interrogate the condition of it having attributes.

[2] At the time of writing, we are unaware whether Data Type and Datatype are semantically different or that the different naming is just a style choice.

```
concept behavior RuleAttributes {

  constructor {
    this.salience = new node<SalienceAttribute>();
    this.salience.salience = new node<IntegerConstant>();
    this.salience.salience:IntegerConstant.value = 0;
    this.noloop = new node<NoLoopAttribute>();
  }

  public boolean hasAttibutes() {
    return this.salience.visible || this.noloop.visible;
  }

}
```

Figure 2.8: Behavior example

2.3.3 Abstract Syntax: Constraints

A Constraints aspect adds further structural restrictions to a Concept. Constraints primarily define scope by controlling if another node can be a child, a parent, or an ancestor of this node. A Constraint can also prevent badly formed properties, children, or references.

Figure 2.9 shows an example of a scope restraint that only allows local variables declared within the same rule or global variables declared in the same file.

```
concepts constraints RuleVariableRef {
  can be child <none>

  can be parent <none>

  can be ancestor <none>

  instance icon <none>

  <<property constraints>>

  link {target}
    referent set handler <none>
    scope (referenceNode, contextNode, containmentLink, position,
                                        linkTarget)->Scope {
      node<RuleStatement> rule =
            contextNode.ancestor<concept = RuleStatement, +>;
      nlist<RuleVariable> localVars =
            rule.descendants<concept = RuleVariable>;
      sequence<node<RuleVariable>> globalVars =
            rule.containingRoot.descendants<concept = GlobalStatement, +>;

      return ListScope.forNamedElements(localVars.concat(globalVars));
    }
    <no presentation (deprecated)>


  default scope <no default scope>
}
```

Figure 2.9: Constraint example

2.3.4 Abstract Syntax: Type System

The Type system aspect and the constraints aspect together represent the static semantics of the language. This aspect is for the computation and evaluation of types of variables, expressions, and statements.

Rules that are available to calculate and enforce the type system include inference, subtyping, comparison and substitute type rules.

Figure 2.10 shows an inference rule that ensures that the calculated type of the import statement matched that of its child “type”.

```
inference rule typeof_ImportStatement {
  applicable for concept = ImportStatement as importStatement
  applicable always
  overrides false

  do {
    typeof(importStatement) :==: typeof(importStatement.type);
  }
}
```

Figure 2.10: Type system example

2.3.5 Concrete Syntax: Editors

Editor aspects define the notation of the nodes. In effect, it is the user interface of the language, projecting the AST to the developer. An editor is a swing panel that renders a tree of editor cells. A Concept can have multiple editors, thus offering multiple views on it.

The definition of the options available to the developers through menus also happens within the Editor Aspect. The choice the developer makes transforms the existing AST.

Additionally, the behaviour of interactions can be defined, such as what will happen to the AST when a particular keypress or editor action occurs at a particular location.

Figure 2.11 shows a component with a projection for the Concept shown in Figure 2.7.

```
editor component rulestatement
  overrides:

  applicable concept:
    RuleStatement

  component cell layout:
    [/
      [> rule " { name } " <]
      ?[> ---> % attributes % <]
      [> ---> when <]
      [> ---> ---> (/ % conditions %  /) <]
                  /empty cell:
      [> ---> then <]
      [> ---> ---> % outcomes % <]
      end
    /]
```

Figure 2.11: Editor example

2.3.6 Concrete Syntax: Intentions

In projectional editing, the IDE is a part of the concrete syntax. Intentions make context-aware suggestions for automatic changes to the program to the developer. Figure 2.12 shows an intention that allows the developer to add, remove or edit a property based on its current value.

```
intention addSalienceRule for concept RuleStatement {
  error intention : false
  available in child nodes : true
  child filter :

  description(node, editorContext)->string {
    node.attributes.salience.visible ?
                          "Remove Salience Attribute" :
                          "Add Salience Attribute";
  }

  execute(node, editorContext)->void {
    if (node.attributes.salience.visible) {
      node.attributes.salience.visible = false;
    } else {
      node.attributes.salience.visible = true;
      node.attributes.salience.select
                              [in: editorContext, cell: FIRST_EDITABLE]
    }
  }
}
```

Figure 2.12: Intention example

2.3.7 Generators

In our implementation we will not be using generators. However, as they are an important part of a typical language implementation, we will discuss them briefly here.

There are two types of generators, the Model-to-Text generator and the Model-to-Model generator.

Whilst designing a language is nice, it should be able to do something. Without doing so, it has no semantic meaning. It is possible to create interpreters that can use the AST generated by MPS.

However, the most common modality for MPS is to generate, via a model-to-text generator, an output that gets compiled and run by commonly known environments. The output stage of generation is called TextGen. It defines how a node becomes runnable code in plain text.

Base level languages will have text generation. Most DSLs will perform model-to-model conversions, eventually converting to a base language. These intermediate stages are known as Generator Aspects. They transform code written in one language to another.

In MPS, a Concept can have multiple generators aimed at different base level languages, such as Java (using MPS BaseLanguage), C (using mbeddr) or XML.

2.4 Summary

Business rules languages codify business rules to be executed in an application. Drools is one of the most popular DSLs for business rules. Drools can become difficult to reason about when it contains many rules.

Projectional editing is the concept of updating and storing the AST directly. All interactions to update the AST is done through editors that are projections of the AST. These projections can take many forms.

MPS is a language workbench that specialises in creating projectional editors for DSLs.

3

Systematic Literature Review

In this chapter, to answer the question “What is the current state of language workbenches supporting projectional editing?” we embarked on a systematic literature review (SLR). In Section 3.1, we describe the method of this SLR. Section 3.2 examines the outcome of our research. Finally, in Section 3.3, we discuss the threats to the validity of our approach.

3.1 Method

To answer the first Research Question, “what is the current state of Projectional Editing?” we conducted a systematic literature review. Hereafter, we describe the method we undertook. We followed Kitchenham’s[65] advice on systematic literature review protocol validation to carry out this review.

3.1.1 Motivation

The motivation that preceded this research was a requirement to understand if projectional editing was an idea that was worth investigating. Our background research showed an interest in the precursors to projectional editing in the late ’70s through to the mid-’80s. Outside of academia, interest arose in the mid-’90s following Charles Simonyi’s treaties on Intentional programming. However, Simonyi’s call to arms did not lead to a swell in academic research as his company’s product, Intentional Domain Workbench, was a closed commercial product. Conversely, after JetBrains’s opensource Meta Programming System (MPS) release, in the late 2000s, there was a flurry of papers on the subject.

Is there a need for a study of this topic? We believe, at least in the microcosm of this master’s project, it is helpful to know whether we are researching in a dying or vibrant area.

There does not seem to be any recent SLRs specifically about projectional editing. This study is not extending any previously conducted SLR. Although there exist literature surveys and mapping studies in some adjacent fields, we found no SLRs about projectional editing. Thus, we believe it may be helpful for those in the language engineering research community to bring together all current research about projectional editing in one place.

3.1.2 Research Question

We try to answer the question "What is the current state of Projectional Editing?" with an SLR. We have broken this question into three sub-questions.

- **Sub Question 1** “Is there any current research in the area of projectional editing?”
- **Sub Question 2** “Which tools are currently being used for research?”
- **Sub Question 3** “What is the sentiment in papers currently discussing projectional editing?”

3.1.3 Search Strategy

The search process is automated as SLRs require a high level of completeness, which one cannot effectively achieve manually. Our first major decision was whether to engage in creating a quasi-gold standard as advised by Zhang[66]. Zhang noted that the ad-hoc nature of search strategies in SLRs has limitations. We executed a preliminary ad-hoc search to try and ascertain the extent of the research space. After satisfying that the

research space was small enough, we decided against using the Quasi-Gold standard, which was overkill for our requirements.

The search terms we landed on were as follows:

```
''PROJECTIONAL EDITING''
 OR
''PROJECTIONAL EDITOR''
```

We adjusted these search terms to fit the query syntax of the various search engines.

As most research search engines offer date ranges, we also used the date range to eliminate unnecessary papers at the automated search stage to save the effort of excluding them later. In our research question, we are specifically looking at the current state of projectional editing. A restriction of many research search engines is that they define date ranges in whole years. When designing our search strategy, it was near the beginning of 2021. We concluded that only including papers from 2021 would be too small a search space. Therefore, we set our date range to be from the beginning of 2020 to the present. For the sake of reproducibility, we advise the removal of any papers after 31st July 2021.

We show the search engines we used in Table 3.1.

ACM digital library	Google Scholar
BASE	CORE
IEEE Xplore	ISI Web of Science
Microsoft Academic	Science.gov
Wiley InterScience	SCOPUS
Semantic Scholar	SpringerLink

Table 3.1: Search engines used

Once we have filtered the automated search through the criteria of the selection stage, we will use that as our starting set for snowballing. We will do all our filtering before we do any quality assessments, as we feel that excluding papers from snowballing based on the quality of the primary study would artificially limit the network of potential papers. Our snowballing procedure shall follow the advice of Wohin[67]. Snowballing is a technique for finding related papers using the reference lists in our starting set and applying the same selection criteria.

Where possible, we will get the forward snowballing papers from the "cited by" functionality of Google Scholar. Because of the range of the search being "to present", all papers that cite the target paper will fall within our criteria. For backward snowballing, we will manually filter the bibliography section of the selected papers, selecting any paper published in 2020 or 2021

After gathering all the papers from the forward and backwards snowballing, we will apply the selection criteria again. The snowballing process will recursively iterate until there are no new papers. The papers accepted in each iteration will form the basis for the following stage - the quality assessment of the primary studies.

After the final iteration, as a final step, the selected papers will have a deeper scan. This deep scan ensures that the papers selected in our initial scan meet our inclusion criteria before moving on to the quality assessment.

3.1.4 Study Selection

The inclusion criteria are:

- Studies are about or mention projectional editing or one of its synonyms.
- The study published date is in the period 2020-2021.

The exclusion criteria are:

- Books and grey literature.
- Not in English.
- Full text unavailable.
- Papers with severe issues with grammar or vocabulary.
- A duplicated paper.
- The primary study is in a previously selected paper.

If multiple papers look at the same study with different approaches, we aggregate the data during the synthesis stage.

As a lone researcher, we must be aware of bias in positively including relevant papers and excluding irrelevant papers. We will follow Kitchenham's suggestions to overcome such bias:

- Test-retest
  - We will assess the papers once (on title abstract and keywords) against the inclusion and exclusion criteria.
  - Save all the suggested results.
  - Assess the papers again three days later in a different order to the first.
- If there are disagreements, we will use Cohen's[68] Kappa agreement statistic to see if refinement of the process is needed.

If our searches appear too large for a lone researcher, we will turn to text mining. We will be cautious about using this. O'Mara-Eves et al.'s[69] systematic review of text mining in systematic reviews recommends using this for prioritisation but finds that for exclusion screening, although promising, it is not yet proven.

An SLR is interested in studies rather than papers. There is a many-to-many relation between papers and studies. We will review the selected papers to note when a study is reported in multiple papers in our results to make sure studies do not get over or undercounted.

3.1.5 Quality of Primary Studies

To discover explanatory reasons for why there may be differences in study results and to weigh how valuable specific studies are, we will assess the quality of the selected studies.

To avoid a "Results Section bias", we will be operating a results-blind quality assessment. We base our study quality on the methods section of the papers only. However, this bias is still a threat because the abstract, which we will read, summarises the results. The study quality will not be measured until after the selection process is complete, though it will, in part, occur before the selection re-test process.

We will use checklists from the Center for Evidence-Based Management, found in Appendix B, to conduct the quality assessments. These checklists address general scientific research.

In software engineering, many studies fall under what Gregor[70], in "A Taxonomy of Theory Types in Information Systems Research", calls "Type V: Theory for Design and Action". These types of studies are also known as Design Science Research (DSR). Staron[71] describes DSR as "...a methodology which we can place close to the engineering, technical areas of software engineering. Design science is the design and investigation of artefacts in context. A design science research project, therefore, seeks to solve an empirical or industrial problem, with the help of an artefact. It recognises two contributions — the construction and evaluation of the artefact and the development of new knowledge."

The checklists do not address this type of research well. Despite investigating how other SLRs conducted quality assessments of DSR studies, we did not find a good checklist for DSRs. Therefore, we will continue with the checklists shown in the appendix, using the checklists meant for Case Studies for the DSR research papers. We will take this into account before dismissing results of this type based on their quality score.

As a lone researcher will carry out this study, there is no need to have a process for disagreements between researchers.

We use the quality assessment checklist to weed out the biases of selection, performance, detection, exclusion, and other threats to the validity of the studies under test.

3.1.6 Data Extraction

No data extraction will be necessary for the first sub-question, "Is there current research in the area of projectional editing?". The existence of papers with verified primary studies either into projectional editing theory or its practical use will be enough to answer the question.

For the question of "What tools are currently being used for research?" we shall note each tool discussed mentioned explicitly in the study.

Finally, for the question "What is the sentiment in papers currently discussing projectional editing?", we shall pass each paragraph of the introduction, the discussion, the conclusion, and any other sections that mention projectional editing or tools through a sentiment analyser, noting its sentiment score. The sentiment analysis tool we shall use is Microsoft Azure Cognitive Services Text Analytics. We show the code to carry out this task in Listing 3.1.

We will gather this data in tables with the categories shown in Table 3.2.

```
1
2       from azure.core.credentials import AzureKeyCredential
3       from azure.ai.textanalytics import TextAnalyticsClient
4
5       endpoint = "REPLACE_WITH_CORRECT_ENDPOINT"
6       key = "REPLACE_WITH_CORRECT_KEY"
7
8       text_analytics_client = TextAnalyticsClient(endpoint=endpoint, credential=AzureKeyCredential(key))
9
10      inputfiles = [[ARRAY_OF_FILES_TO_BE_ANALYSED]]
11
12      with open('/content/sample_data/sentiment/output_all.txt','a') as outf:
13      for sections in inputfiles:
14          for section in sections:
15          print("Section: {}".format(section),file=outf)
16          f = open('/content/sample_data/sentiment/'+section)
17          content = f.readlines()
18          # for brevity an optimization to deal with 10 document limit is removed
19          if len(content) != 0:
20              result = text_analytics_client.analyze_sentiment(content, show_opinion_mining=True)
21              docs = [doc for doc in result if not doc.is_error]
22              for idx, doc in enumerate(docs):
23              print("sentiment: {}".format(doc.sentiment),file=outf)
24              print("Document text: {}".format(content[idx]),file=outf)
```

Listing 3.1: Text Analytics code

#	Data Type	Description	RQ
1	Study ID	Unique identifier for the study	
2	Title of Study	The paper name	
3	Year of Publication	It will be either 2020 or 2021	
4	Author(s) Names		
5	Source of Study	Name Of Online Database/ Digital Library	
6	Type of Study	Experiment/Case Study/Survey/DSR	
7	Name of Venue	publishing Journal/Conference	
8	Tools in Study	A list of the tools used	RQ 1.2
9	Sentiment	The sentiment scores from appropriate paragraphs	RQ 1.3

Table 3.2: Data extraction form

3.1.7 Data Aggregation and Synthesis

Kitchenham[65] explained that primary studies would be too heterogeneous for any statistical analysis in software engineering. Synthesising outcomes from multiple methods will be complex. Thus, our synthesis will take a narrative approach.

Narrative synthesis tells a story of the who, how, and why of the success or otherwise of the research. For DSR research, the focus will be on what will help or hinder the adoption of the implementations. It will also examine how reliable the results are and the relationships between the studies.

3.2 Results

We carried out a systematic literature review (SLR). We summarise the results of our SLR as “undetermined”. We make this statement because the review design was not wholly appropriate for the problem domain.

We could not find a quality assessment checklist that adequately dealt with design science research (DSR) studies. This inadequacy proved problematic, as most of the primary studies were DSR studies. Therefore, what follows should be considered the results of a quasi-SLR, with the quality assessment stage ignored.

3.2.1 Papers Selected

We logged the details of what we describe in this section in Appendix A.

Figure 3.1, on the following page, shows the results of the five iterations that the search went through. Out of 173 results, we had 50 papers that provisionally seemed to pass our inclusion and exclusion criteria from our initial search. From the initial 50, we added 18 papers from a possible 109 in our first iteration of forward and backwards snowballing. The next snowballing iteration returned three papers that matched our criteria. The third round of snowballing had no papers matching our criteria and thus terminated this stage of selection.

Our final selection iteration involved a deeper scan of the remaining 71 papers. In this stage, we rejected 12 papers that were not primary research and one paper which reported on an already represented study. Further, we rejected twenty-seven papers that were, on closer reading, not about projectional editing.

This final selection filter left us with 31 papers before the quality assessment filter.

3.2.1.1 Sensitivity and Precision

As a curio, we reappropriated Zhang's[66] ideas of sensitivity and precision and applied them to the search engines rather than search strings. We calculate the values for sensitivity and precision of the search engines as follows:

$$sensitivity = \frac{\#\,retrieved\,relevant\,studies}{\#\,all\,relevant\,studies}\,100\%$$

$$precision = \frac{\#\,retrieved\,relevant\,studies}{\#\,studies\,retrieved}\,100\%$$

Table 3.3 show that Google Scholar had the highest sensitivity, returning 22 of the 31 chosen studies. This sensitivity came at the cost of a considerable proportion of false positives. Microsoft Academic and SpringerLink were the joint-most precise, with half of their search results ending up in the final roster. With the second-highest count of documents, the second-highest sensitivity, and joint highest precision, SpringerLink would appear to be the best all-around search engine for this field. However, these figures are skewed by several of their articles coming from a single collection specifically about projectional editing.

Search engine/library	original #	selected #	sensitivity	precision
ACM	18	3	10%	16%
BASE	7	3	10%	43%
CORE	7	1	3%	14%
Google Scholar	82	22	71%	27%
IEEExplores	5	2	6%	40%
Microsoft Academic	10	5	16%	50%
Science.gov	0	0	0%	0%
SCOPUS	10	3	10%	30%
Semantic Scholar	10	4	13%	40%
SpringerLink	22	11	35%	50%
Wiley Online	1	0	0%	0%
Web of Science	1	0	0%	0%

Table 3.3: Search engine sensitivity and precision

3.2.2 Quality Assessment

During this quality assessment, we discovered that two of the papers we had initially categorised as primary studies were, in fact, proposals, and thus we removed them from our analysis. Using the quality assessment checklists developed by Crombie et al.[72], shown in Appendix B, we examined the remaining 29 papers, which on the surface represented 35 primary studies.

Unfortunately, there were no checklists for DSR studies. We, unsuccessfully, searched for an appropriate quality assessment checklist for DSR studies. We did not find a suitable checklist and did not consider ourselves suitably qualified to make one. So, we used the quality assessment checklist for case studies to assess the DSR studies.

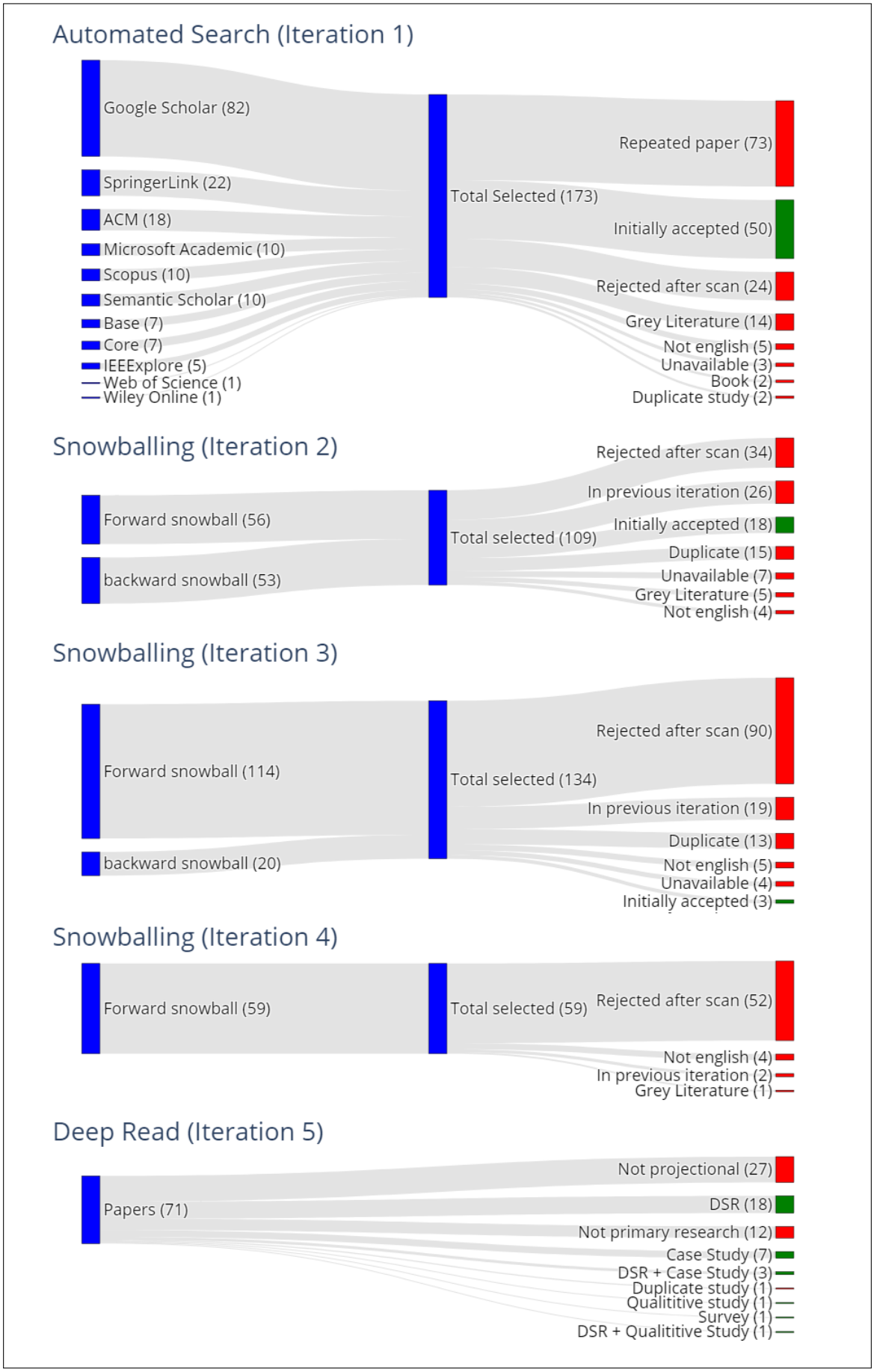


Figure 3.1: SLR search and snowballing results

We used a rudimentary scoring system of +1 value for positive answers, 0 for undetermined, and -1 for negative answers. We arbitrarily defined that any study with an overall score greater than 0 was high enough quality to be part of our final analysis.

Unfortunately, we only found 6 out of 37 studies of high enough quality to pass this filter with this scoring. Thus, we had to choose between changing our scoring, only using these six studies, terminating the SLR or ignoring the QA findings.

Changing a method until it gave the desired answer seemed unscientific to us. Six studies seemed too few to give an overview of a field. Abandoning the SLA seemed the correct course of action. However, as we still wanted an overview, we decided to take a different course. We accept that what follows is no longer an SLA. We titled it a Quasi-SLA, which is like an SLA, which ignores the quality assessment results.

We could not reconcile that 84% of studies were high-quality enough to appear in recognised scientific journals yet were not of high enough quality to pass our SLR QA stage. After considering this disconnect, we found two significant threats to the validity of the Quality Assessment stage. The first is that a single researcher with no previous experience executed the QA stage. The second is that either case study checklists are inappropriate for DSR studies or that DSR studies are inappropriate for SLRs.

3.2.3 Analysis

After Identifying the primary studies, we extracted data. Appendix D shows the forms containing the raw extracted data. Figure 3.2 shows that most of the primary studies in our review were DSR studies.

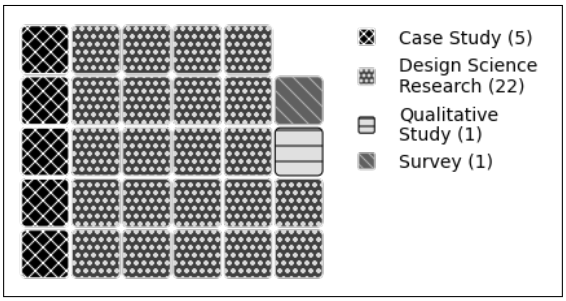


Figure 3.2: Study types

3.2.3.1 Tools Used

We split the studies to see which were to do with purely research projects and which were researching using already publicly available commercial or open-source products. To calculate this, we removed one primary study, a survey, as it covered many tools and options, but none of which was in-depth. Figure 3.3 shows that over 80% of the projects were studying already existing publicly available options.

Of the publicly available software studies, we wanted to know which software attracted the most academic interest. Figure 3.4 shows that 74% of the studies into projectional editing that used a publicly available product used JetBrains MPS.

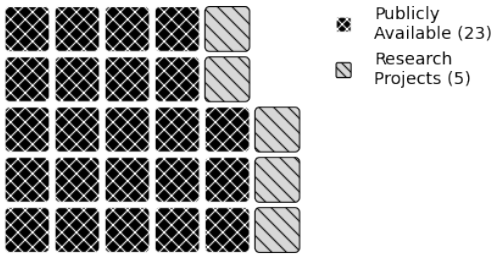


Figure 3.3: Public vs research

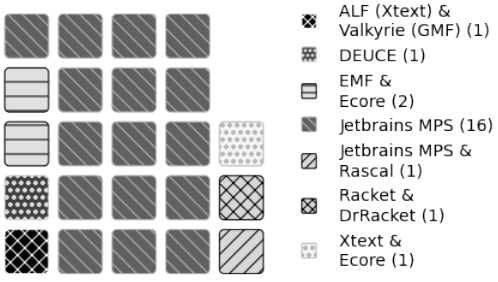


Figure 3.4: Publicly available programs

3.2.3.2 Sentiment

In this study, we included all 29 papers. We tagged each section from those papers that talked about projectional editing. We then broke each section into sentences and ran those sentences through a sentiment analyser as described in Section 3.1.6. We show the outcome of this sentiment analysis in Figure 3.5 on page 33. The charts show the relationship between the positive, neutral, and negative sentiment outcomes. On the y-axis, we have an Id for the papers examined. We show the keys linking the Ids to the paper names in Table 3.4, on page 34, the page following the charts.

The charts in the first column of Figure 3.5 show the absolute number of sentences analysed per paper partitioned by whether they returned negative, neutral, or positive sentiment results. The charts in the second column show these as percentages so that the papers are comparable. In the charts in the third column, we removed the neutral scores and calculated the percentage positive to negative. The final chart column is an aid to make it easier to scan whether papers trended positive or negative. If we found the papers to be equally positive and negative, we classified them as neutral.

Over the 29 papers, we scanned a total of 3003 sentences. Of these sentences, 435 were analysed as being positive, 1953 were neutral, and 615 were negative. Hence, 14% were positive, and 16% were negative.

10 of the 29 papers were more positive than negative when discussing projectional editing, 16 more negative and three equally negative and positive.

In Figure 3.6, we attempt to separate sentiment by product category. These categories are Research projects, MPS, and all the other used products. We ignored the survey paper in this one as it covered all these types.

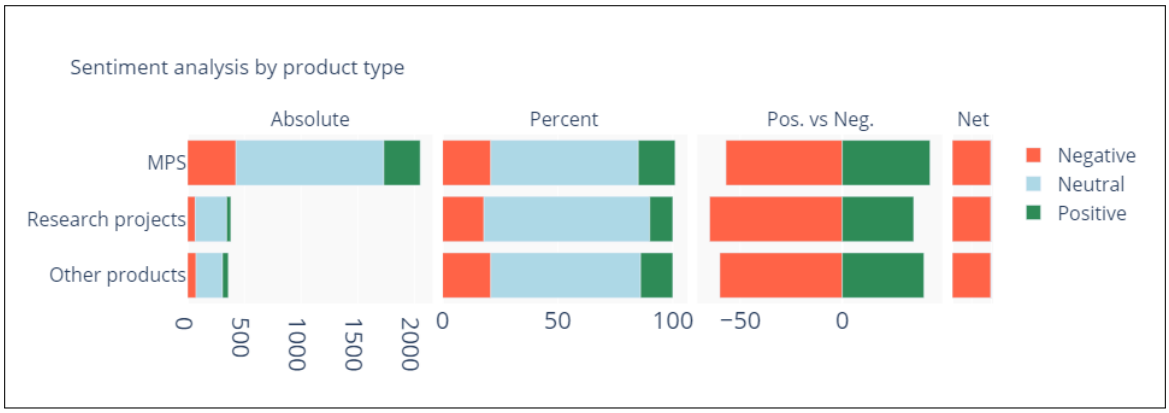


Figure 3.6: Sentiment analysis by product

MPS dominates the sentences accounting for 17 (61%) of the 28 papers and 2051 (73%) of the 2791 sentences analysed.

3.2.3.3 A Narrative Synthesis

Our synthesis of the papers that appear in Table 3.4 will be short. We will avoid rehashing the advantages and disadvantages of projectional editing, which come up again as we discussed these thoroughly in Section 2.2.3 and Section 2.2.4.

Many of these papers focus on models and model-driven development, occasionally suggesting a shift towards textual modelling languages. However, other papers point out that text does not always supply a suitable level of abstraction in modelling. One paper suggested that developers prefer text, whereas maintainers and domain experts prefer visual projections, though this suggestion was unsourced.

When authors have used solutions other than MPS, they complain about issues such as MPS being heavyweight, with much overhead. However, these authors then spend a great deal of time theorising about fixing issues in their architecture which, because of its architecture, MPS does not encounter. These issues include synchronising between various views and how grammars deal with notations.

There is a fair bit of mention of a “semi-projectional” approach, which involves parsing at the leaf node level of the AST. This approach is mainly from papers not using MPS, but also some which do. The approach, it seems, is a reaction to the difficulty in simulating the text language experience in a projectional editor. It echoes the approach the Synthesizer Generator adopted when facing this same problem in the 80s.

The projects they describe are not in industrial use. These authors suggest that projectional editing is probably best suited to helping novices learn a language.

Figure 3.5: Sentiment analysis

Id		Paper name
1	[73]	A domain-specific language for payroll calculations: A case study at DATEV
2	[74]	A framework for projectional multi-variant model editors
3	[75]	A generic projectional editor for EMF models
4	[76]	A model-driven approach towards automatic migration to microservices
5	[54]	AdaptiveVLE: An integrated framework for personalized online education using MPS JetBrains domain-specific modeling environment
6	[77]	Adding interactive visual syntax to textual code
7	[78]	Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment
8	[79]	Classification algorithms framework (CAF) to enable intelligent systems using JetBrains MPS domain-specific languages environment
9	[80]	DSL based approach for building model-driven questionnaires
10	[81]	Efficient editing in a tree-oriented projectional editor
11	[82]	Efficient generation of graphical modelviews via lazy model-to-text transformation
13	[83]	Engineering gameful applications with MPS
15	[84]	Fasten: An extensible platform to experiment with rigorous modeling of safety-critical systems
16	[26]	Gentleman: A light-weight web-based projectional editor generator
17	[85]	Integrating UML and ALF: An approach to overcome the code generation dilemma in model-driven software engineering
18	[86]	Javardise: A structured code editor for programming pedagogy in Java
19	[87]	JetBrains MPS as core DSL technology for developing professional digital printers
20	[88]	Learning data analysis with metaR
21	[89]	Migrating insurance calculation rule descriptions from Word to MPS
22	[90]	Model-based safety assessment with SYSML and component fault trees: Application and lessons learned
23	[91]	Papyrus for gamers, let's play modeling
24	[92]	Projecting textual languages
25	[93]	SpecEdit: Projectional editing for TLA+ specifications
26	[94]	Teaching language engineering using MPS
27	[95]	Teaching MPS: Experiences from industry and academia
28	[96]	Tiny structure editors for low, low prices! (generating GUIs from toString functions)
29	[97]	Towards ontology-based domain specific language for internet of things
30	[98]	Type-directed program transformations for the working functional programmer
31	[99]	What do practitioners expect from the meta-modeling tools? a survey

Table 3.4: Paper key

Those authors who use MPS primarily discuss products developed for use in an industrial setting or how best to teach projectional editing to a broader audience. MPS, when used, is often seen as a critical enabler. Two of MPS' properties that garner the most mentions are the ease of composition and multiple views.

The users of MPS agree that simulating the experience of the text editor user in the projectional environment is still very hard. However, new plug-ins are making this somewhat more manageable. The most prominent carrion call amongst the MPS users is for a web-based interface.

The steep learning curve is another issue. Several papers offer solutions to this, such as example-driven development, gamification, grammar to MPS plug-ins and something called a "language wheel".

In general, researchers using MPS have a few gripes with usability but seem to be very positive. One paper, which was not using MPS, said that some problems become intractable when dealing with graphical models. Another paper, in a coincidence of word use, when describing the decision to use MPS, explained that it was because it presented a tractable level of complexity.

3.3 Discussion

We now examine threats to the construct, the internal and external validity of our SLR, its reliability, and areas of improvement.

3.3.1 Threats to Validity

As discussed in their tertiary study of SLRs, da Silva et al.[100], one of the main problems of SLRs in Software engineering is a focus on practice and not experimentation. Because of the nature of the subject area, we will be making this same shortfall. We feel that we have fully addressed their other concerns of SLRs not assessing the quality of our primary studies, bad integration, and lack of guidelines.

As with all SLRs, the main threats to validity are an incomplete set of studies due to an insufficient search strategy, researcher bias in paper selection and inaccuracy in data extraction. Our study quality assessment used Runeson et al.'s[101] four suggested limitations of studies, namely construct validity, internal validity, external validity, and reliability. It is only fair that we point this towards our study.

3.3.1.1 Construct Validity

Regarding construct validity, i.e., whether our research questions match the research subjects' methods and measures, whilst no measurement system is perfect, some are much further from perfect than others.

For the construct to be valid, we need to present the best available evidence. The nature and modernity of projectional editing might mean plenty of good evidence is available in grey literature and industrial articles. This under-representation of actual but non-academic studies could lead to a false positive or negative for some of the questions, leading to errors in recommendations.

There may be circumstances that influence the best evidence, such as who is funding the study. Is a researcher working or consulting at a projectional editing product supplier, and will this skew results?

Are some projectional editors being ignored because of the preference for English papers only? The focus on English language papers might be biased against projectional editors aimed at non-English speaking markets.

The use of the sentiment analysis tool may have been inappropriate. Scientific papers may not lend themselves to sentiment analysis in general. Conversely, the Azure sentiment analysis service may not be appropriate for scientific papers.

3.3.1.2 Internal Validity

Internal validity, or the causal relationships, questions whether one factor causes an effect or whether both factors influenced by something unseen.

An incomplete search term may have led to selection bias. Using "projectional editing" and "projectional editor" may have led us into a small corner of this field. If one tool uses this term and others use other terms to describe a similar approach, these search terms may misrepresent the field. Other tools or projects could use different terms, such as "language orientated programming" or more antiquated terms like "structured programming" or "syntax-directed editing".

One causal relationship that could have influenced the outcome was that the book "Domain-Specific Languages in Practice with JetBrains MPS" was published right at the end of our selection period. Of the 11 papers

published in this book, 7 made it into our final paper selection. As this book is all studies involving using MPS, then this skews the data towards MPS.

We also had an error with the identification of primary studies. Initially, we took the word of the paper when it said it had done a case study, but frequently, that just meant trying their code out on a current problem rather than a case study in the academic sense.

A final threat to internal validity was that no expert in either Drools or projectional editing evaluated the conclusions we drew.

3.3.1.3 External Validity

External validity is the ability to generalise the findings.

The prevalence of the DSR methodology in software engineering is why these results are impossible to generalise. The "primary study" data in these cases are often just the source code.

Another threat to validity is whether we have a good representation of the field in our restricted timespan of the search. Would the findings be similar across a longer time frame? However, as the scope of the question was to examine the current state, we still feel that this restriction is necessary and informative.

3.3.1.4 Reliability

Reliability is how the data and analysis are dependent on specific researchers. Here we are presented with a very credible threat in that a single researcher carried out this research. Under particular threat from the single-researcher bias were the quality assessments.

Whilst measures were put in place to try and mitigate this, the reliance on a single person's judgement of the underlying studies leaves the door to bias wide open. Another threat is the use of narrative review. This review type can be subjective and, therefore, difficult to reproduce.

As mentioned in the results, we were not happy with any of the paper quality assessments for DSR studies. Using an inappropriate quality assessment tool was instrumental in the almost complete failure of the quality assessments.

3.3.1.5 Repeatability Vs Reproducibility

Greenhalgh et al.[102] in their review of where papers come from in SLRs, they found that 24% of papers come from "personal knowledge or personal contact". For the sake of reproducibility, we decided not to hunt down relevant papers from knowledgeable people in the field, as this would require anyone trying to reproduce this study to ask the same people at the same knowledge level as us.

3.3.1.6 Method Improvement

An area we would improve is sentiment analysis. Our technique was flawed. To avoid the risk of bias through us cherry picking paragraphs, we were very coarse-grained in our input selection criteria. For all papers, we chose the introduction, the conclusion and then any section that discussed anything to do with projectional editing.

The problem is that sometimes the sections would run for paragraphs, with only one or two being about projectional editing. Frequently, especially with papers involving MPS, the paper was focused on a problem and using MPS to solve it. Thus, the introductions and conclusions would occasionally barely mention projectional editing.

If we were to do this again, we would take a more fine-grained approach as we feel the cherry-picking risk is less significant than the noise from the unrelated text.

3.4 Summary

In this chapter, we presented a description of the details of the SLR we conducted. The quality assessment filter was not adequate for the task of this review. Thus, the review did not follow the methodology design completely. Looking only at papers created in the 18 months prior to conducting the review, 29 papers met our criteria.

A large majority of these papers were design science research studies. JetBrains MPS was the tool used for most of the papers. Papers using MPS focused on industrial and educational use, whereas those not using MPS mainly experimented with the projectional form.

We ran sentiment analysis on these papers. Results were inconclusive.

Two areas of improvement would be to design a better quality assessment filter and to improve the data cleaning for the sentiment analysis.

4

Implementing Projections of Drools

In this chapter, to answer the question "Which projections can we create to help developers get appropriate feedback about rules?" we undertook a design science research (DSR) approach. In Section 4.1, we describe the method of this DSR. Section 4.2 examines the outcomes of our research. Finally, in Section 4.3, we discuss the threats to the validity of our approach.

4.1 Method - Drools in MPS

Even though Drools is a relatively small DSL, we did not need to implement all the functionality to answer our questions. For this reason, we first created a pilot study language to assess the possibilities of projectional solutions to our question. We finally created a larger but incomplete version of Drools, in which we could create projections that would be recognisable to experienced Drools users.

4.1.1 Really Simple Drools Language

Our pilot study created a simple approximation of the Drools language to create our first projections. We called this language "Really Simple Drools" (RSD). We describe this pilot study language here, as it contains many of the projections that we considered for our research question.

4.1.1.1 Concepts

`File` RSD, like Drools itself, has a `File` Concept as its root node. The `File` Concept only contains `FactDeclaration` nodes and `Rule` nodes.

`FactDeclaration` and `FactProperty` In Drools, a `FactDeclaration` Concept represents a Java Bean with its child properties, which can also have their child properties, ad infinitum. In RSD, we limited properties to allow only boolean values. We decided this because fact selection is a predicate and thus can only return a boolean. By only allowing boolean values, we also simplify the operations allowed on a `FactProperty` node.

`Rule` We only simulated the Left-hand side, or the "when" conditions, of a Drools Rule for the `Rule` Concept. We believed this would provide us with compelling options for projections and did not want to overcomplicate this pilot project.

An RSD Rule consists of a collection of conditions. Should all those conditions return "true", then the `Rule` node is selected.

`AbstractCondition` A condition operates on one or more `FactSelectors` nodes. There are four condition Concepts - `ExistsCondition`, `NotCondition`, `AndCondition`, and `OrCondition`. The `ExistsCondition` and `NotCondition` Concepts are unary conditions and evaluate one `FactSelector` node. The `AndCondition` and `OrCondition` Concepts evaluate two `FactSelector` nodes.

`FactSelector` A `FactSelector` Concept consists of a reference to a `FactDeclaration` node and a collection of `AbstractPredicate` nodes. If the `FactDeclaration` node exists and all the predicates evaluate to `true`, then the `FactSelector` evaluates to `true`.

`AbstractPredicate` The predicate is an operation on a `FactProperty` node, to which the Concept has a reference. Because `FactProperty` nodes represent a boolean value, the only predicate operations are “And”, “Or”, “Is”, and “Not”.

We realised this design in MPS. As the aim was to attempt different projections, we did not initially optimise for editing. The Structure is as shown in Figure 4.1. We have here translated our Conceptual design of the RSD into MPS Concepts. There is an almost one to one relationship with our concept hierarchy diagram. The one difference is that the references in the concept hierarchy diagram, represented by the dashed red lines, are represented in our MPS Structure by SmartRef Concepts `FactDeclarationSmartRef` and `FactPropertySmartRef`.

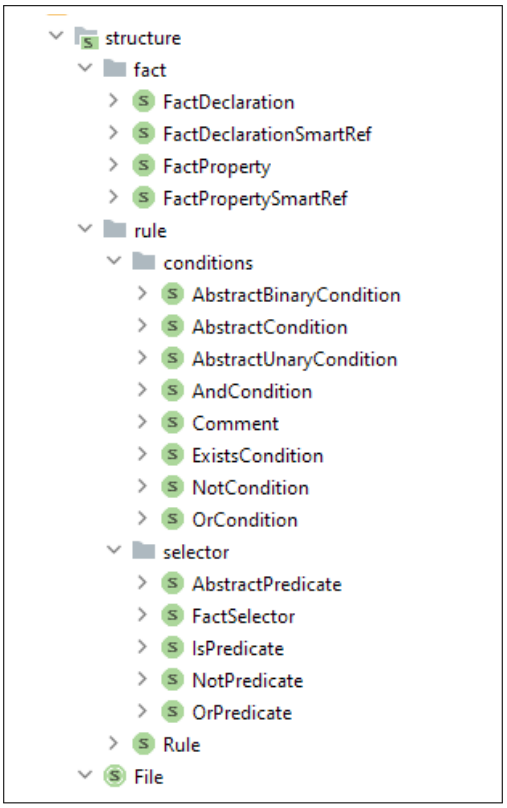


Figure 4.1: RSD Language Structure

Figure 4.2 shows the Concept hierarchy for this straightforward implementation.

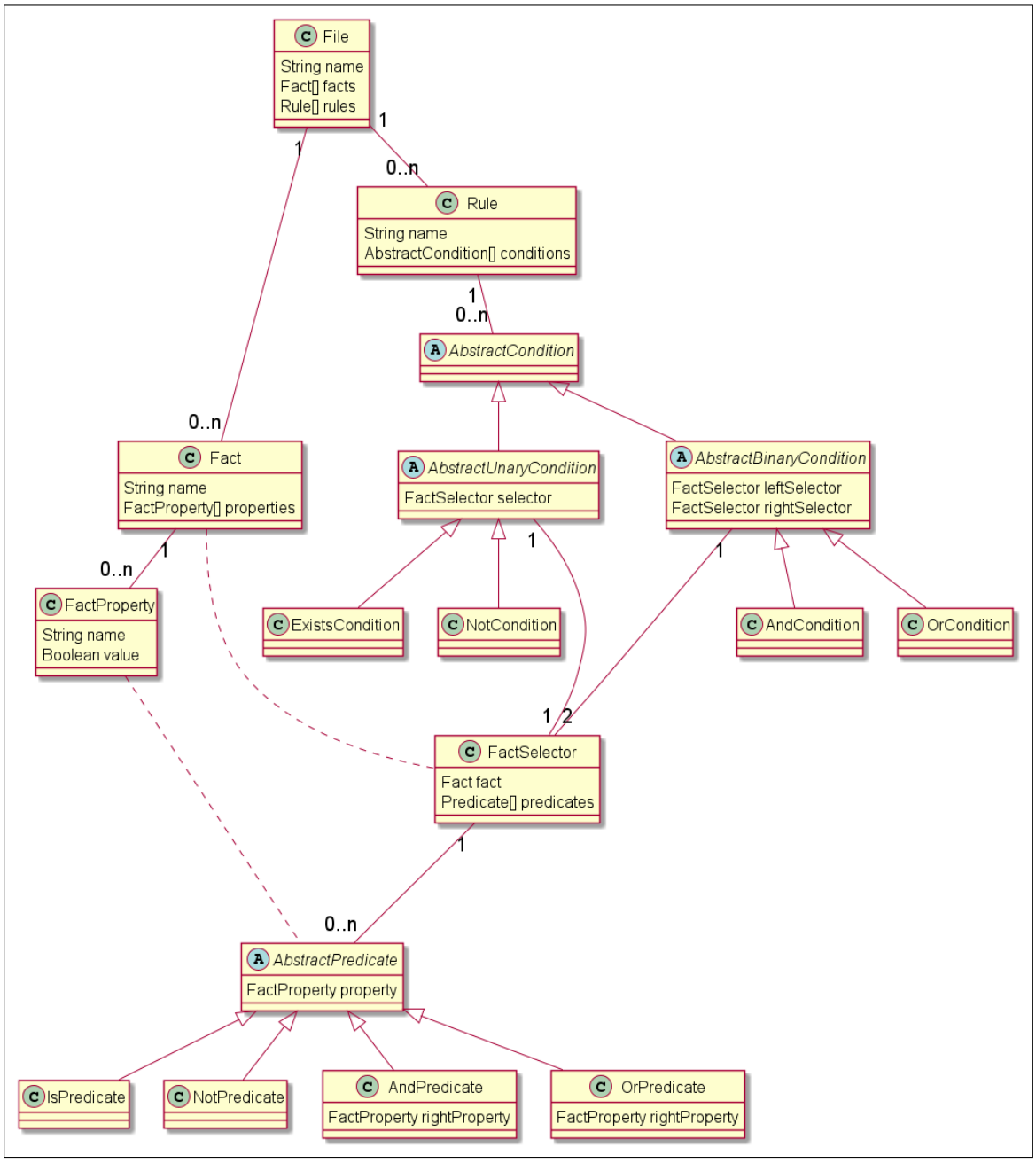


Figure 4.2: RSD concept hierarchy

We realised this design in MPS. As the aim was to attempt different projections, we did not initially optimise for editing. The Structure is as shown in Figure 4.1 on page 39. We have here translated our Conceptual design of the RSD into MPS Concepts. There is an almost one to one relationship with our concept hierarchy diagram. The one difference is that the references in the concept hierarchy diagram, represented by the dashed red lines, are represented in our MPS Structure by SmartRef Concepts `FactDeclarationSmartRef` and `FactPropertySmartRef`.

4.1.1.2 Editors

We show the definitions of the editors in Figure 4.3 on page 41. The first editor describes the `File` Concept. It shows that the first line will have the text "rule file name: " followed by the file's name. Following an empty line, there is a vertical listing of the `FactDeclaration` nodes stored in the "facts" child of the `File` node. Thereafter is another empty line followed by a vertical listing of the `Rule` nodes stored in the "rules" child.

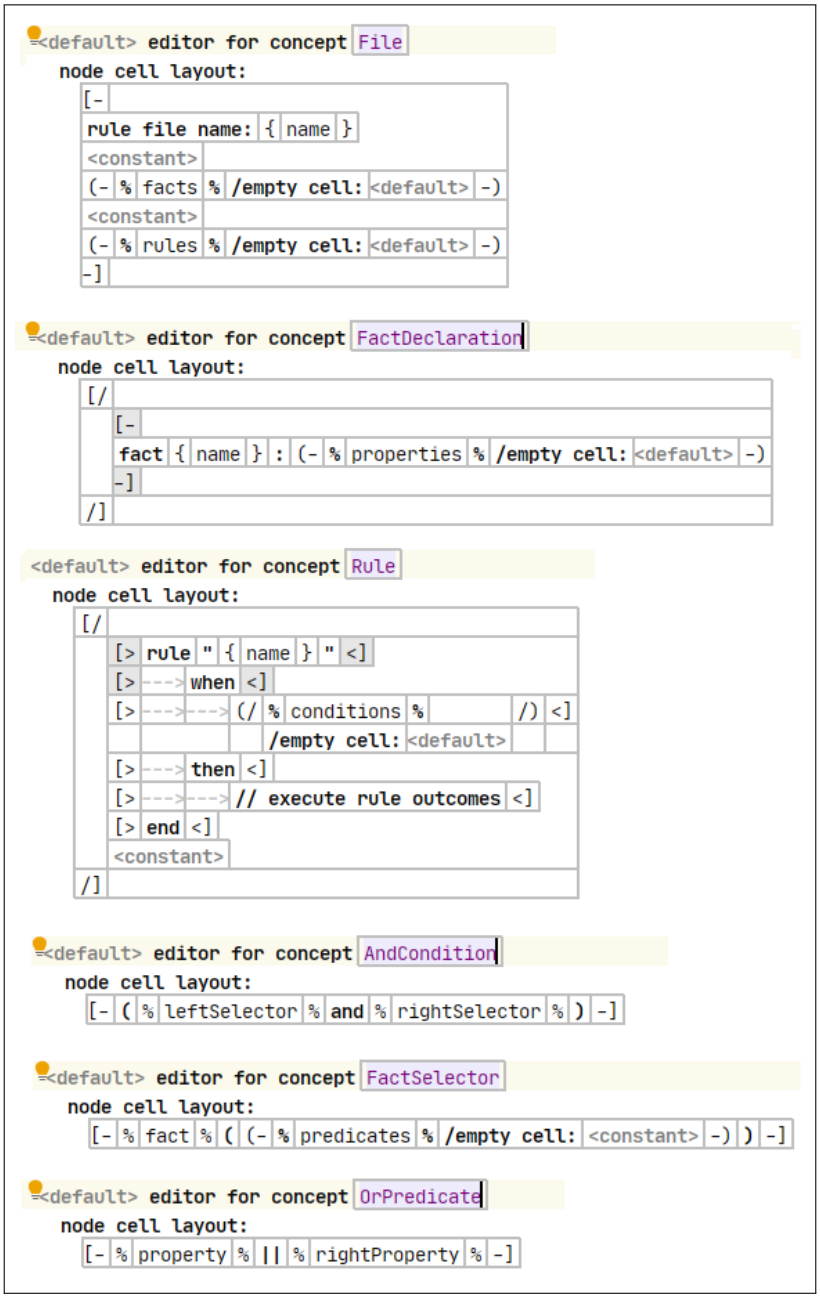


Figure 4.3: Editors

The next editor shown is the editor of the `FactDeclaration` Concept. Each `FactDeclaration` node will be shown on a single line starting with the word "fact" followed by its name. Then, between parentheses, a horizontal, comma-separated list of its child `FactProperty` nodes.

The subsequent editor describes the `Rule` Concept. The layout it describes is similar to that of a Drools rule, with the first line being "rule" followed by the `Rule` node name in inverted commas. There is a hardcoded indented "when", followed by a further indented vertical list of the condition children, which are `AbstractCondition` nodes. At appropriate indentation levels, the conditions precede three hardcoded lines of "then", a place holder for the right-hand side of a standard Drools rule, and "end".

Next, we show one of the `AbstractCondition` Concepts, in this case, the `AndCondition` Concept. This Concept editor displays the two `FactSelector` child nodes separated by "and" and surrounded by parentheses.

The editor for the `FactSelector` Concept displays the fact child, which is a `FactDeclarationSmartRef` node, for which we do not show its editor here, but in this case, shows the name of the `FactSelector` node that it is referencing. The name precedes a horizontal, comma-separated list of the `AbstractPredicate` nodes stored in the predicate's children, encased in parentheses.

The final editor we show is an example of an `AbstractPredicate` Concept, specifically an `OrPredicate` Concept, which shows the two properties separated by a "||" symbol. The properties are `FactPropertySmartRef` nodes, and they will just display the name of the `FactProperty` node to which they point.

4.1.1.3 The Language

We show the result of the editors in Figure 4.4, which shows an example of our default Drools-like text projection. This projection is of the root AST of a `File` node called "DossierSleutelbos". It has 4 `FactDeclaration` child nodes and 3 `Rule` child nodes. The `FactDeclaration` nodes called "Dossier", "Episode", and "Milestone" each have two child `FactProperty` nodes, whilst "DroolsContext" has none.

The `Rule` node "0" has 2 `ExistsCondition` nodes containing `FactSelector` node children with `FactDeclarationSmartRef` nodes referencing the `FactDeclaration` nodes "Dossier" and "DroolsContext".

The `Rule` node "[WVGGZ/CM] start CM Procedure" points to the "Dossier" `FactDeclaration` node, as well as having two predicates. The first predicate is an `IsPredicate` node with a `FactPropertySmartRef` node pointing to the "isWvGGZ" `FactProperty` node. The other predicate is a `NotPredicate` node pointing to the "hasRunningEpisode" `FactProperty` node.

The final `Rule` node has a complex nesting of `AbstractCondition` nodes. The first is an `OrCondition` node containing an `ExistsCondition` node on its left-hand side and an `AndCondition` node on the right. The right-hand side node contains an `ExistsCondition` node on both the left and right sides.

```
rule file name: DossierSleutelbos

fact Dossier : hasRunningEpisode , isWvGGZ
fact DroolsContext : << ... >>
fact Episode : isCM , done ,
fact Milestone : cmDecisionStep , finished ,

rule "0"
  when
    Dossier (  )
    DroolsContext (  )
  then
    // nothing
end

rule "[WVGGZ/CM] Start CM procedure"  "
  when
    Dossier ( isWvGGZ , !hasRunningEpisode  )
  then
    // nothing
end

rule "[WVGGZ/VCM] dossier_Start_VCM  "
  when
    ( Dossier ( isWvGGZ , !hasRunningEpisode  ) or
      ( Episode ( isCM , !done  ) and
        Milestone ( cmDecisionStep , finished  ) ) )
  then
    // nothing
end
```

Figure 4.4: RSD program

4.1.1.4 Editing aids

Part of our research question is using projections for reasoning about large files. To answer this, we needed to simulate a large file. To do this, we had to enter many `Rule` nodes. As this becomes tedious, we added some editing aids, including substitute menus, to speed up the entry of Conditions, as shown in Figure 4.5 on page 44.

This image shows that we originally had to select an `ExistsCondition` Concept and select the `FactDeclaration` node for the Condition. After adding the substitute menu, we could immediately select the `FactDeclaration` node we wanted, and it would automatically wrap it with an `ExistsCondition` node.

We show the code to do this in the section “wrap substitute menu”. What it does is when it encounters the possibility to input an `ExistsCondition` node in a menu, it replaces it with the default menu for the `FactDeclarationSmartRef` Concept. MPS handles SmartRefs in menus by showing all the possible reference items available. When the user picks the reference item, this code will create a new `ExistsCondition` node with a new `FactSelector` node containing the chosen `FactDeclarationSmartRef` node to insert at that point in the AST. This substitute menu saves several keystrokes, as otherwise, we manually first had to insert an `ExistsCondition` node followed by a `FactSelector` node before filling the `FactDeclaration` child.

Finally, we added a Constraint to scope the `FactProperty` references in `Predicate` Concept to the `FactDeclaration` reference chosen in the `FactSelector` node. This scope Constraint made it much easier to select `FactProperty` nodes in a `Predicate` node, as indicated in Figure 4.6, on page 45.

The figure shows that before adding the scoping constraint, it showed a list with dozens of potential `FactProperty` nodes representing all the `FactProperty` nodes in the model. After adding the constraint, it only shows the two `FactProperty` references associated with the `FactDeclaration` referenced in the `FactSelector`. We achieve this scoping in the code in the scope method. The code first finds the relevant `FactDeclaration` node from the `FactSelector` node we are at in the AST. From this, it returns a list of properties as a `Scope`.

Thus, we have described the entire implementation of the Really Simple Drools Language.

After implementing the language, we wrote a program with many rules. This program on which we will

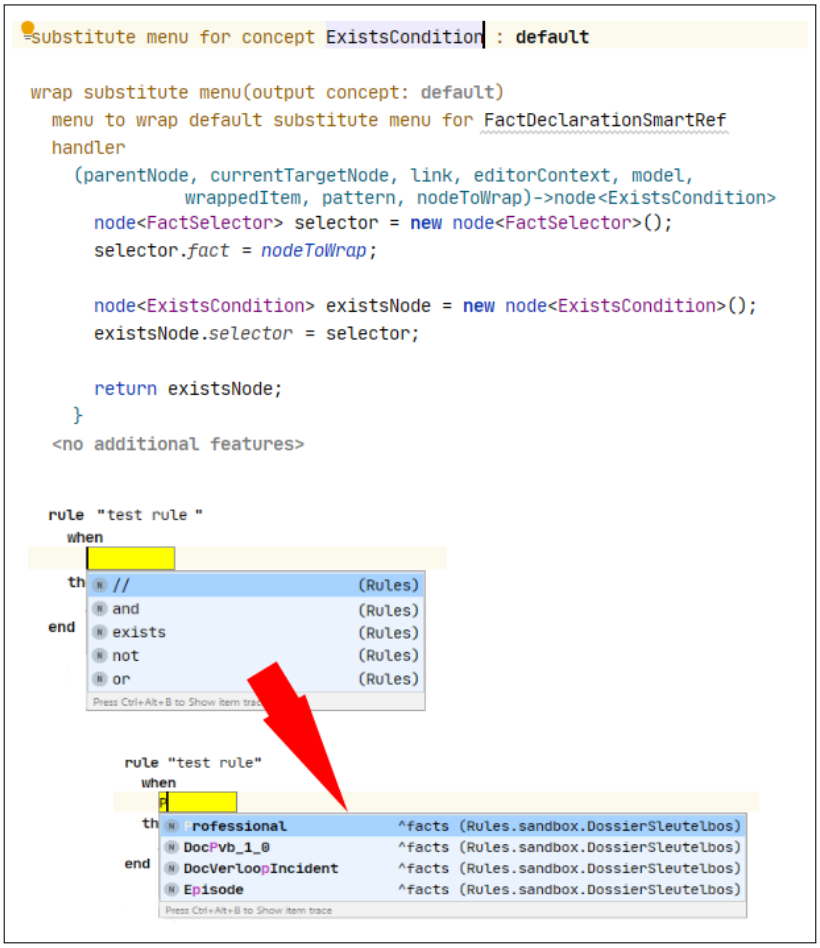


Figure 4.5: RSD substitute menu

experiment with the different projections.

We discuss the alternative projections in the results Section 4.2.

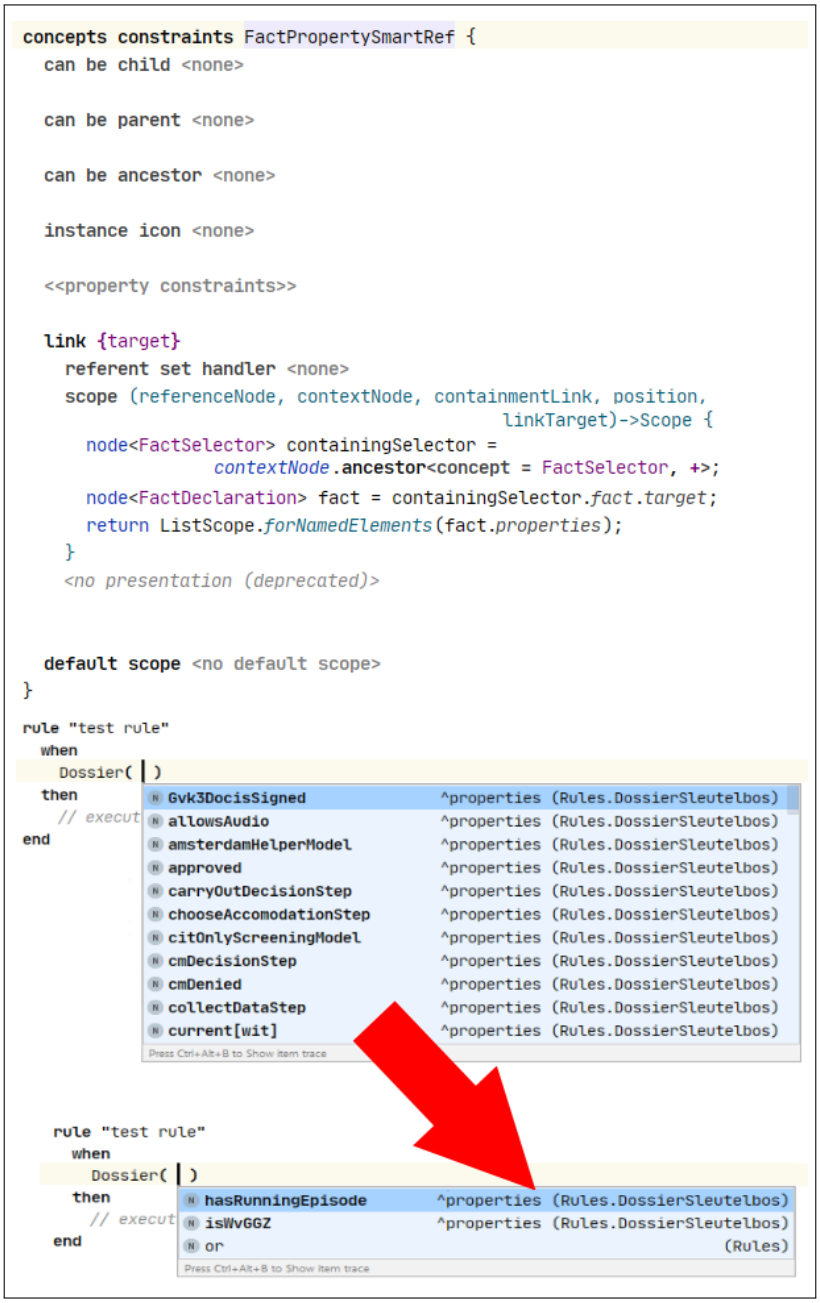


Figure 4.6: RSD scoping constraint

4.1.2 Drools-Lite Language

The RSD served as a useful pilot to demonstrate that different projections could be useful. However, it suffered from two significant issues. Firstly, its limitations as a language were so substantial that it could not handle many necessary scenarios. Secondly, developers with Drools experience will validate our projections. Thus, as RSD would be too alien to them, it would be impractical to use for validation. For this reason, we needed to create a projectional language that was much closer to the Drools language.

Our following Language, Drools-Lite, contains many more of the features of Drools. Our method of selecting the features involved implementing the examples delivered with Drools (including the corrupt politician example shown in Section 2.1.3). We would implement just enough features to complete the examples. Whenever we had any queries about designing Concepts, we referred to our analysis of the Drools Language, shown in Appendix E. We show the preliminary design we achieved using this method in Figure 4.7 on page 47. Later, there were some places we diverged a little from our design. We merged and decoupled our Concepts when we thought it would simplify the code.

4.1.2.1 Concepts

`RuleFile` The `RuleFile` Concept contains `FactDeclaration` nodes, `Global` nodes and `Rule` nodes. It also contains semantically unimportant empty lines.

FactDeclaration A `FactDeclaration` Concept has a "type" property. We implement the "type" property using a `ClassifierType` node from the MPS BaseLanguage. This implementation allows a `RuleFile` node to refer to BaseLanguage classes implemented in the same solution or from Java JAR files.

We created a smart reference Concept for this to take advantage of built-in MPS UI functionality. A smart reference is a node with a single reference of 1:1 cardinality. The editor builders know how to select which nodes are in scope to display to the developer if one uses this object rather than directly referencing the node it refers to.

FactProperty In RSD, we had `FactProperty` nodes as children of `FactDeclaration` nodes. Now that our `FactDeclaration` nodes refer to actual classes (`ClassifierType`), our `FactProperty` Concept should reflect this. To do this, the Concept itself only references an `InstanceMethodDeclaration`, the MPS BaseLanguage's definition of a method signature. We scoped the Concept to only show properties associated with a selected `FactDeclaration`.

Drools interacts with Java objects as if they are Java Beans. To simulate this, we limited the scope of the properties to just getters, i.e., methods that start with "get" or "is", and used a Behavior to display them without the "get" or "is" prefix. We also made a smart reference for this Concept.

Another option for achieving this is to have wrapped the `ClassifierType` Concept and referenced its related `InstanceMethodDeclarations`. We would have then had to limit the functionality of these items from the BaseLanguage. Whilst this allows the functionality we wished for, we feel our construction offers decoupling and that, we think, correctly reflects the structure of the language. Perhaps if we were to redo this, we would have taken the other approach.

`Global` Our `Global` Concepts are very straightforward. They have a "name" property and a BaseLanguage "type" child node. We added a smart reference so that `Rule` nodes can easily use them. The reference extended the `Expression` Concept from the BaseLanguage. This extension is so that we could use it in the Java code of the Right-hand side.

`Rule` Our `Rule` Concept has three children: an `AttributeCollection` node, a Right-hand side node and a list of `AbstractConditions` that make up the Left-hand side. We created a component to describe the `Rule` editor for reuse, as we imagined that we would wrap this in other projections.

`RuleVariables` The `FactDeclaration` node referenced by a `FactSelector` node and the `FactProperty` referenced by a `FieldConstraint` node can be bound to `RuleVariable` nodes. `RuleVariable` nodes are scoped to a `Rule` node. A `RuleVariable` Concept has only a `name` property and a `Type` child node. We also create a smart reference for it so that it can be used elsewhere within the `Rule`. Like the `Global` Concept, it extends BaseLanguage's `Expression` to be available in the Java code of the Right-hand side.

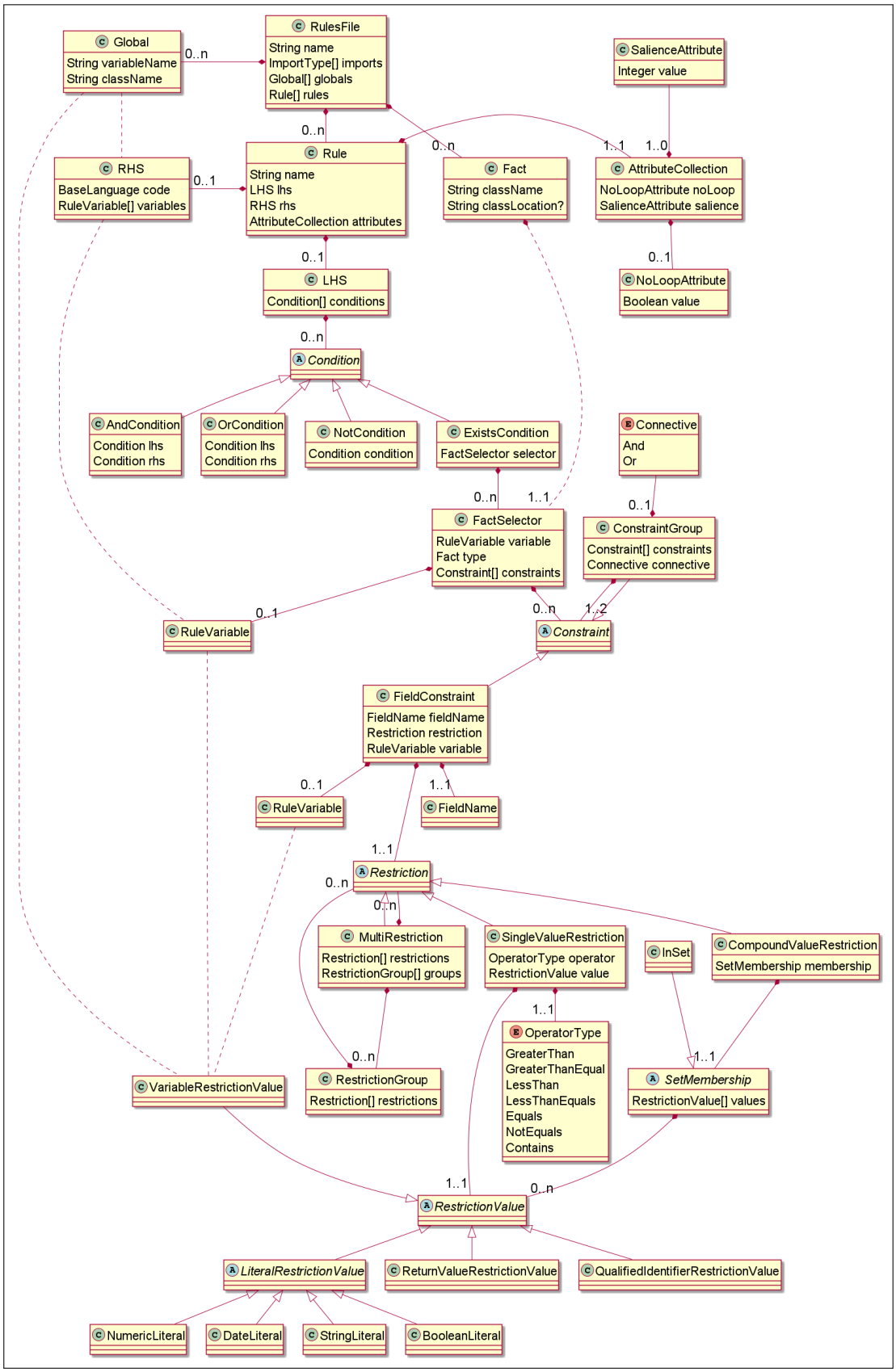


Figure 4.7: Drools-Lite structure

Right-hand side The right-hand side of the `Rule`, for the most part, is Java code. To implement this, we made the right-hand side of the `Rule` Concept a single `StatementList` node. A `StatementList` Concept is a list of `Statement` nodes, both from the BaseLanguage. We chose these because they keep track of, amongst other things, the scope of variables among the statements.

There are some non-Java, Drools specific items that are available to the right-hand side. Items that had to be useable within the right-hand side were `Global` references, `RuleVariable` references and Drools specific functions. These all extend the `Expression` Concept from the BaseLanguage. This extension allows seamless integration with the Java code.

The `Insert`, `InsertLogical`, `Modify`, `Delete` and `Halt` Concepts represent the Drools specific required methods.

```
rule "set up"
  when
    $s : Student(  )
  then
    modify( $s ) { setCumlaude( false ) };
    Program program = $s.getProgram();
    insert(  program ) ;

    foreach course in program.getCourses() {
      insert(  course ) ;
    }
end
```

Figure 4.8: RHS

Figure 4.8 shows some of the features discussed for the right-hand side, as shown in our default projection. The right-hand side is the text shown between the "then" keyword and the "end" keyword. The figure shows examples of plain Java code, such as assigning to the variable "program" and the "foreach" loop. We can also see that Drools-Lite `RuleVariable` node "$s" is in the Java statements. We have also highlighted the Drools specific methods placed in the code. In this case, the **`modify`** and **`insert`** methods.

```
1 rule "Rule #1"
2   salience 10
3   when
4     Program( faculty in ( Faculty.Law, Faculty.FNWI ) )
5     $s : Student( avg >= 8 ) || not Result( grade <= 7 && > 8 )
6   then
7     modify( $s ) { setCumlaude( true ) };
8 end
```

Figure 4.9: Rule

`AttributeCollection` The `AttributeCollection` Concept is a container to hold all the attributes that apply to a `Rule` node. Initially, we have only implemented the `NoLoopAttribute` and `SalienceAttribute` Concepts. A developer activates these attributes using two intentions we added to the `Rule` Concept. On line 2 in Figure 4.9, we can see an example of a `SalienceAttribute` node added to a `Rule` on line 2. [1]

Left-hand side This is a collection of `AbstractCondition` nodes. There are four types of `AbstractCondition` Concepts. `AndCondition`, `OrCondition`, `NotCondition` and `ExistsCondition` Concept. `AndCondition`, `OrCondition`,

[1] In Figure 4.9, we added line numbers to this figure to make it easier to talk about. The keywords "rule" on line 1, "when" on line 3, "then" on line 6, and "end" on line 8 have no meaning in the abstract syntax. We added them to give the developer the same look and feel as a standard Drools file.

and NotCondition have one or two children who are also AbstractCondition nodes. The ExistsCondition Concept contains a FactSelector node.

We added dynamic braces to only show braces around a Condition if it is another Condition's child. These braces add visual clarity without adding unnecessary clutter. We also added some intentions to make it easy to switch between ExistsCondition and NotCondition nodes.

On line 4 in Figure 4.9, the whole line represents an ExistsCondition node. Line 5 shows an OrCondition node containing an ExistsCondition node and a NotCondition node. The default editor, through an intention, can make the ExistsCondition visibly explicit with an "exists" keyword. However, the standard practice with Drools developers is to make this implicit, so this is how we show it here.

FactSelector This always has a reference to a FactDeclaration node. These are references to the FactDeclaration nodes named "Program" in line 4 of Figure 4.9 and "Student" and "Result" from line 5.

Optionally, the FactSelector node can be bound to a variable. In Figure 4.9, line 5, the FactSelector node referencing the FactDeclaration named "Student" is bound to the RuleVariable node named "$s".

The FactSelector Concept also contains a list of constraints on FactProperty references, all of which must return true for a FactSelector node to return true.

Constraints We have three types of AbstractConstraint Concepts. AndConstraint and OrConstraint Concepts contain other child constraints. The FieldConstraints Concept places restrictions on FactProperty references.

FieldConstraints A FieldConstraint Concept refers to a FactProperty node and can be bound to a variable. It also has a restriction applied to that FactProperty node. Using a substitute menu, we wrapped the FactPropertySmartRef Concept. This substitution automatically creates a FieldConstraint node from the FactProperty node selection by the developer.

There are several types of restrictions and several types of values that they can restrict.

RestrictionValues The AbstractRestrictionValue Concepts that a FactProperty node can be compared with are as follows:

- LiteralRestrictions: These are Integer, Float, String, DateTime and Boolean.
- VariableRestrictions: These can be Global references, RuleVariable references referring to FactDeclaration references from the FactSelector node, or a RuleVariable node from other FieldConstraint nodes.
- ReturnValue: This compares to anything expressed as an Expression, which includes referring to constants or values behind qualified identifiers.

In Figure 4.9, on line 4, we have the return values "Faculty.Law" and "Faculty.FNWI". On line 5, the literal values "7" and "8".

Restrictions A SingleValueRestriction Concept compares a FactProperty node against a value. A MultiRestriction Concept compares a FactProperty node against multiple values, not necessarily using the same comparison for each value. A SetMembership restriction Concept checks if a FactProperty node is a member or not a member of a group.

In Figure 4.9 on line 4 a SetMembership restriction node is shown with the "in (Faculty.Law, Faculty.FNWI)" text. Line 5 in the first FactSelector node is the SingleValue restriction node represented by "avg >= 8". The second FactSelector node shows a MultiRestriction node using "grade <= 7 && > 8".

Thus, we have described the pertinent implementation details of the Drools-Lite language.

4.1.3 Wireframes

There are some potential projections we have conceived for which there is not sufficient time to implement. We want Drools experts to assess these and thus would like them to appear as realistic as possible to the assessors.

Our solution to this conundrum is to develop these presentations in a wireframing tool. The wireframe tool we chose was Axure[103]. We chose this because we had previous experience with the product. Also, it is available to students for free.

We settled on two possible projectional programming aids: Truth table and circuit diagram. We will discuss these in more detail in the results section.

4.2 Results

4.2.1 Really Simple Drools

The Really Simple Drools Language (RSD) pilot language gave us a training ground for our new projections.

4.2.1.1 Context-Aware Colour Scheme

After the default text projection, the first projection we made was giving the text a colour scheme. This form of augmentation in IDEs is probably the most basic that we see. Available in structured editors since the 1980s[104], syntax highlighting displays text in various colours and fonts according to the meaning of the terms. Syntax highlighting is helpful for the comprehension of code, at least for small code bases[105].

Developers at our host organisation use Eclipse or IntelliJ Community Editions to edit code, neither of which has syntax highlighting for Drools. Thus, the addition of this feature would immediately benefit them. However, IntelliJ IDEA, the paid version, already provides this feature for Drools. We extended the colour scheme to indicate whether the selection is looking for a positive or negative match to offer another visual augmentation that we considered valuable. Figure 4.10 shows this projection.

Figure 4.10: Context aware colour scheme

`FactDeclaration` references contained by `NotCondition` nodes and `FactProperty` references that are part of a `NotPredicate` node appear highlighted in Red. `ExistCondition` and `IsPredicate` nodes have their content coloured green. We did not test whether this improved understanding.

4.2.1.2 Summary Projection

Our next projection allows developers to have a quick overview of the `Rule` nodes and the complexity of those nodes. Figure 4.11 shows that the developers can get an overview of both the number of `Rule` nodes and the number of `FactDeclaration` references in each of the `Rule` nodes.

The building of this projection only required adjusting two editors. The `Rule` node count and `FactDeclaration` reference count were added to the `File` Concept editor using Read-Only Model Access to count the descendants of the `File` node that are `Rule` nodes and `FactDeclaration` references. The `Rule` Concept editor

```
rule file name: DossierSleutelbos  rule count 152  fact count 357

0 : # facts:  2
Dossierdetails kunnen inzien : # facts:  1
Dossiers samenvoegen : # facts:  0
[*] dossier_NieuweNotitie : # facts:  1
[*] dossier_StuurVeiligeMail : # facts:  1
Dossier verwijderen : # facts:  1
[HV ] dossier_Sluiten : # facts:  1
[HV ] dossier_Afdoen : # facts:  1
[HV ] dossier_Actualiseren : # facts:  1
[WVGGZ/VO] Bewerk NAW in WVGGZ VO voor meldmedewerkers : # facts:  3
[WVGGZ/IBS_WZD] Bewerk NAW in WVGGZ IBS_WZD : # facts:  1
[WVGGZ/CM] Start CM procedure" : # facts:  1
[WVGGZ/CM] Verzamel Stuurgegevens : # facts:  2
[WVGGZ/CM] dossier_Act_Opstellen_CMMedischeVerklaring : # facts:  2
[WVGGZ/CM] dossier_Act_Onderteken_MedischeVerklaring : # facts:  3
[WVGGZ/CM] dossier_TerugzettenAfgerondeCMMedischeVerklaring : # facts:  5
```

Figure 4.11: Summary projection

was adjusted only to show the `Rule` nodes title and, again using the model access, the count of the descendants of the `Rule` that were `FactDeclaration` references.

Whilst this may look like a report that any language workbench could create, the `File` node name and the names of the `Rule` nodes are editable in this projection.

4.2.1.3 Filtering

Whilst investigating how to handle extensive collections of rules, we looked to domains that already handle extensive collections of items. The domain of data analysis has a long history of handling large volumes. Among their two most used tools for exploration are sorting and filtering.

The nature of business rules lends them to some projectional options that would not make sense with other programming styles. Because of the independent nature of the rules, filtering lends itself to the business rules style. The semantic meaning of the order of business rules means we did not find a good use case for sorting rules. So, we decided to implement a filtering projection.

Whilst filtering occurs in other places in the coding pipeline, such as deciding on what code completion to present[106] and version control visualisation[107], we were unable to find any research on applying filtering directly to code files. Consequently, we think what we present here is an original idea.

`Rule` nodes that use the same `FactDeclaration` references or `FactProperty` references are likely to be related. Thus, these seemed the obvious items to filter. We created a projection where if the developer filtered by a `FactDeclaration` or a `FactProperty`, the projection would filter out all `Rule` nodes that did not contain references to the nodes. Once the `Rule` nodes were filtered, the projection only shows `FactDeclaration` nodes and `FactProperty` nodes that are referenced by those `Rule` nodes.

In our implementation, shown in Figure 4.12 on the following page, we show three places where we use intentions to filter the code. The first is an intention associated with a `FactDeclaration` node. We show the outcome of choosing this filter on the righthand side of Figure 4.12. The second intention is on a `FactProperty` node. As the `FactProperty` node is a child of a `FactDeclaration` node, we see this intention and the previously described one. The third highlighted intention is on a `FactProperty` reference. It also shows an intention associated with a `FactDeclaration` reference in the `FactSelector` node that holds the `FactPropertyReference` as a child.

One of our guidelines was, as much as possible, to build our projections as separate languages, non-invasively extending RSD. In our first approach at the filtering, we failed on this count by invasively adding properties to the `FactDeclaration` and `FactProperty` Concepts of the RSD to determine whether they were visible.

Our following approach created subclasses of the `FactDeclaration`, `FactProperty` and `File` Concepts. This approach, however, requires running a macro on the code file to migrate `FactDeclaration`, `FactProperty` and `File` nodes to `FilteredFact`, `FilteredFactProperty`, and `FilteredFile` nodes. This migration means that the `FilteredFile` could now only be used by languages that extend our new filtered language.

Our final approach was to add a `Filter` Concept, reference the filtered nodes, and have the editors make the

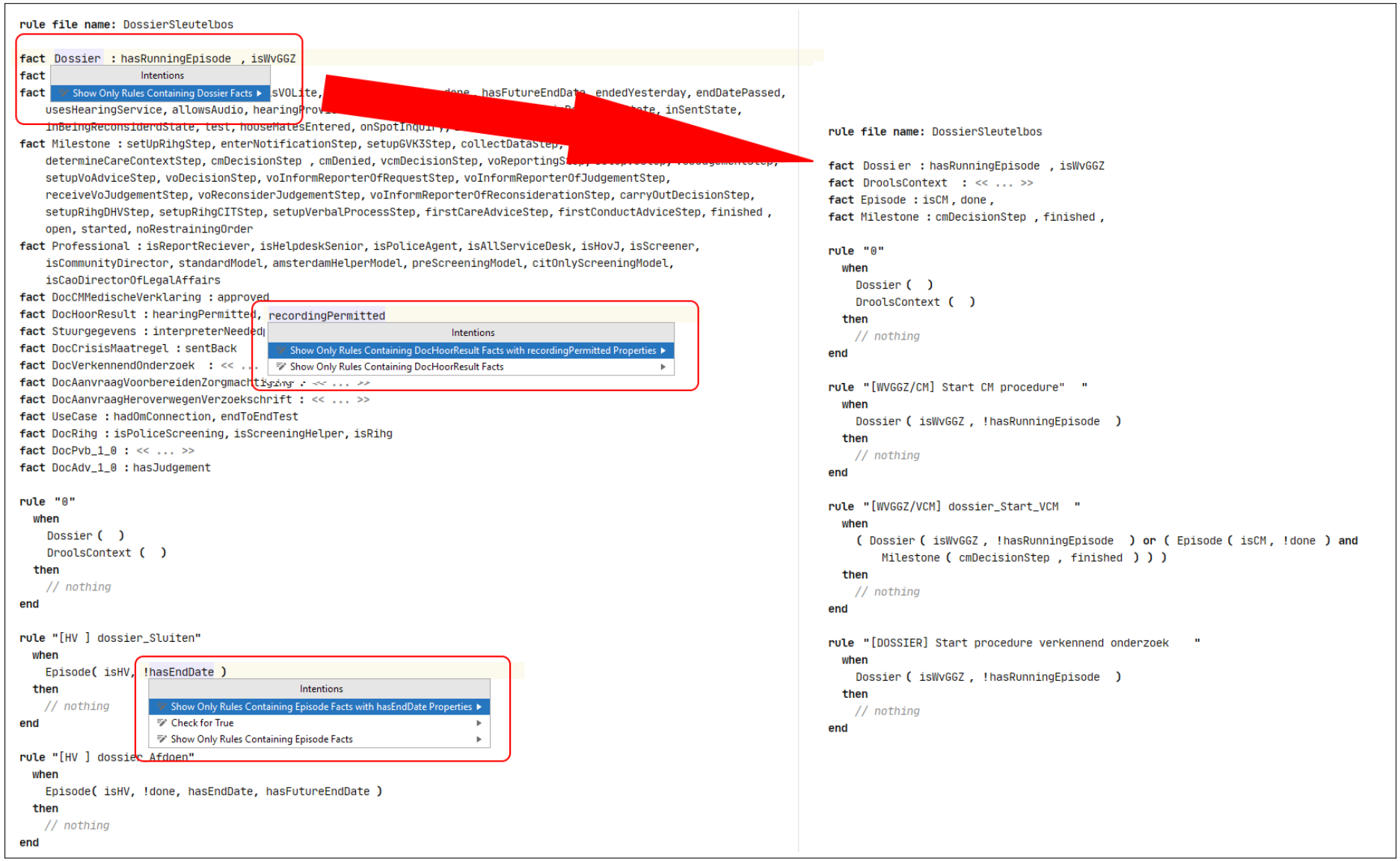


Figure 4.12: Filtering projection

visibility calculations based on this singleton node. Whilst more complex, this removed the need for invasive changes and allowed other languages to combine with the filtering language.

Filtering is a handy projection. However, it breaks Dijkstra's rule "the purpose of abstraction is not to be vague but to create a new semantic level in which one can be absolutely precise."[108]. This projection fails this rule by hiding some of the meaning of the code. This projection has no way of containing the whole code whilst a filter is applied. However, we feel that so long as there is a clear indication that a filter is applied, then we see this as a tool in a similar vein to the code collapsing functionality found in most modern-day editors.

4.2.1.4 Table

Thus far, our projections have been textual ones that other non-projectional language workbenches could implement. Creating a table was our first non-parsable projection.

We chose the table projection based on the observations of Miller[2] about the number of items people can retain in their memory. This observation leads us to conclude that the fewer essential items that are off the screen and, therefore, in the developers' memory, the better.

Figure 4.13 on page 54 shows our rudimentary first table. This simple table has only the "name" property and the "when" children of the `Rule` nodes in the `File` node. We implemented this projection using the tables extension in the MPS-Extension plug-in, created by Sascha Lißon.

4.2.1.5 Crosstab

Our next tabular projection is a crosstab inspired by a decision table. Figure 4.14, on page 55, shows our implementation of the crosstab. Because of the large amount of screen real estate that this projection uses, we only present a truncated view of the table.

The reason behind this projection is that the previous table does not give any visual cues as to how `Rule` nodes are related. With a crosstab, one can easily see which rules contain the same facts.

The rows of the crosstab represent the rules, with the name in the left-most column. The columns represent the facts.

The cells show the facts used in each rule. A number preceded by a hash symbol indicates if a rule requires a fact. The number represents the ordinal order of when the facts appear in the conditions. Thus, the first occurrence of a fact is represented by "#1". If there are multiple occurrences of a fact within a rule, multiple numbers will appear in the same cell.

In the figure, a close-up shows that all the details of the selected fact are available in the inspector panel.

At the top, we can see an immediate problem with a crosstab. If we have the whole `File` node represented, then the table will be very sparse.

Everything is editable in this table, including deleting a `FactDeclarationSmartRef` from a `Rule` node. The table plug-in and MPS enabled most of the editing in the projection by default. An extra editing feature we added to this table was the ability to delete a `FactDeclaration` node and all the related references from all the `Rule` nodes in the `File` node by deleting a fact column. The code shown in Figure 4.15, on page 56, shows how we can walk the trees in each `Rule` to delete unary conditions and convert the non-deleted side of binary conditions into unary conditions to allow this `FactDeclaration` reference deletion.

From the "on delete" section of the vertical section, we loop through all the `Rule` nodes to remove the references and rearrange the conditions in the AST. This projection calls a recursive "pruneCondition" method to walk the tree to detach `FactSelector` nodes containing references to the deleted `FactDeclaration`. If it is a unary `AbstractCondition`, then it detaches the condition, thus removing the contained `FactSelector`. Upon removal of both children of a binary `AbstractCondition`, then the condition is removed. Upon removing one child from the binary condition, the remaining child replaces the parent binary condition.

After removing all the `FactDeclaration` references, the code removes the `FactDeclaration` node that the column represents. Now that there is an updated AST, the projection will re-display itself.

Here we end our experiments in the RSD language.

rule file name: DossierSleutelbos **count** 153

rules:

Name	When conditions
0	Dossier( ) DroolsContext( )
hello	Dossier( )
Dossierdetails kunnen inzien	Dossier( )
Dossiers samenvoegen	<< ... >>
[*] dossier_NieuweNotitie	<< ... >>
[*] dossier_StuurVeiligeMail	<< ... >>
Dossier verwijderen	<< ... >>
[HV] dossier_Sluiten	<< ... >>
[HV] dossier_Afdoen	<< ... >>
[HV] dossier_Actualiseren	<< ... >>
[DOSSIER] dossier_BewerkNAW_Rihg	Milestone(setUpRihgStep, !finished[groen])
[DOSSIER] dossier_BewerkNAW_Gvk3	Milestone(setupGVK3Step, !finished[groen])
[DOSSIER] dossier_StartHvAanvraag	Dossier(!hasRunningEpisode)
[WVGGZ/CM] Bewerk NAW in WVGGZ CM	<< ... >>
[WVGGZ/VO] Bewerk NAW in WVGGZ VO	Professional(!isReportReciever) // cannot handle checking professional works for owner of episode

[WVGGZ/VO] Bewerk NAW in WVGGZ VO voor meldmedewerkers	Professional(isReportReciever)
	// cannot handle checking professional works for owner of episode
	Milestone(enterNotificationStep, open[wit/rood])

[WVGGZ/IBS WZD] Bewerk NAW in WVGGZ IBS WZD	<< ... >>

Figure 4.13: Table projection

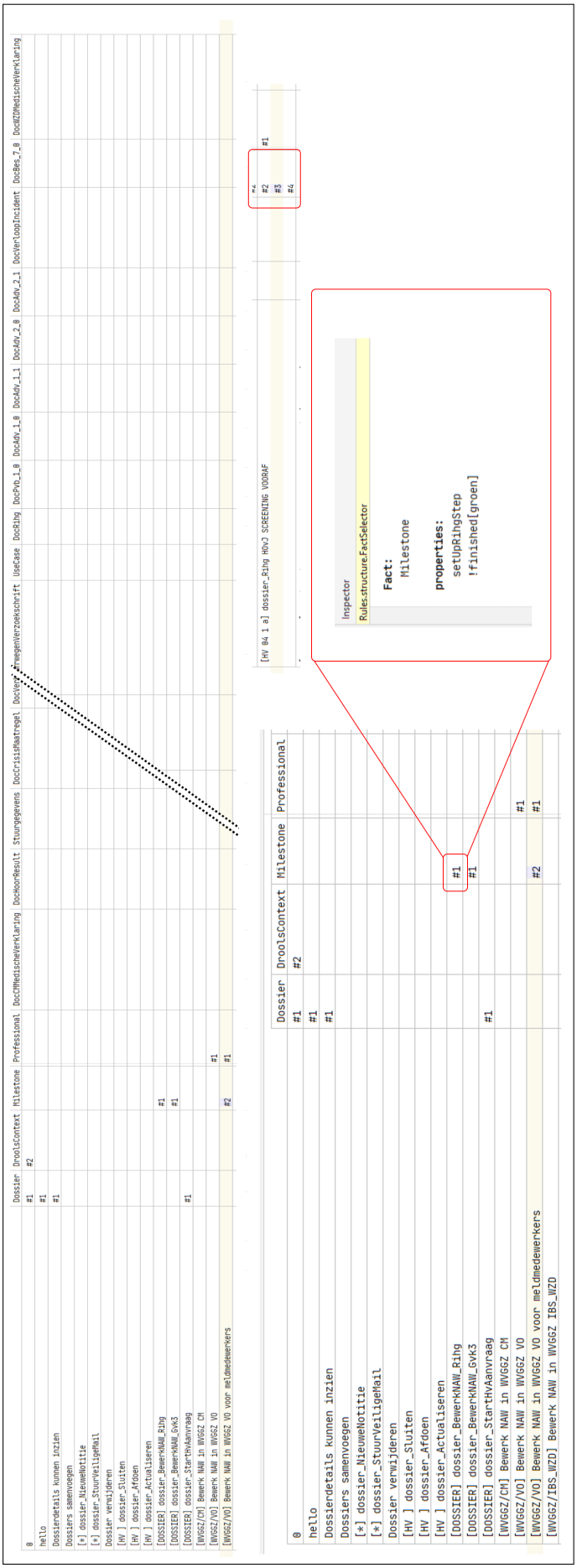


Figure 4.14: Crosstab projection

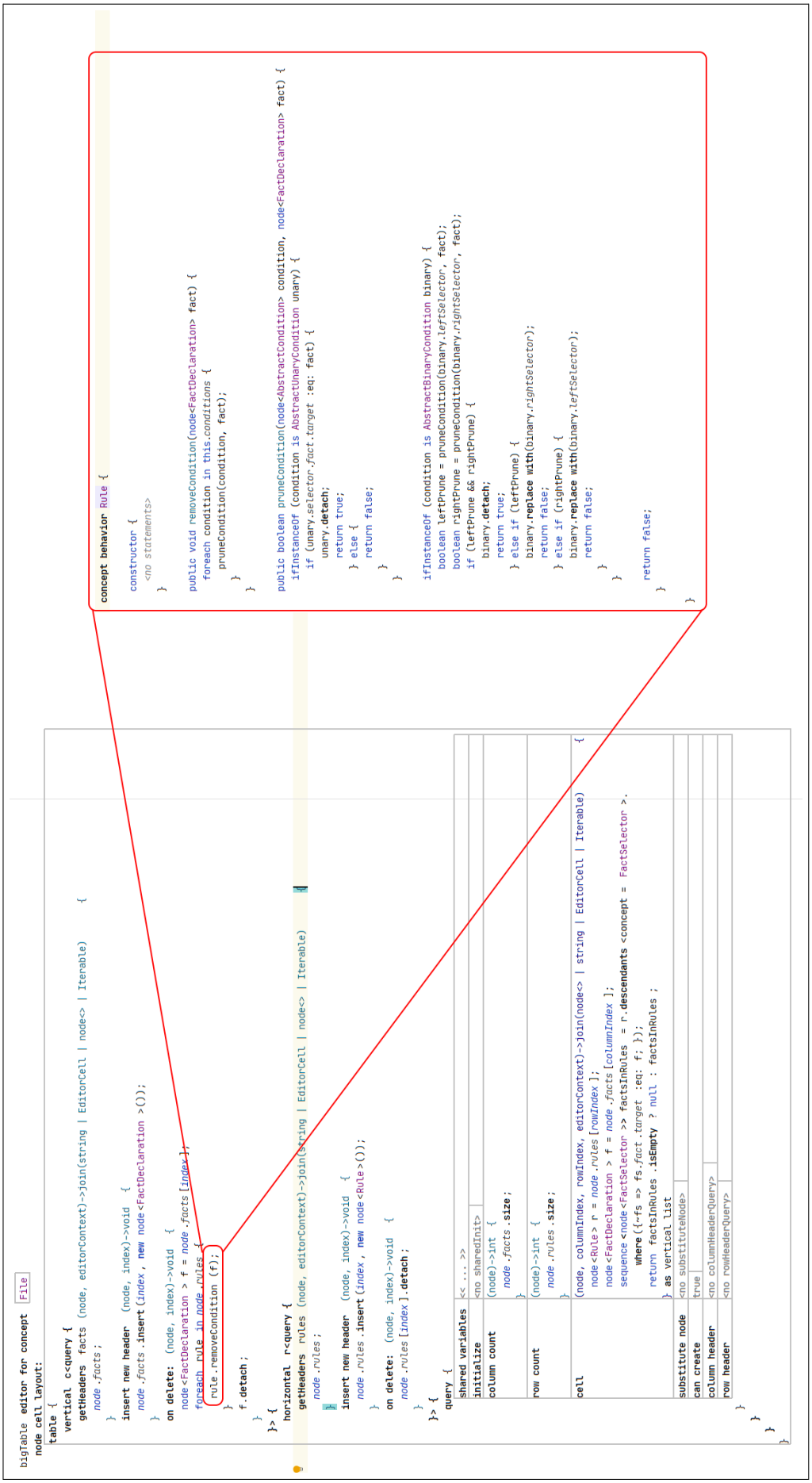


Figure 4.15: Table Fact deletion code

4.2.2 Drools-Lite

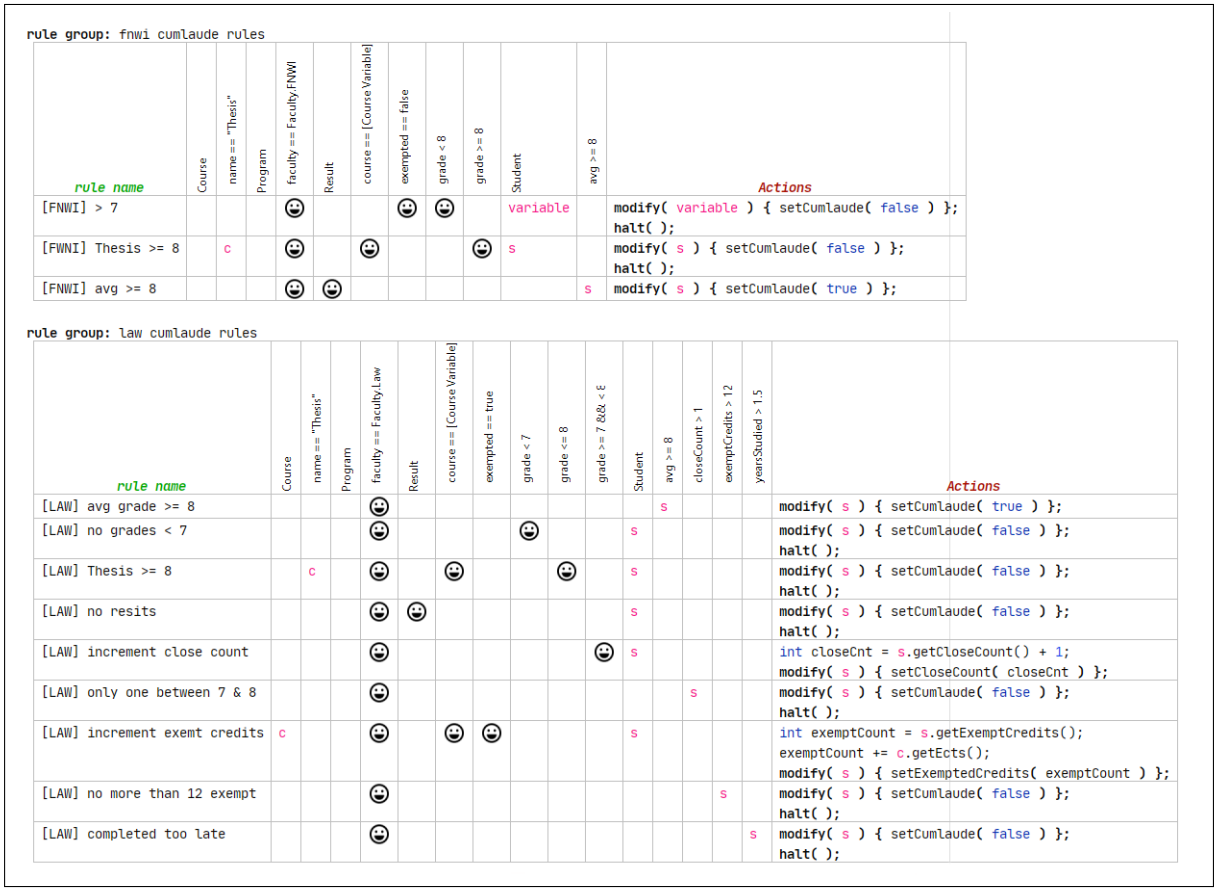
Our subsequent experiments were with projections with the Drools-Lite language. As described in Section 4.1.2, Drools-Lite is an implementation that is much closer to the complete Drools language. This realism will allow us to create projections that we can present to experienced Drools developers for evaluation.

Of the learnings from the RSD language, one we felt needed fixing to improve understanding was the sparseness of the tables. By implementing the principle of maximising cohesion, we discovered we could reduce the sparseness issue. Therefore, as a precursor to our projections, we extended Drools-Lite with a new language that contained one structural item - the `RuleCollection` Concept. A `RuleCollection` node is a child of the `File` Concept and holds a collection of `Rule` nodes. This idea is that related `Rule` nodes can be placed in the `RuleCollection` node to make it easier to examine them together. This language naturally also added an editor for the `RuleCollection` Concept. Additionally, we added intentions to move `Rule` nodes in and out of groups.

4.2.2.1 Decision Table

As the Drools language is analogous to a series of if-then statements, then perhaps its best visual equivalent is the decision table. Decision tables are a "powerful aid in programming, documentation, and in effective man-to-man and man-to-machine communications"[109].

We designed our table, shown in Figure 4.16, to include some of the lessons learned from the RSD crosstab shown in Figure 4.14. The RSD language taught us that wasting visual real estate exacerbates sparseness issues in tables. In the crosstab table, horizontal scrolling is necessary, in part due to the column widths. The columns were wide because the name of the `FactDeclaration` node was displayed horizontally.



rule group: fnwi cumlaude rules

rule name	Course	name == "Thesis"	Program	faculty == Faculty:FNWI	Result	course == [CourseVariable]	exempted == false	grade < 8	grade >= 8	Student	avg >= 8	Actions
[FNWI] > 7				☺			☺	☺		variable		modify(variable) { setCumlaude(false) }; halt( );
[FWNI] Thesis >= 8		c		☺		☺			☺	s		modify(s) { setCumlaude(false) }; halt( );
[FNWI] avg >= 8				☺	☺						s	modify(s) { setCumlaude(true) };

rule group: law cumlaude rules

rule name	Course	name == "Thesis"	Program	faculty == Faculty:Law	Result	course == [CourseVariable]	exempted == true	grade < 7	grade <= 8	grade >= 7 && < 8	Student	avg >= 8	closeCount > 1	exemptCredits > 12	yearsStudied > 1.5	Actions
[LAW] avg grade >= 8				☺								s				modify(s) { setCumlaude(true) };
[LAW] no grades < 7				☺				☺			s					modify(s) { setCumlaude(false) }; halt( );
[LAW] Thesis >= 8		c		☺		☺			☺		s					modify(s) { setCumlaude(false) }; halt( );
[LAW] no resits				☺	☺						s					modify(s) { setCumlaude(false) }; halt( );
[LAW] increment close count				☺						☺	s					int closeCnt = s.getCloseCount() + 1; modify(s) { setCloseCount(closeCnt) };
[LAW] only one between 7 & 8				☺									s			modify(s) { setCumlaude(false) }; halt( );
[LAW] increment exemt credits	c			☺		☺	☺				s					int exemptCount = s.getExemptCredits(); exemptCount += c.getEcts(); modify(s) { setExemptedCredits(exemptCount) };
[LAW] no more than 12 exempt				☺										s		modify(s) { setCumlaude(false) }; halt( );
[LAW] completed too late				☺											s	modify(s) { setCumlaude(false) }; halt( );

Figure 4.16: Decision table projection

The Drools-Lite language allows for much longer selection criteria on `FactProperty` references, which would lead to much wider columns. Our solution was to develop a vertically orientated header cell and use indentation to indicate if the cell is referring to just the `FactDeclaration` node or a `FactDeclaration` and `FactProperty` node combination.

Because this projection presents both the left and right-hand side of the rules, we had to handle the Concept that spans both - the `RuleVariable` Concept. We had to find a way to represent a `RuleVariable` node that

can be bound and used on the LHS and used on the RHS. We achieved this by referencing a `RuleVariable` node's name in the cell representing the `FactDeclaration` reference or `FactProperty` reference to which it is bound. With `RuleVariable` nodes now being represented in the cells, we could no longer represent the cell being selected with an "X", as this could be confused with a `RuleVariable` node's name. Projectional editing does not require communication of meaning through parsable ASCII text. Thus, we decided to represent `FactDeclaration` nodes selection with an image. For arbitrary reasons, we chose a smiley face as that indicator.

The `Rule` node's names and actions are editable through the default functionality of the MPS extension. We use intentions to add the selection of a `FactDeclaration` node or `FactProperty` reference to a `Rule` node. We also use intentions for binding `RuleVariables`.

The major drawback of this design is that editing a `Rule` node with yet non-existent selection criteria became very clunky. If the `Rule` node we wished to edit already existed in the table, we had to use an intention to extract it from the group, change the criteria and place it back in. Then, the table would automatically adjust the column headings.

Experts examined this design in the questionnaire.

4.2.2.2 SpreadSheet

The domain-specific language for the finance world is the spreadsheet. One study estimated that 90% of computers had a spreadsheet on them[110]. Dan Bricklin's VisiCalc drove personal computers into the office. VisiCalc was succeeded by Lotus 1-2-3, which Microsoft Excel then succeeded as the dominant spreadsheet program in the workplace.

This level of familiarity with a paradigm led us to design a projection that had the look and feel of an Excel spreadsheet. We show this design in Figure 4.19 on page 59. To this end, we created a design where the selection criteria could be directly edited in the cell, as highlighted in the figure.

Each row is a `Rule` node in this design, and each column is for a `RuleVariable` node or a `FactProperty` reference. If a property is selected, then the selection criterion is in the appropriate cell. A grey/beige colour indicates unselected cells. The RHS of the `Rule` node appears in the Action column. Adding as yet unused `FactDeclaration` references or `FactProperty` references, or removing existing ones, can be achieved with intentions, as shown in Figure 4.17.

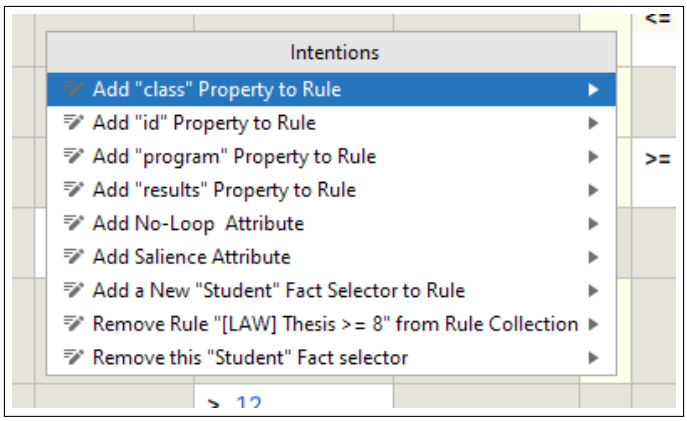


Figure 4.17: Intention

This design also allowed us to have more than one selector for the same `FactProperty` reference, essential for our host organisation's code. We demonstrate this in Figure 4.18.

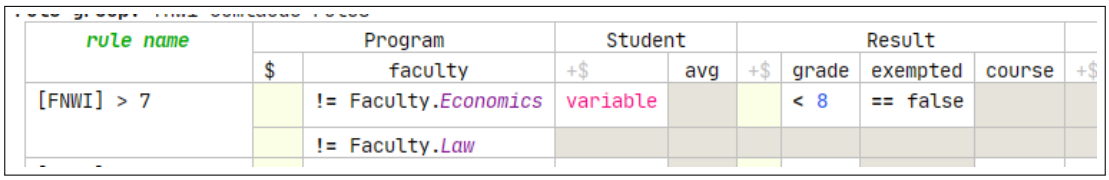


Figure 4.18: Two of same property

Experts examined this design in the questionnaire. Here we end our experiments in the Drools-Lite language.

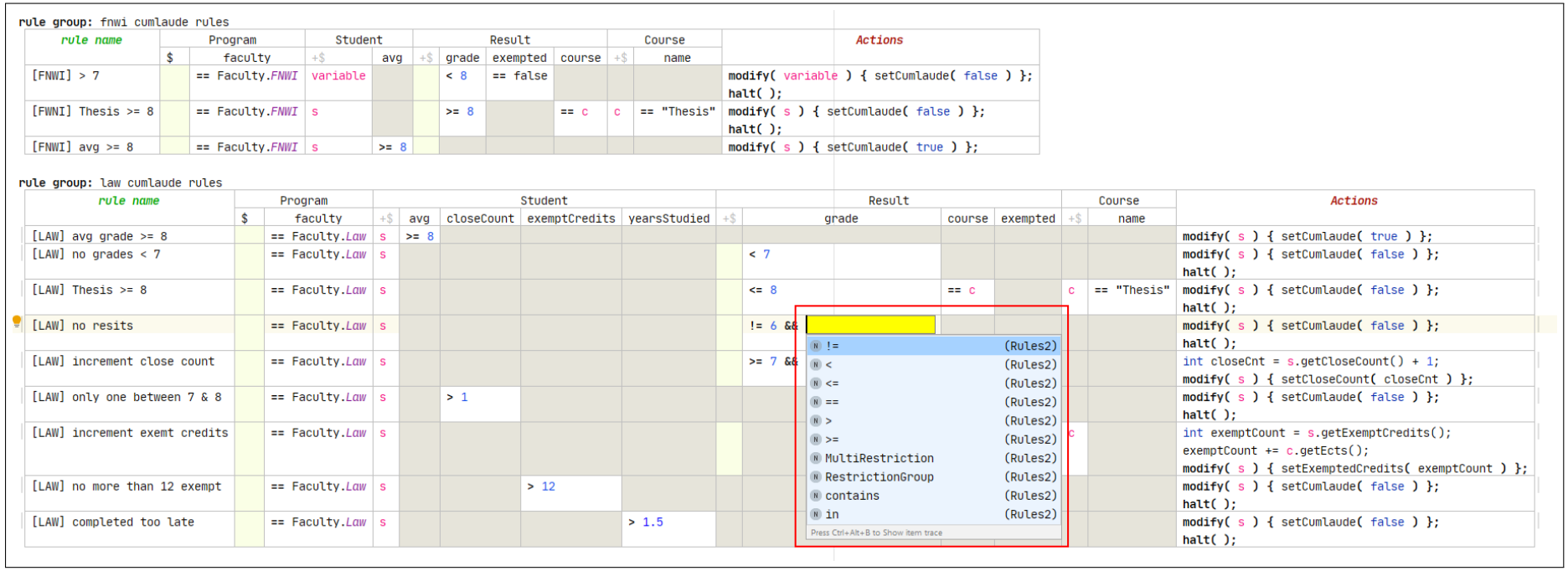


Figure 4.19: Spreadsheet projection

4.2.3 Wireframe

After brainstorming several ideas to present as wireframes to experts as possible projectional aids to understanding, we chose two. We discuss them briefly in this section.

4.2.3.1 Truth Table

We decided to produce a truth-table wireframe example as we had had personal experience building truth tables to confirm the validity of Drools rules in our work.

The truth table seemed apt for the LHS of the Drools rule as, in essence, it is a boolean function. Wittgenstein popularised the truth table in the Tractatus Logico-Philosophicus[111]. They are so widely used in mathematics and computer science that we do not need to explain their use further. Because of the combinatorial explosive nature of truth tables, with 2^n possible combinations, we would limit the display to a max of 6 `FactSelector` nodes and only show the paths that lead to the RHS execution.

Figure 4.20 shows how we designed this to look. The user experience would be that the `Rule` node is selected, and the developer presses the up and down arrow keys to step through the different true (highlighted in green) and false (highlighted in red) `FactSelector` nodes that result in the `Rule` nodes selection.

Figure 4.20: Truth table projection

We presented this design to our experts to be validated.

4.2.3.2 Circuit Diagram

In our final projection design, we wanted to present a part of projectional editing that we had heretofore only made minimal use of. That is the use of manipulatable graphics that can change the AST.

We chose a logic circuit. The logic circuit represents a boolean operation as NOT, OR, XOR and AND Gates, with their inputs and outputs being inputs to other gates. In our design, shown in Figure 4.21, the input wires to the gates are the `FactDeclaration` nodes or `FactProperty` nodes referenced in the LHS.

The user experience is that once the `Rule` node is selected, the developer, by pressing the up and down arrow keys, can step through the different `FactSelector` nodes (highlighted in yellow) and shown in the circuit diagram, thus showing how the `FactDeclaration` nodes relate to each other.

We present this design in the questionnaire for validation.

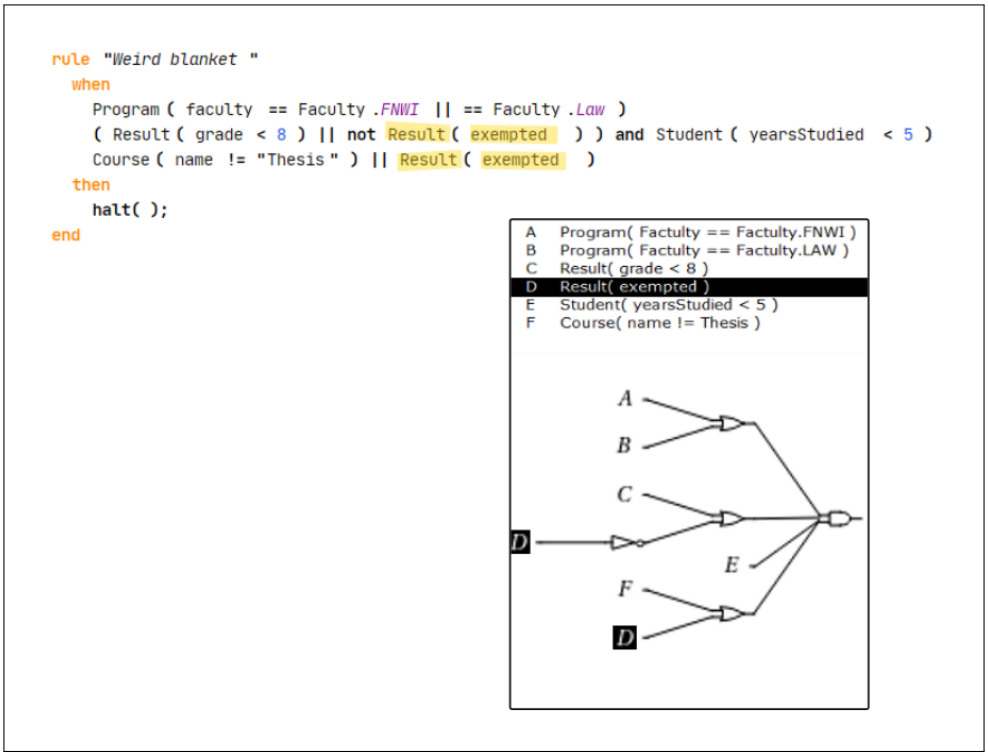


Figure 4.21: Circuit diagram projection

4.3 Discussion

We now examine threats to the conclusion, instantiation, internal and external validity of our DSR and its reliability.

4.3.1 Threats to Validity

4.3.1.1 Conclusion Validity

We felt that the projections we built improved our overview of large rule sets. However, as we spent a lot of time with the rules as the projections are being built, we could have been influenced by the previous knowledge of the rules. Thus, we needed to validate our conclusions with others, which Chapter 5 attempts to do.

4.3.1.2 Instantiation Validity

As Lukyanenko[112] points out about the validity of DSR, “executable software systems differ in important ways from “traditional” experimental stimuli . . . thus unique challenges inherent in the design of IT artefacts warrant additional attention”. Thus, here we use his “Instantiation Validity” rather than the traditional construct validity.

We were trying to observe whether projectional techniques could help with understanding Drools rules. As non-projectional language workbenches could achieve some of the outcomes, these did not prove a link between better understanding and projectional editing. As the tabular projections would not be achievable in parser-based languages, these could show the direct relationship between projections and understanding.

We offered only a few solutions possible in the instantiation space. Given more time, we could have provided more candidates to improve understanding.

The artefact complexity, with regards to a user having to learn methods of interaction with a projectional editor, may impact the view of understandability. The nature of software artefacts means that it is challenging to ensure that the projections we show represent the possible projections.

Developing working software is a resource-hungry activity. It took us a long time to get to an adequate implementation of the Drools-lite language. As a result of this constraint, we did not create the ideal artefact, with only a few projections accomplished. This limited supply of alternatives does not allow us to control for confounding effects.

4.3.1.3 Internal Validity

Was the outcome a result of the treatment? By which we mean, if there was a better understanding of the code, was this a consequence of the projectional editing or some other factor? There is a risk from the interaction of different treatments impacting the outcome. Many advantages come from the implementation in MPS. These include the context-aware code completion menus. Some of the side effects of MPS, rather than the core projectional concept, could have impacted the change in understandability.

4.3.1.4 External Validity

Our Drools-Lite language is not a full implementation of the Drools language. To take in the whole language would have taken more time. Whether the examples we built would generalise to all the functionality of Drools is not known.

The mono-operation bias of approaching our research question about business rules and projectional editing by only implementing Drools and only using MPS could mean that the results do not generalise to other business rules languages and editors built with different language workbenches.

4.3.1.5 Reliability

Reliability is how the data and the analysis is dependent on specific researchers. With DSR studies that build a working opensource prototype, then the reliability of the data is known. The code can just be downloaded and run by any interested party.

The reliability of our analysis that the projections improved our understanding is open to the effects of a range of cognitive biases. Together with experimenter expectancies in the design, this could lead to an unreliable conclusion. To mitigate the effect of our biases, we surveyed others, as described in Chapter 5.

4.4 Summary

In this chapter, we presented a description of the details of the DSR that we conducted. We pursued two languages. In our pilot study, we implemented the Really Simple Drools language and a few projections on top of this. Once we had established the usefulness of this approach, we created the language Drools-Lite which was a language that would be recognisable to experienced Drools developers.

Of the projections we created, the most capable were the Spreadsheet-like and Decision table-like projections. These both succeeded in massively reducing the screen real estate required. Other advantages included being able to group related rules and being able to visually scan which rules used the same facts and properties.

We could not show rules with complex nested conditions in the tabular projections — however, the ability to mix notations mitigated this. We could show rules too complicated for the tabular projections with textual projections in the same file.

5

Survey

In this chapter, to answer the question "Do projections of Drools business rules lead to a greater understanding of those rules?" we undertook research using a survey. In Section 5.1, we describe the method of this survey. Section 5.2 examines the outcomes of our research. Finally, in Section 5.3, we discuss the threats to the validity of our approach.

5.1 Method

We tested the validity of the prototype using a survey. If a survey is not well designed, then it could lead to invalid or irrelevant outcomes. This chapter describes the design and procedure of the survey. Additionally, it outlines any threats to its validity. Our choice of survey technique is a questionnaire.

5.1.1 Questionnaire Design

To design the survey of our prototype, we followed the following rules derived from the works of Bryman[113] and de Vaus[114].

- *Introduction:* We devised a clear introduction to describe the research.
- *Existing work:* We considered existing questions. Regarding projectional editing, we requested the original questionnaires from three papers[31, 36, 54] about tools developed using projectional editing. Unfortunately, none of the original questions were available for assessment.
- *Question in mind:* We had the specific research question, "Which projections can help developers get appropriate feedback about rules?" in mind when formulating the questions.
- *Succinct:* The pool of Drools users that we were personally in contact with was tiny. Thus, we had to rely on responses from strangers. For this reason, we tried to make the questionnaire as quick to finish as possible. This constraint meant we looked particularly hard at removing questions that did not help us to our research goal.
- *Pilot:* We piloted the questionnaire with both ourselves and our industrial supervisor. The result of this pilot led to more explanative text before the questions.
- *Clarity:* The instructions to each of the questions were tested for clarity by a non-technical third party. We took care to rework questions that were long, ambiguous, general or leading not to be so. We also took care to remove jargon, negative wording, and questions that asked about more than one thing.
- *Closed questions:* The only open questions were ones from which we wished to extract sentiment. To avoid binary questions, where appropriate, we applied a Likert scale[115].
- *Single page questions:* Thanks to the UI of SurveyMonkey, no questions spanned multiple pages.
- *Important questions first:* We started with the research-based questions, leaving the socio-demographic questions, such as skill level, to the end.

Appendix F shows the questionnaire we designed following these principles.

5.1.2 Participants

The requirement for participants is that they have at least a little experience with using Drools. We hoped to get a statistically significant number of participants.

5.1.3 Validity

We addressed the non-response bias[116] by making the questionnaire short and easy to answer. Because of the nature of the participant selection for this survey, it will be challenging to address the self-selection bias caused by the voluntary nature of the response.

Common method bias, i.e., "variance that is attributable to the measurement method rather than to the construct the measures represent"[117] can be responsible for 25% or more of variable relational influence. As we are only conducting a single survey, we will not be able to prevent this. However, we took the following small precautions. We tested the survey to remove question ambiguity, mood influences, and length issues. We mixed the order of questions in the survey to mitigate the issues caused by the similarity, proximity, and location of items. We varied the scales and order of our Likert scales.

The main statistical methods to address common method bias, i.e., "Harman's single factor test"[117] and the "marker variable"[118] have been found to be lacking in grounding[119]. The marker variable approach is appropriate if used with caution. However, it may not be possible to gain a statistically significant outcome with our expected response size.

5.1.4 Pre-test

We sent our first pass of the survey to our industrial supervisor, who has experience with Drools. With this pre-test, we hoped to remove ambiguously worded or leading questions. Additionally, we wanted to confirm that the questionnaire took around 10 minutes to complete.

As a result of this pre-test, we updated much of the explanatory text.

5.1.5 Sampling

Within our professional network, we only had a connection with very few Drools developers. We also considered that having acquaintances answer the questionnaire could introduce some biases we could not account for. So, we had to expand our sampling reach.

Our first approach was to search StackOverflow for question askers and answerers about Drools. Our preference was to find email addresses, failing that Twitter contacts. Unfortunately, this proved quite limited. We only managed to harvest 13 email addresses and six Twitter handles.

Our following approach was to interrogate our LinkedIn connections for anyone who claimed Drools as one of their marketable skills. At two degrees of separation in our LinkedIn network, we found 204 candidates with Drools skills listed. From these, we harvested 54 email addresses and 40 Twitter handles.

We chose not to expand our search to three degrees of separation. At two degrees of separation, we share a relationship with the same person. We thought it would be harder to take advantage of the social pressure to answer when we do not have that shared relationship.

Our final group of potential respondents were people who had academic written papers that included developing something with Drools. We limited this pool to papers written in the previous 2 years. We harvested 73 emails from these papers.

5.1.6 Procedure

As described in Appendix F, the questions were turned into a survey using the SurveyMonkey service. We also show screenshots of one version of the questionnaire in Appendix F.

We crafted a short introduction email to encourage response, heavily relying on techniques designed to enhance response as discussed by Cialdini[120].

5.2 Results: Survey

5.2.1 Population Selection

We initially had two sources to find experienced Drools users to be subjects of our survey. From LinkedIn, we selected users who were at two degrees of separation from us and listed Drools in their skills. From StackOverflow, we selected users who had asked or answered questions about Drools.

In these two websites, the users do not typically list their contact details. Some investigation was required. From the initial selection, we harvested email addresses and, failing that, Twitter handles.

Following this, we determined that academic papers could be a good source of a population of expertise. We used Google Scholar to look up Drools papers from the previous two years. After skimming the papers to ensure that they were explicitly about or using the Drools language, we harvested email addresses from them.

On the second and fourth day after the initial sending of the survey, two subjects forwarded a web link to mailing lists. A developer from the core Drools team sent it to a list of known Drools consultants. The other sent it internally in his company. The subjects who sent the survey to their mailing list forwarded links to version C of the survey.

We had created four versions of the questionnaires to combat the single source bias. We distributed the surveys to the subjects harvested from LinkedIn and StackOverflow evenly. There was an overrepresentation of Survey C because of the mailing lists. Therefore, we distributed the subjects harvested from academic papers evenly over Surveys A, B and D.

We show the collection results in Figure 5.1. We see here that the collection method did not have much of an impact on return rates. Whilst StackOverflow had a higher rate, the number of people contacted was small, thus having an outsized effect on the response rate.

The first three pie charts represent the collection methods over which we had control. These three represented 24 of our 30 completed questionnaires. The last pie chart represents six completed and four partially completed questionnaires returned from the mailing lists. We do not know the size of the starting population of these mailing lists. Thus, this pie chart only shows the ratio of partial to completed results.

In summary, a survey reached 154 potential participants, of which 24 completed it, for a Response Rate of 15.5%. In addition, an unknown number of potential participants were reached through mailing lists, returning a further six completed surveys.

5.2.2 Participant Demography

Responses came from around the world. Figure 5.2 shows that most respondents reside in Europe, except for one in each of the USA, Israel, and Singapore. Italy and the Netherlands provided the most significant number of responses, with 7 and 5 respectively.

The experience of our subjects was relatively high. As can be seen in Figure 5.3a, most of our subjects have over ten years of programming experience. 17% of our recipients had a low experience of Drools, and 30% were very experienced, as shown in Figure 5.3b. Figure 5.3c shows that 40% of our recipients had used Drools in this current year, and 30% had not used it in the last five years.

Half of our subjects reported only ever using one editor for Drools, with the slight majority of those only using Eclipse. Eclipse also had the most instances of reporting of having been used. Out of the 55 instances of editors reported as being used, 20 of those were Eclipse. There was a surprising diversity of tools used. The purpose of this section was to calibrate responses against exposure to IDEs with greater Drools Support. The wide diversity of editor usage and high incidence of multiple editor usage means that these answers are not suitable for use in the sub-categorisation of responses. We show the distribution of usage in Figure 5.4 on page 67.

5.2.3 Question Analysis

5.2.3.1 Grouping

When displaying subtypes, we shall split them into groups. The source of the response will create a pseudo-cross-section of our participants. The ten contacted through academic papers will be considered our academics. The remaining 20 will be considered practitioners.

The next grouping is based on Drools Experience. The nine who replied they had used Drools “for years and intensely” are categorised as experts. The 16 who either answered “for years, but occasionally” or “not for long, but intensely” we categorise as seniors. The five who answered “I barely touched it” are categorised as novices.

Another grouping will be on recency of use. The 12 who have used Drools in 2021 are categorised as current users, the remaining 18 as past users.

To remove some bias in the questionnaires, we changed question order, order of projections and used rule sets. When a question is affected by this, then this segregation will also be displayed.

5.2.3.2 Display

For the remainder of this results section, whilst displaying results that regard our Likert scaled questions, we will be following the advice of Robbins et al.[121] by using diverging stacked bar charts, with counts added.

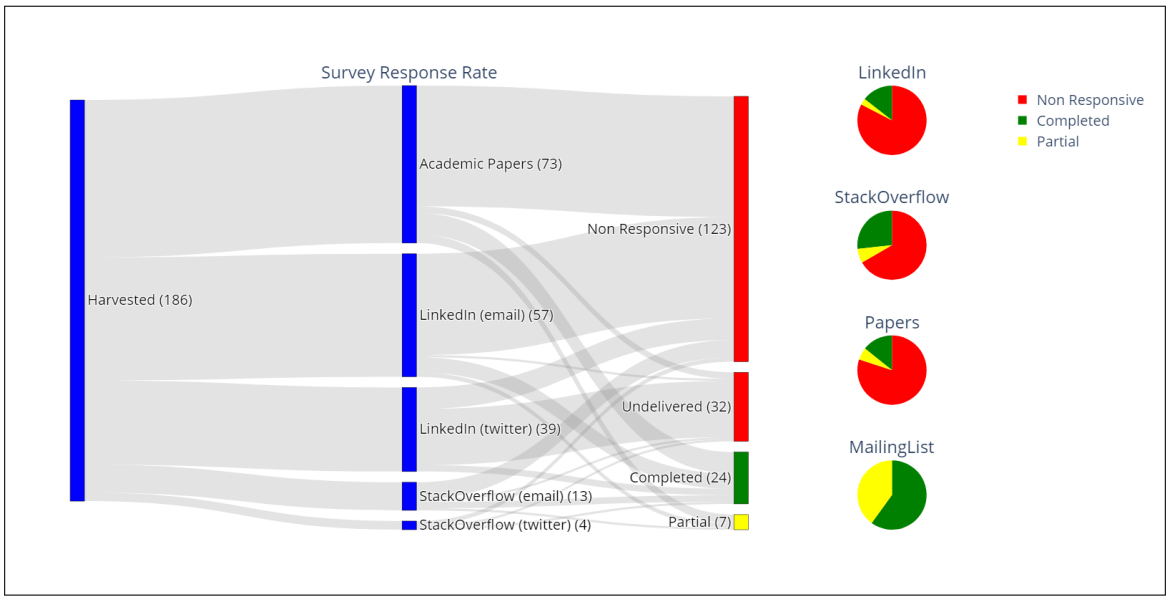


Figure 5.1: Survey participants

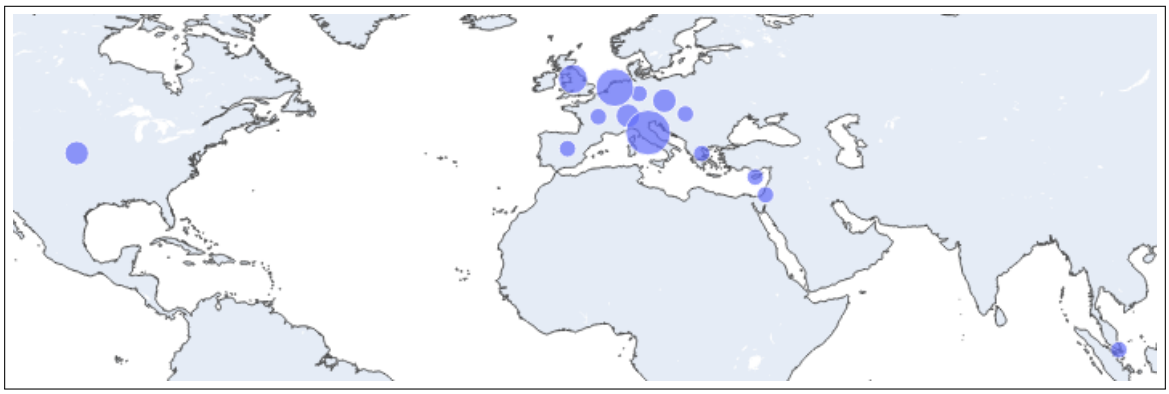
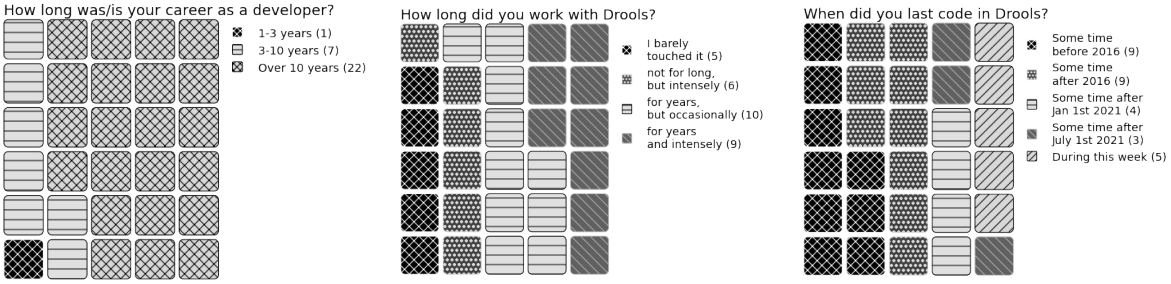


Figure 5.2: Survey locations



(a) SE experience **(b) Drools experience** **(c) Recent use**

Figure 5.3: Subject experience

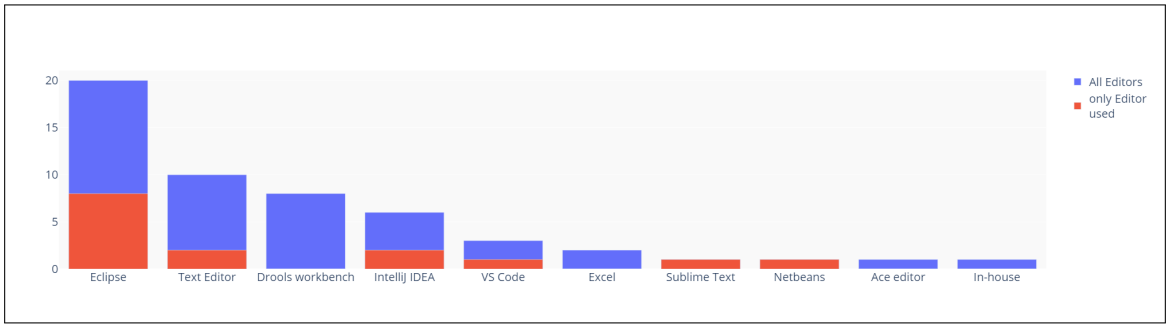


Figure 5.4: Editors used

This style allows the evaluation of the results of subclasses. The addition of counts makes it easier to spot when small numbers skew the results.

In our charts, we will place positive scores to the right of the centre line and neutral and negative scores to the left. Our particular question design cannot tell if the neutral responses are substantive or hidden non-responses[122].

5.2.3.3 Grouping Analysis

Our data does not fall under a normal distribution. So, we will be using nonparametric analysis. Vaus[114] advises that when analysing ordinal variables with nominal grouping variables, then the statistical methods one should use for checking for differences between groupings would be the Mann-Whitney[123] or the Kruskal Wallis[124] tests. In the grouping section, we had one group of only 5 participants, the novices. As this group is too small to think of using in analysis, we will ignore it. Ignoring this group means that all the groupings we have are divided into two. As Mann-Whitney is a specialisation of Kruskal Wallis for two groups, this is the analysis we will use.

For all our analyses, our null hypothesis, H_0, is that there is no difference between the ranks of the two groups. Our alternative hypothesis, H_1, is that there is a difference between the ranks of the two groups. Our alpha, or significance, level will be 0.05 for no other reason than it seems to be a mutually agreed-upon value within the statistical community and justifying a different significance would take more time than is warranted for the value these analyses will give. Our sample size is greater than 20. Therefore, we can use a Z distribution.

To work out our Z score, we have an alpha level of 0.05 and a two-tailed test. Thus, we could work the distribution out by using the following formula:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

However, it is easier to look up in a Z table. Thus, we have an "area in body" of 0.9750, which correlates in the Z Table to a z value of 1.96. Ergo, our decision rule is, if our z is less than -1.96, or greater than 1.96, we reject our null hypothesis. [1]

[1] We carried out the calculations using the Mann-Whitney U Test calculator at `https://www.socscistatistics.com/tests/mannwhitney/default2.aspx`

5.2.3.4 Question 1, 2 and 3: First Impressions

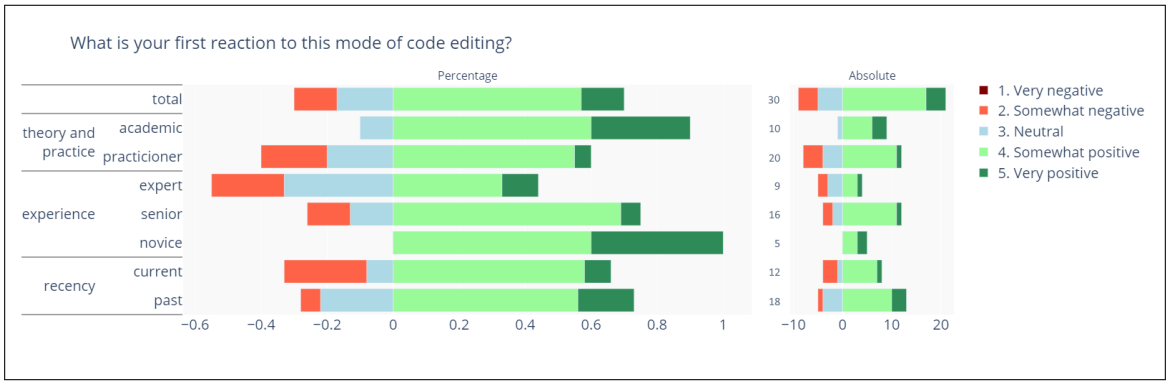


Figure 5.5: Question 1 - first impressions

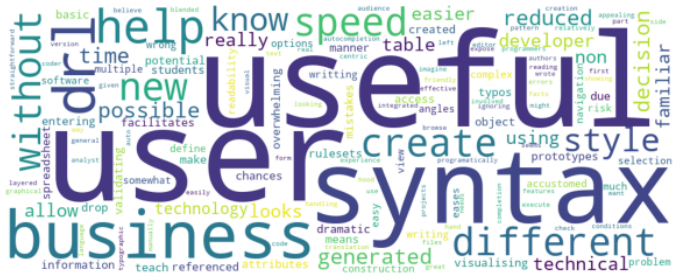
Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	55	54.5	-1.97974	0.0477	$\mathbf{H_1}$
Expert vs Senior	37	55	0.93413	0.35238	H_0
Current vs Past	61	91	0.6985	0.48392	H_0

Table 5.1: Mann-Whitney test question 1 - first impressions

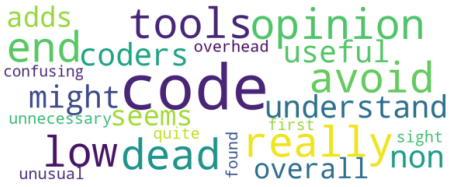
This question shows the subject an example of projectional editing a Drools file alongside a table projection, as an animated GIF, along with an explanation. Then she is asked her first reaction. The chart in Figure 5.5 shows the outcomes. Eyeballing the chart, we observe that the novice and the academics, where there is much crossover, found the initial presentation more positive than the experienced practitioners.

Table 5.1 shows a significant difference between the academics and the practitioners' first reaction to seeing the projectional editing example. Here, the academics have a far more positive view of the example than the practitioners.

In general, we see an overwhelmingly positive response. There were five times as many positive (21) than negative (4) responses. SurveyMonkey directed those who had a positive or negative response to answer the open questions "Q2. How would this coding style be useful to your interactions with Drools?" and "Q3. What do you find negative with this style of coding?" respectively. Figure 5.6 shows a visualisation of the subject's responses.



(a) Positive



(b) Negative

Figure 5.6: Initial thoughts

Amongst the positive comments, it appears the subjects had picked up on many of the advantages that projectional editing brings. The ones that got the most mentions, using other words, were exploratory coding, correctness by construction, and multiple viewpoints. Subjects also noted that, with the projections shown, development could be quicker and easier to check.

Among the few negative comments, they discussed the failures of no/low code solutions, confusing views, and unnecessary overhead.

5.2.3.5 Question 4 and 5: Interpret Projection

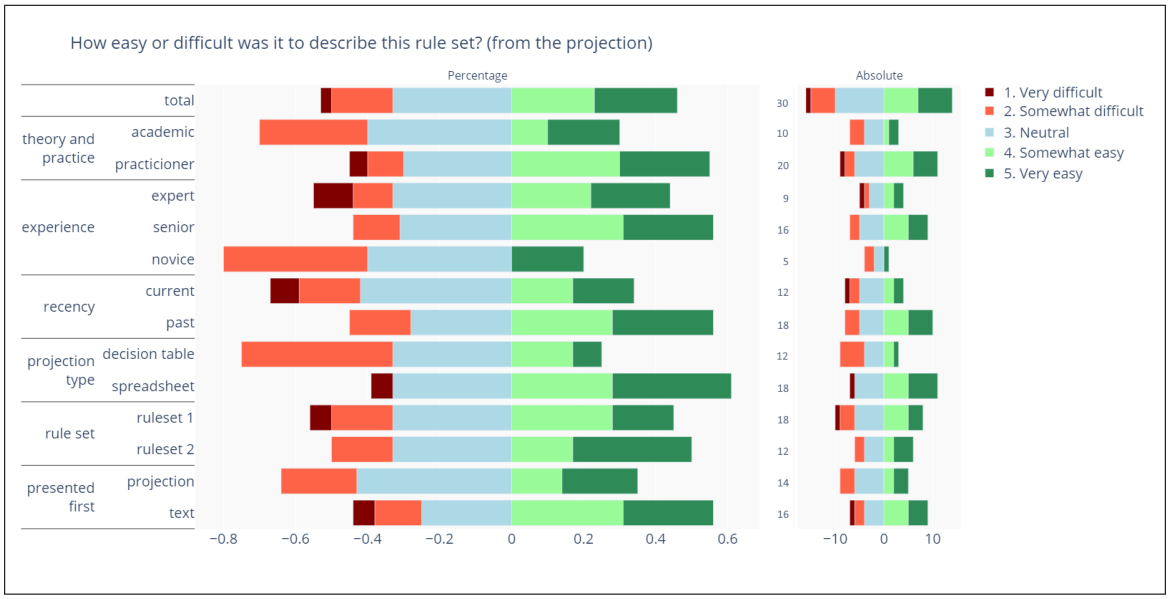


Figure 5.7: Question 5 - interpret projection

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	55	77	0.98987	0.32218	H_0
Expert vs Senior	37	61.5	0.56614	0.56868	H_0
Current vs Past	61	82.5	1.05833	0.28914	H_0
Decision Table vs Spreadsheet	55	61	2.2225	0.02642	$\mathbf{H_1}$
Ruleset 1 vs Ruleset 2	61	92	0.65617	0.50926	H_0
Projection first vs Text first	64	97	0.60277	0.5485	H_0

Table 5.2: Mann-Whitney test question 5 - interpret projection

This question asks the subject to describe the meaning of the projection and then describe how hard it was to do that. Few people described the meaning of the projection well.

The chart in Figure 5.7 shows the outcomes. Looking at the chart, it appears evident that there is a difference in the subjects' confidence between the projections presented. The participants believed they understood the spreadsheet-style projection better than the decision table projection.

The Mann-Whitney analysis, shown in Table 5.2, confirms this observation. Otherwise, no other grouping differed significantly from the general population.

The ratio of those evaluating it as easy to those thinking it was hard to understand the projection was 2:1, 14 to 6.

5.2.3.6 Question 6 and 7: Interpret Text

Figure 5.8: Question 7 - interpret text

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	17	34	0.48869	0.62414	H_0
Expert vs Senior	8	20.5	-0.3873	0.69654	H_0
Current vs Past	12	31.5	-0.04929	0.96012	H_0
Ruleset 1 vs Ruleset 2	17	24.5	-1.36868	0.17068	H_0

Table 5.3: Mann-Whitney test question 7 - interpret text

This question asks the subject to interpret a rule set presented in a Drools style text projection. The purpose of this question was threefold. First, to calibrate how well the subject understood Drools. Second, for comparison with a later projection. Finally, to calibrate whether and how much easier the text was than the tabular projection to those used to seeing the text version.

Unfortunately, we assume that due to the questionnaire design, there were many non-respondents to this question. 12 of the 30 respondents did not answer this question. All 12 of these were from the 16 that were presented the text projection before the tabular projections.

Reporting on 18 responses has significantly less validity. With that in mind in Figure 5.8, we can see much higher confidence in the easiness of understanding the ruleset. We see a 5:1 ratio of a greater belief that it was easy to understand the text projections' meaning. This proportion is significantly higher than those who thought it was easy to understand the tabular projections.

The Mann-Whitney table does not show any significant differences in any of the groups. We cannot report the difference between those presented with the text projection first or the tabular projection first. The 4 participants responding from the group presented text first were too low to give a meaningful score.

5.2.3.7 Question 8: Compare Projections

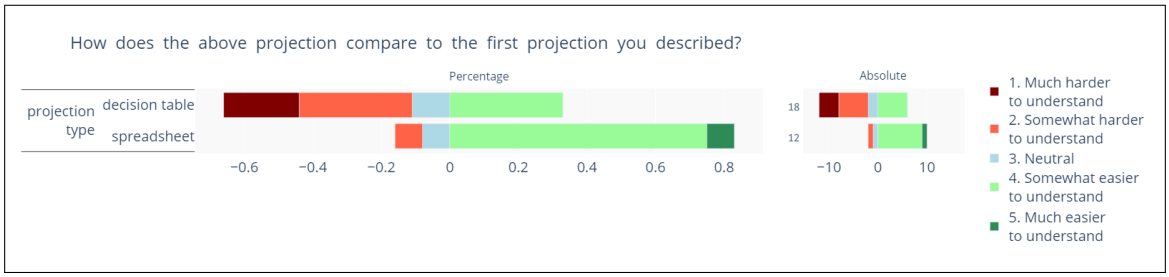


Figure 5.9: Question 8 - compare projections

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Decision Table vs Spreadsheet	61	45	-2.64584	0.00804	$\mathbf{H_1}$

Table 5.4: Mann-Whitney test question 8 - compare projections

This question asked the participant to compare the two tabular projections. This question was to calibrate whether one projection was considerably worse than the other and whether that would affect the comparison with text.

Looking at the chart in Figure 5.9, we have ignored all bars except the difference between the projections presented. The way we constructed this question means that the other groupings are not relevant.

It is evident in this chart and confirmed in the Mann-Whitney analysis shown in Table 5.4 that the spreadsheet presentation is considered far easier to understand than the decision table. This result correlates well with the difference in understanding found previously in question 5.

5.2.3.8 Question 9: Compare Projection to Text

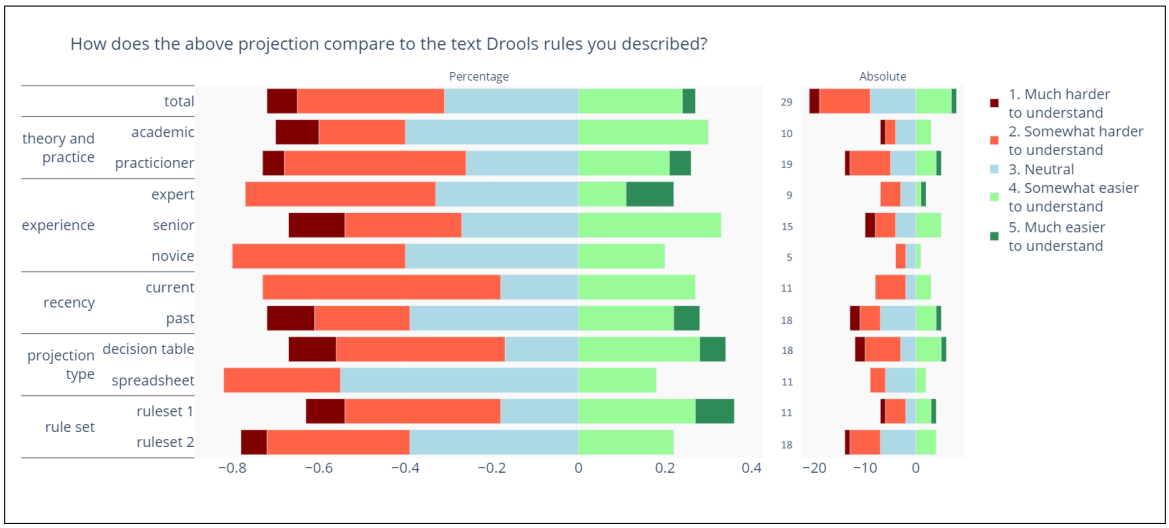


Figure 5.10: Question 9 - compare projections with text

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	52	85.5	-0.41295	0.6818	H_0
Expert vs Senior	34	67.5	0.02981	0.97606	H_0
Current vs Past	55	88	0.47194	0.63836	H_0
Decision Table vs Spreadsheet	55	89.5	-0.40452	0.68916	H_0
Ruleset 1 vs Ruleset 2	55	94.5	-0.17979	0.85716	H_0

Table 5.5: Mann-Whitney test question 9 - compare projections with text

This question asked our subjects to compare a tabular projection with the text projection. There was one participant that chose not to answer this question.

This result, shown in Figure 5.10, was pretty definitive. The subjects found very much that the textual projections were more understandable than the tabular projections. The ratio of harder to easier to understand

was 3:2, with 12 subjects finding the text projection easier and eight finding the tabular projection easier. Table 5.5 shows this was independent of any other factors.

5.2.3.9 Question 10: Truth Table Validation

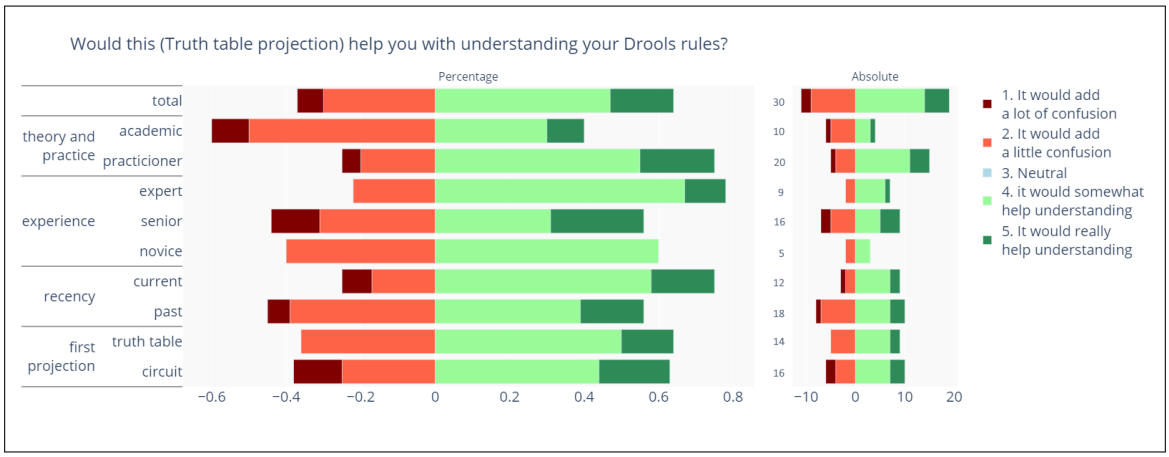


Figure 5.11: Question 10 - truth table

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	55	65	1.5178	0.12852	H_0
Expert vs Senior	37	64	-0.4246	0.67448	H_0
Current vs Past	61	93	-0.61383	0.54186	H_0
Truth table first vs Circuit first	64	108.5	-0.12471	0.90448	H_0

Table 5.6: Mann-Whitney test question 10 - truth table

This question presented the subject with a wireframe of a truth table and asked if it would help them understand rules.

Every subject had a positive or negative view of the projection. 0 of the 30 subjects gave a neutral result. We found this an improbable result.

There was a net positive view of this projection, with a little less than 2:1 finding it more helpful (19) than confusing (11).

On the first view of the chart in Figure 5.11, it seems that academics have a much more negative view of this projection, and experts have a more positive outlook. However, when the figures are examined under Mann-Whitney, as seen in Table 5.6, these differences were not statistically significant.

5.2.3.10 Question 11: Circuit Diagram Validation

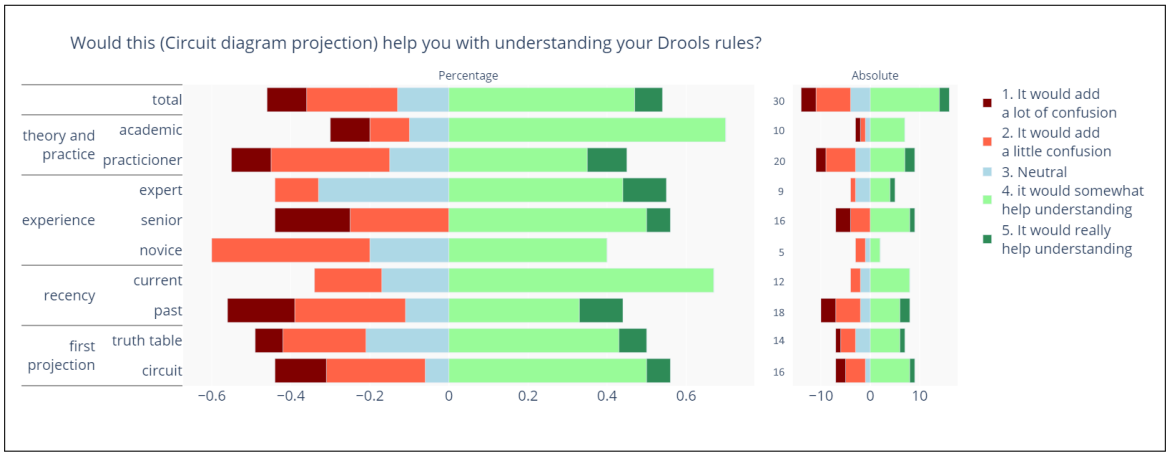


Figure 5.12: Question 11 - circuit diagram

Group Comparison	Critical U	U-value	z-Score	p-value	Hypothesis
Academic vs Practitioner	55	83	-0.7259	0.4654	H_0
Expert vs Senior	37	58.5	-0.73598	0.4593	H_0
Current vs Past	61	83	-1.03717	0.29834	H_0
Truth table first vs Circuit first	64	110	-0.06236	0.95216	H_0

Table 5.7: Mann-Whitney test question 11 - circuit diagram

This question presented the subject with a wireframe of a circuit diagram and asked if it would help them with understanding.

The view of the circuit diagram was a net positive, with a ratio of $\tilde{3}$:2 (16 positive, ten negative).

5.2.3.11 Question 15: Closing Remarks

Our last question asks for any last comments. Sixteen participants decided to add some words.

We ran sentiment analysis on these 16 comments, using the same service and similar code as we used in our SLR research, as described in Section 3.1.6. Nine were positive, six mixed and one negative.

Figure 5.13 shows another word cloud visualisation.

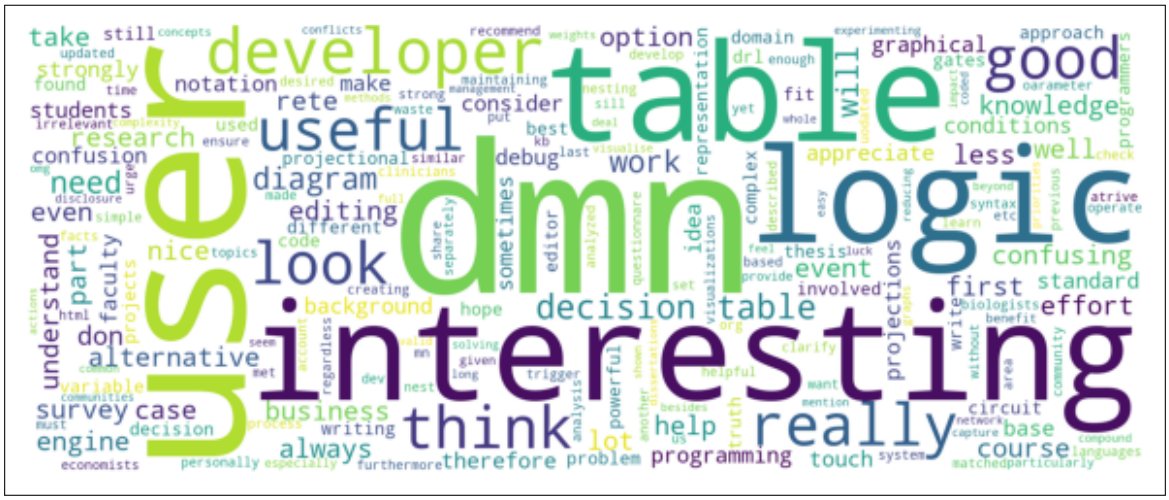


Figure 5.13: Question 15 - responses

The word that pops out, DMN, here is indicative of a few comments. DMN is a tool for working with Drools as a non-developer. There were some comments that our research should look in this direction.

The feeling was that the projections would give more advantages to non-developers.

There were some hints as to how to improve our projections or other projections to try. Others suggested that the projections cannot capture some of the complexity of the rules that exist.

5.3 Discussion - Survey

5.3.1 Threats to Validity

5.3.1.1 Construct Validity

The most prominent issue with this survey is whether it is generalisable to the initial questions it was trying to answer. The question being "can projectional editing be used to increase the comprehensibility of large Business rules files?" There is a solid case that this questionnaire did not ask the right questions to answer this question. It is difficult to simulate large business rules files in a brief questionnaire. Comparing the projections to relatively small and non-complex rules collections may not be generalisable to a significant rules collection.

We perhaps had an issue with a mono-operation bias because we only used one tool for measuring - the survey. We performed many techniques, such as changing orders of questions, measurement, to try and overcome this. However, all the subjects were still only answering a survey delivered through the same medium - SurveyMonkey.

A possible twist on the response bias may occur from two fronts, where parties may participate in hypothesis guessing and answer questions based on their outcome. Firstly, two of the respondents were directly known to us, one through work and another through meetings at conferences. This relationship could have coloured their responses toward a more positive view of our work. Second, two people who responded were part of the Drools core development team and thus work for JBoss/RedHat. On top of that, all the people who answered through the Drools consultants mailing list possibly had a direct monetary relationship with RedHat. This relationship could influence their response towards a more positive view of the status quo.

5.3.1.2 Internal Validity

Whilst we felt we had tried to overcome the apprehension of the subject being judged when answering questions, the fact that one of our questions asked the subject to describe a Drools rule's meaning could overrule all our previous attempts to assuage that fear.

As with all surveys, the higher the sample size, the greater the chance of validity. It is difficult to measure the validity of our outcomes due to the relatively small size of our survey. While 30 respondents are on the low side for using statistical tools to give reasonable responses, statistical validity is also a function of population size. We requested the size of total population size of Drools users from a member of the core Drools development team. Unfortunately, that number was not known. The cross-factor comparisons between the subgroups are of dubious validity as their sample sizes were so small.

5.3.1.3 External Validity

Whilst we feel we reached the right audience of tool users, there was a potential for a geographical selection bias in our population selection technique. Because a portion of our respondents came from our connections on LinkedIn, and we are from Europe, then there is a particular European bias to our respondents. Only four of our completed surveys were not from Europe.

There is also a self-selection bias in the sampling. As this survey is voluntary, and there is no real personal connection between the subjects and us, then the people who would answer this question are the sort of people who would answer an unsolicited questionnaire. This bias may affect generalisability to novices, as there was a tendency towards experts in answering the survey.

5.3.1.4 Reliability

Much like our SLR analysis, we also have the credible threat of a single researcher. All surveys have a subjective nature in their scoring.

Additionally, Our measurement did not consider cultural differences in question answering between the Dutch and Italians, for example.

5.3.1.5 Repeatability vs Reproducibility

Repeatability is difficult as the survey is about a particular implementation of a particular tool. However, our survey could be valid with a different underlying tool, and our population sample selection could work for other researchers.

5.3.1.6 Method improvement

We feel our largest problem was smallness. We would have tried various measures to increase our sample size if we were to try this again. These would include contacting people directly within LinkedIn, rather than only those whose email addresses or Twitter handles we could harvest. Also, we would have followed up on the partially completed questionnaires. Another option would be to select a more extended period for academic papers from which to harvest addresses.

To overcome our mono-operation bias, we could interview people in person.

5.4 Summary

In this chapter, we presented a description of the details of the survey we conducted. Thirty, mostly experienced or expert Drools users, responded.

The goal of this survey was to find if users of Drools would find the projections we created useful. The outcome seems to be that they seem helpful, but not better than the text they are currently used to.

We tried to control for different groupings that may affect the responses, and in most cases, we found no significant differences in the groups we presented. There was a significant difference between the projections presented, with the sample, in general, preferring the spreadsheet projection over the decision table.

6

Related Work

We split related work into two sections. Namely, research relating to the state of projectional editing and research that address the understandability of business rules.

6.1 The State of Projectional Editing Workbenches

Whilst we were particularly looking into the current state of projectional editing, there have been previous studies. The language workbench (LWB) challenges, which ran from 2011 to 2016, inspired many papers, most notably Erdweg et al.'s summaries of the 2013[125] and the 2015[21] challenges. These papers investigated the capabilities of LWBs, including projectional LWBs. Whilst these were interesting from the point of view of capabilities of LWBs, they did not touch on market penetration or other usage indicators, and thus the experimental workbenches were lined up equally against the well-used. Schindler et al.[61] looked at the experience of the language workbench challenge from the point of view of a projectional editing LWB, namely MPS. We addressed the areas where projectional editing LWBs had a significant advantage over the other LWBs. We feel it is a shame that the LWB challenge is no longer occurring.

There does not seem to be much analysis of what is going on in projectional editing currently, except when there is an intersection with adjacent fields such as Model-Driven Approaches, Low-code, Language Orientated Programming, or language workbenches. We found one systematic mapping study of LWBs[126] from 2020 that again looked at the entire field of LWBs, including the projectional ones. Whilst looking at LWB features, it also extended into areas such as domains of use.

Regarding MPS, papers from earlier in the last decade tended to concentrate on improving the experience of using MPS, prototypes or new products. Many of these are referenced in Section 2.2.3 and Section 2.2.4. Today, as mentioned in papers discussed in Section 3.2.3, research involving MPS tends to be maturing into industrial use of the product and using it in teaching.

6.2 Understandability of Business Rules

A different approach to ours was taken in the VODRE project[127]. In place of visualising how the rules belonged together, it visualised how they execute. In optical design, they showed the execution paths that the expert systems used to draw their conclusions.

In a similar fashion to the Drools DMN, Ostermayer et al.[128] attempt the generation of rules using templating and an external editor. Along the same lines, the G-AMC tool[129] created a front end for developing rules, in the domain of access control management. These both face the issues we were trying to overcome by separating the tool and the language. They also focus on the generation of rules rather than the understanding of them.

7

Conclusion

Although Drools rules are simple in isolation, they become difficult to reason about in aggregate. The goal of our work was to research if projectional editing could be successfully applied to alleviate this problem. To address this, we dove into the current state of projectional editing. We created, presented, and verified some projections to improve the understanding of Drools rules. We achieved this by conducting a systematic literature review (SLR), creating prototypes, and surveying the impact they may have.

We examined the current state of projectional editing through an SLR, finding it dominated by a single product. We implemented limited versions of the Drools language in MPS, and through our prototypes, we demonstrated some of the possibilities projectional editing can bring.

Our survey failed to confirm that these projections would bring the benefits we had initially expected. Although this may be a correct result, alternatively, we believe this could be a consequence of the responding developers' current experience of text-based development, leading to the text projection feeling more natural than the tabular projections. It may also be that the projections do not allow the level of complexity that text does. Finally, perhaps, a short survey was not the best way to introduce a whole new paradigm to people.

We described our work by first translating the Drools DSL into a projectional language, then exploring projections. We discussed the advantages and disadvantages of the different projections we created and analysed experienced developers' reactions to them.

This last section will return to the research questions we were trying to answer as described in Section 1.3. We will summarise the findings of our research and the contributions our work has made. Finally, we will discuss subjects that, though out of scope for this project, could lead to future research opportunities.

7.1 Research Summary

Our primary research question, "Is projectional editing a viable and effective solution to issues developers have with reasoning about Drools business rules?" motivated us to explore the field of projectional editing in this project. This question led us to examine if this field was worth investigating and, if so, how we can use existing tools to answer this.

7.1.1 Research Question 1: "What is the current state of language workbenches supporting projectional editing?"

We presented an SLR that investigated the current state of projectional editing. Our findings were that:

- LWBs for projectional editing are currently a very narrow field, dominated by one product, JetBrains MPS.
- Studies using products other than MPS spend time and effort discussing how to solve issues already solved in MPS.
- Studies using MPS tend to focus on industrial products or how to use MPS as a tool to teach metaprogramming.
- MPS is a little behind in developing a web-based environment, both for the language engineer and the developer.

We presented an SLR that investigated the current state of projectional editing. This SLR found that LWBs for projectional editing are currently a very narrow field, dominated by one product, JetBrains MPS. Studies using products other than MPS spend time and effort discussing how to solve issues already solved in MPS.

Studies using MPS tend to focus on industrial products or how to use MPS as a tool to teach metaprogramming. MPS is a little behind in developing a web-based environment, both for the language engineer and the developer.

A monoculture can be a risk. Whilst there are many advantages to projectional editing, having only one successful product and supplier feels a little unhealthy. On the other hand, the number of users of the tool is growing, as evidenced by the papers representing new projects in multiple industries, from hardware, through the automotive industry, to Finance.

To conclude, the state of the projectional editing market, whilst niche, is maturing but with a current product monoculture.

7.1.2 Research Question 2: "Which projections can we create to help developers get appropriate feedback about rules?"

In our research, we were able to develop some projections. This success was in large part facilitated by the flexibility and extensibility of the MPS tool, which presented the ability to develop and extend DSLs very efficiently.

We built two reduced implementations of Drools, on top of which we created seven projections of varying levels of complexity and usefulness. The projections we built within the Drools-lite language could implement large and complex rule sets with many features that aid understanding and development. Our research led us to two specific editing styles, the decision table and the Spreadsheet-like table.

We can conclude that the tooling is sufficient to create projections that can make large and complex rule sets easier to interact with and, at least for us, understand.

7.1.3 Research Question 3: "Do projections of Drools business rules lead to a greater understanding of those rules?"

We presented two projections to experienced Drools users, along with two wireframes of more exotic solutions. There was a distinct preference between the projections we presented, with the spreadsheet-like table being more understandable than our decision table. Both of our tabular projections were significantly less understandable to the survey subjects than the textual projection.

This performance difference could be because a textual presentation is inherently better than tabular or graphical ones. However, we speculate that the prior experience, both with textual languages in general and Drools in particular, coloured the results. We feel it would be interesting to run experiments on people with less textual programming experience.

While we could not show an advantage of our projections in our study, there was distinct interest in our approach and recognition of the advantages it may bring.

7.2 Summary of Contributions

This masters project makes the following contributions:

- *The Projectional Editing Status Quo* in our SLR, we provide an overview of the current state of research in the field of Projectional Editing
- *An alternative Base Language* with a bit of extra work, the Drools-Lite language can be used as an alternative base language for model-to-model transformations in MPS.
- *A new way to enter rules* our implementation allows Drools developers a more compact manner to enter rules.

7.3 Future work

Drools-Lite is open for future extension, matching the feature sets of the complete Drools language and adding extra projectional capabilities. Projections we would particularly like to take advantage of include graphical mappings of rule interactions and live test output.

Guarantees of completeness could help understanding business rules. We would be interested in applying the formal specifications developed within FASTEN[34] to some of our potential projections.

While a survey was informative, we would also like to run an experiment using our projections.

Finally, a question that has occurred to us both in the literature review and the survey was - how much of the opinions about projectional editing are a consequence of a history of textual language use? We briefly worked on a projectional implementation of the gradual pedagogical language Hedy[130]. Comparing the results of those with no programming experience to the use of projectional and text-based languages could point to the influence of experience in choice. This experiment, probably best performed in schools where Hedy is used, would have to wait until a web-based implementation of MPS is implemented, as installing an application on school machines is impractical.

Bibliography

[1] 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.

[2] 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, pp. 81–97, 1956.

[3] W. V. Quine, "Three grades of modal involvement," in *Proceedings of the XIth International Congress of Philosophy*, Cambridge University Press, vol. 14, 1953, pp. 65–81.

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

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

[6] P. Jackson, *Introduction to Expert Systems*. Addison-Wesley, 1986.

[7] E. H. Shortliffe, "MYCIN: A rule-based computer program for advising physicians regarding antimicrobial therapy selection," Stanford University, Tech. Rep., 1974.

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

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

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

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

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

[13] 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.

[14] V. Donzeau-Gouge, G. Huet, G. Kahn, and B. Lang, "Programming environments based on structured editors: The MENTOR experience," Institut National de Recherche en Informatique et en Automatique, Tech. Rep., 1980.

[15] R. Medina-Mora and P. H. Feiler, "An incremental programming environment," *IEEE Transactions on Software Engineering*, vol. SE-7, no. 5, pp. 472–482, 1981.

[16] D. Notkin, "The GANDALF project," *The Journal of Systems and Software*, vol. 5, no. 2, pp. 91–105, 1985.

[17] T. Teitelbaum and T. Reps, "The Cornell program synthesizer: A syntax-directed programming environment," *Communications of the ACM*, vol. 24, no. 9, pp. 563–573, 1981.

[18] T. W. Reps and T. Teitelbaum, *The Synthesizer Generator: A system for constructing language-based editors*. Springer, 2012.

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

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

[21] S. Erdweg, T. van der Storm, M. Völter, L. Tratt, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, *et al.*, "Evaluating and comparing language workbenches: Existing results and benchmarks for the future," *Computer Languages, Systems & Structures*, vol. 44, no. Part A, pp. 24–47, 2015.

[22] *Más post mortem*, https://medium.com/@dslmeinte/post-mortem-for-m%C3%A1s-aeca7542c4c8, Accessed: 2021-09-07.

[23] *Whole Platform*, https://whole.sourceforge.io/, Accessed: 2021-09-07.

[24] D. H. Lorenz and B. Rosenan, "Cedalion: A language for language oriented programming," in *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ACM, 2011, pp. 733–752.

[25] B. Hempel, J. Lubin, G. Lu, and R. Chugh, "Deuce: A lightweight user interface for structured editing," in *Proceedings of the 40th International Conference on Software Engineering*, IEEE/ACM, 2018, pp. 654–664.

[27] *Google Blockly product page*, https://developers.google.com/blockly/, Accessed: 2021-09-08.

[28] J. Klimeš, "Domain-specific language for learning programming," Master's thesis, Univerzita Karlova, Matematicko-fyzikálnıé fakulta, 2016.

[29] 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*, IEEE, 2017, pp. 83–92.

[30] 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*, ACM, 2010, pp. 301–304.

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

[32] 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*, 2021.

[33] M. Völter, 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.

[34] 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*, IEEE, 2019, pp. 41–50.

[35] D. Ratiu, B. Schaetz, M. Völter, 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*, IEEE, 2012, pp. 9–15.

[36] 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*, ACM, 2016, pp. 763–774.

[37] P. Vysoký, P. Parízek, and V. Pech, "INGRID: Creating languages in MPS from ANTLR grammars," Department of Distributed and Dependable Systems, Univerzita Karlova, Tech. Rep., 2018.

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

[39] M. Völter and S. Lißon, "Supporting diverse notations in MPS' projectional editor," in *2nd International Workshop on The Globalization of Modeling Languages*, ACM/IEEE, 2014, pp. 7–16.

[40] 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.

[41] M. Völter, A. van 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*, ACM, 2015, pp. 655–674.

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

[43] S. M. Guttormsen, A. Prinz, and T. Gjøsæter, "Consistent projectional text editors," in *International Conference on Model-Driven Engineering and Software Development*, Springer, 2017, pp. 515–522.

[44] M. Völter and B. Merkle, "Domain specific: A binary decision?" In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ACM, 2010, pp. 61–67.

[45] A. Wortmann and M. Beet, "Domain specific languages for efficient satellite control software development," in *Proceedings of the symposium Data Systems In Aerospace*, European Space Agency, 2016.

[46] M. Völter, 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.

[47] 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*, ACM, 2006, pp. 451–464.

[48] M. Völter, 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*, ACM, 2016, pp. 28–40.

[49] V. Pech, A. Shatalin, and M. Völter, "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*, ACM, 2013, pp. 165–168.

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

[51] D. Ratiu, H. Nehls, and J. Michel, "Taming the software development complexity with domain specific languages," in *Modellierung*, Gesellschaft für Informatik eV, 2018, pp. 281–292.

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

[53] M. Völter, 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.

[55] D. Pavletic, S. A. Raza, M. Völter, B. Kolb, and T. Kehrer, "Extensible debugger framework for extensible languages," in *Reliable Software Technologies – Ada-Europe 2015*, vol. 9111, Springer, 2015, pp. 33–49.

[56] M. Völter, "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.

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

[58] M. Völter, 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*, ACM, 2012, pp. 121–140.

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

[60] M. Völter, D. Ratiu, and F. Tomassetti, "Requirements as first-class citizens: Integrating requirements directly with implementation artifacts," in *Proceedings of Model-based Architecting of Cyber-Physical and Embedded System Workshop*, Linköping University Electronic Press, 2013.

[61] 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, Systems, Programming, Languages and Applications: Software for Humanity 2016*, ACM, 2016.

[62] A. Prinz, "Teaching language engineering using MPS," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 315–336.

[63] F. Tomassetti and V. Zaytsev, "Reflections on the lack of adoption of domain specific languages," in *Software Technologies: Applications and Foundations Workshops*, Springer, 2020, pp. 85–94.

[64] M. Fowler, *Language workbenches: The killer-app for domain specific languages?* `http://martinfowler.com/articles/languageWorkbench.html`, Accessed: 2021-02-02, 2005.

[65] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews*. CRC press, 2015.

[66] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 625–637, 2011.

[67] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ACM, 2014, pp. 314–324.

[68] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[69] A. O'Mara-Eves, J. Thomas, J. McNaught, M. Miwa, and S. Ananiadou, "Using text mining for study identification in systematic reviews: A systematic review of current approaches," *Systematic Reviews*, vol. 4, no. 1, pp. 1–22, 2015.

[70] S. Gregor, "The nature of theory in information systems," *Management Information Systems Quarterly*, pp. 611–642, 2006.

[71] M. Staron, "Action research vs. design research," in *Action Research in Software Engineering*, Springer International Publishing, 2019, pp. 141–151.

[72] I. K. Crombie and B. J. Harvey, *The pocket guide to critical appraisal: a handbook for health care professionals*. BMJ Publishing Group, 1996.

[100] F. Q. Da Silva, A. L. Santos, S. Soares, A. C. C. França, C. V. Monteiro, and F. F. Maciel, "Six years of systematic literature reviews in software engineering: An updated tertiary study," *Information and Software Technology*, vol. 53, no. 9, pp. 899–913, 2011.

[101] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[102] T. Greenhalgh and R. Peacock, "Effectiveness and efficiency of search methods in systematic reviews of complex evidence: Audit of primary sources," *The BMJ (British Medical Journal)*, vol. 331, no. 7524, pp. 1064–1065, 2005.

[103] *Axure product page*, `https://www.axure.com/`, Accessed: 2021-08-23.

[104] M. F. Cowlishaw, "LEXX—a programmable structured editor," *IBM Journal of Research and Development*, vol. 31, no. 1, pp. 73–80, 1987.

[105] A. Sarkar, "The impact of syntax colouring on program comprehension," in *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group*, ACM, 2015, pp. 49–58.

[106] D. Hou and D. M. Pletcher, "Towards a better code completion system by API grouping, filtering, and popularity-based ranking," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, ACM, 2010, pp. 26–30.

[107] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of fine-grained code change history," in *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, IEEE, 2013, pp. 119–126.

[108] E. W. Dijkstra, "The humble programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, 1972.

[109] U. W. Pooch, "Translation of decision tables," *ACM Computing Surveys*, vol. 6, no. 2, pp. 125–151, 1974.

[110] L. Bradley and K. McDaid, "Using Bayesian statistical methods to determine the level of error in large spreadsheets," in *2009 31st International Conference on Software Engineering-Companion Volume*, IEEE, 2009, pp. 351–354.

[111] L. Wittgenstein, *Tractatus logico-philosophicus*. Routledge, 2013.

[112] R. Lukyanenko, J. Evermann, and J. Parsons, "Instantiation validity in IS design research," in *Advancing the Impact of Design Science: Moving from Theory to Practice*, Springer, 2014, pp. 321–328.

[113] A. Bryman, *Social research methods*. Oxford University Press, 2016.

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

[115] R. Likert, *A technique for the measurement of attitudes*. New York University, 1932.

[116] 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.

[117] 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, pp. 879–903, 2003.

[118] 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, pp. 114–121, 2001.

[119] G. Gorrell, N. Ford, A. Madden, P. Holdridge, and B. Eaglestone, "Countering method bias in questionnaire-based user studies," *Journal of Documentation*, vol. 67, no. 3, pp. 507–524, 2011.

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

[121] N. B. Robbins, R. M. Heiberger, *et al.*, "Plotting Likert and other rating scales," in *Proceedings of the 2011 Joint Statistical Meeting*, vol. 1, 2011.

[122] J. Blasius and V. Thiessen, "The use of neutral responses in survey questions: An application of multiple correspondence analysis," *Journal of Official Statistics - Stockholm*, vol. 17, no. 3, pp. 351–368, 2001.

[123] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, pp. 50–60, 1947.

[124] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[125] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, *et al.*, "The state of the art in language workbenches," in *International Conference on Software Language Engineering*, Springer, 2013, pp. 197–217.

[126] A. Iung, J. Carbonell, L. Marchezan, E. Rodrigues, M. Bernardino, F. P. Basso, and B. Medeiros, "Systematic mapping study on domain-specific language development tools," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4205–4249, 2020.

[127] M. Lapaev and M. Kolchin, "VODRE: Visualisation of Drools rules execution," in *Proceedings of 15th Conference of Open Innovations Association*, IEEE, 2014, pp. 77–84.

[128] L. Ostermayer, G. Sun, and D. Seipel, "Simplifying the development of rules using domain specific languages in Drools," in *20th International Conference on Applications of Declarative Programming and Knowledge Management*, Christian-Albrechts-Universität zu Kiel, 2013, pp. 198–2012.

[129] J. Sá, S. Alves, and S. Broda, "The G-ACM tool: Using the Drools rule engine for access control management," *ArXiv*, 2016.

[130] F. Hermans, "Hedy: A gradual language for programming education," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ACM, 2020, pp. 259–270.

Systematic Literature Review Bibliography

[26] L.-E. Lafontant and E. Syriani, "Gentleman: A light-weight web-based projectional editor generator," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ACM/IEEE, 2020, pp. 1–5.

[54] 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.

[73] M. Völter, S. Košcejev, M. Riedel, A. Deitsch, and A. Hinkelmann, "A domain-specific language for payroll calculations: A case study at DATEV," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 93–130.

[74] J. Schröpfer, T. Buchmann, and B. Westfechtel, "A framework for projectional multi-variant model editors," in *International Conference on Model-Driven Engineering and Software Development*, Springer, 2021, pp. 294–305.

[75] J. Schröpfer, B. Westfechtel, and T. Buchmann, "A generic projectional editor for EMF models," in *International Conference on Model-Driven Engineering and Software Development*, Springer, 2020, pp. 381–392.

[76] A. Bucchiarone, K. Soysal, and C. Guidi, "A model-driven approach towards automatic migration to microservices," in *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Springer, 2019, pp. 15–36.

[77] L. Andersen, M. Ballantyne, and M. Felleisen, "Adding interactive visual syntax to textual code," *Proceedings of the ACM on Programming Languages*, vol. 4, pp. 1–28, 2020.

[78] L. Addazi and F. Ciccozzi, "Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment," *Journal of Systems and Software*, vol. 175, pp. 169–186, 2021.

[79] S. Meacham, V. Pech, and D. Nauck, "Classification algorithms framework (CAF) to enable intelligent systems using JetBrains MPS domain-specific languages environment," *IEEE Access*, vol. 8, pp. 14 832–14 840, 2020.

[80] A. L. Furtado, "DSL based approach for building model-driven questionnaires," in *Enterprise Information Systems: 22nd International Conference*, Springer, 2021, pp. 458–480.

[81] T. Beckmann, "Efficient editing in a tree-oriented projectional editor," in *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, ACM, 2020, pp. 215–216.

[82] D. Kolovos, A. de la Vega, and J. Cooper, "Efficient generation of graphical model views via lazy model-to-text transformation," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ACM/IEEE, 2020, pp. 12–23.

[83] A. Bucchiarone, A. Cicchetti, and A. Marconi, "Engineering gameful applications with MPS," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 227–258.

[84] D. Ratiu, A. Nordmann, P. Munk, C. Carlan, and M. Völter, "FASTEN: An extensible platform to experiment with rigorous modeling of safety-critical systems," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 131–164.

[85] J. Schröpfer and T. Buchmann, "Integrating UML and ALF: An approach to overcome the code generation dilemma in model-driven software engineering," in *International Conference on Model-Driven Engineering and Software Development*, Springer, 2019, pp. 1–26.

[86] A. L. Santos, "Javardise: A structured code editor for programming pedagogy in Java," in *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, ACM, 2020, pp. 120–125.

[87] E. Schindler, H. Moneva, J. van Pinxten, L. van Gool, B. van der Meulen, N. Stotz, and B. Theelen, "JetBrains MPS as core DSL technology for developing professional digital printers," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 53–91.

[88] M. Simi, "Learning data analysis with MetaR," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 259–290.

[89] N. Stotz and K. Birken, "Migrating insurance calculation rule descriptions from Word to MPS," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 165–194.

[90] P. Munk and A. Nordmann, "Model-based safety assessment with SysML and component fault trees: Application and lessons learned," *Software and Systems Modeling*, vol. 19, no. 4, pp. 889–910, 2020.

[91] A. Bucchiarone, M. Savary-Leblanc, X. L. Pallec, J.-M. Bruel, A. Cicchetti, J. Cabot, S. Gerard, H. Aslam, A. Marconi, and M. Perillo, "Papyrus for gamers, let's play modeling," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ACM/IEEE, 2020, pp. 21–25.

[92] M. V. Merino, J. Bartels, M. van den Brand, T. van der Storm, and E. Schindler, "Projecting textual languages," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 197–225.

[93] R. Cuinat, C. Teodorov, and J. Champeau, "SpecEdit: Projectional editing for TLA+ specifications," in *2020 IEEE Workshop on Formal Requirements*, IEEE, 2020, pp. 1–7.

[94] A. Prinz, "Teaching language engineering using MPS," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 315–336.

[95] M. Barash and V. Pech, "Teaching MPS: Experiences from industry and academia," in *Domain-Specific Languages in Practice*, Springer, 2021, pp. 293–313.

[96] B. Hempel and R. Chugh, "Tiny structure editors for low, low prices! (generating GUIs from toString functions)," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE, 2020.

[97] E. Negm, S. Makady, and A. Salah, "Towards ontology-based domain specific language for internet of things," in *Proceedings of the 2020 9th International Conference on Software and Information Engineering*, ACM, 2020, pp. 146–151.

[98] J. Lubin and R. Chugh, "Type-directed program transformations for the working functional programmer," in *10th Workshop on Evaluation and Usability of Programming Languages and Tools*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, pp. 1–12.

[99] M. Ozkaya and D. Akdur, "What do practitioners expect from the meta-modeling tools? a survey," *Journal of Computer Languages*, vol. 63, pp. 1–21, 2021.

A

Systematic Literature Review Log

Search Description

We show the papers we found with our automatic search in Table A.2. From the 173 total papers found with the search strings across all venues when we used the search string and date restrictions, there were 100 unique papers. First, we excluded those not in English (5) or unavailable to us (3), reducing the count to 92. Next, we downloaded each of the papers.

Next, we skimmed all of these papers to remove any unrelated to projectional editing (24), reducing the count to 68.

Two of the papers were referring to the same study, which reduced the count to 66.

We then excluded all grey literature (14), i.e., masters projects, proposals and PhD theses, and books (2), reducing the count to 50 papers.

Table description

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

The First column, "Paper Title", is for the paper's name returned by the search engine.

The second column, "Venue", indicates which library or search engine found the paper. Table A.1 maps the keys to the venues.

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 columns F# and B# represent the count of papers harvested from these papers for the first forward and backwards snowballing iteration.

Key	Search engine/library	Key	Search engine/library
1	Google Scholar	2	IEEExplores
3	ACM	4	BASE
5	CORE	6	Web of Science
7	Microsoft Academic	8	SCOPUS
9	Semantic Scholar	10	SpringerLink
11	Wiley Online	12	Science.gov

Table A.1: Search engine/library key

Paper Title	Venue	Included	Excluded	F#	B#
“Filmar, assistir e problematizar” – contribuições à aprendizagem de cálculos	9		not English	X	X
20. Internationales Stuttgarter Symposium	10		book	X	X
A Domain-Specific Language for Payroll Calculations: a Case Study at DATEV	1	✓		0	0
A Domain-Specific Language for Payroll Calculations: An Experience Report from DATEV	1,10	✓	Duplicate	X	X
A Framework for Modernizing Domain-Specific Languages	1	✓	grey	X	X
A Framework for Projectional Multi-variant Model Editors	1,8	✓		0	0
A Generic Projectional Editor for EMF Models	1,7,8,9	✓		2	0
A language-driven Development framework for simulation components to generate simulated environments	1	✓	grey	X	X
A Model-Driven Approach Towards Automatic Migration to Microservices	10	✓		5	0
A survey of Model Driven Engineering in robotics	1	✓		2	2
A Survey on the Design Space of End-User Oriented Languages for Specifying Robotic Missions	1,10	✓		1	5
A survey on the formalisation of system requirements and their validation	1	✓		0	0
A text-based syntax completion method using LR parsing	1			X	X
Activities and costs of re-engineering cloned variants into an integrated platform	1			X	X
AdaptiveVLE: An Integrated Framework for Personalized Online Education Using MPS JetBrains Domain-Specific Modeling Environment	1,2	✓		1	1
Adding Interactive Visual Syntax to Textual Code	3	✓		3	0
An approach to generate text-based IDEs for syntax completion based on syntax specification	1	✓		1	0
An MPS implementation for SimpliC	1	✓	grey	x	x
Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment	1	✓		0	0
Block-based syntax from context-free grammars	1,3,5	✓		0	2
Bridging the worlds of textual and projectional language workbenches	1	✓	grey	X	X
Classification Algorithms Framework (CAF) to Enable Intelligent Systems Using JetBrains MPS Domain-Specific Languages Environment	2,5	✓		4	0
Code and Structure Editing for Teaching: A Case Study in using Bibliometrics to Guide Computer Science Research	1,9	✓		2	0
ComPOS - a Domain-Specific Language for Composing Internet-of-Things Systems	1,4	✓	grey	X	X
Concepts of variation control systems	1	✓		9	5
Concise, Type-Safe, and Efficient Structural Diffing	3			X	X

Paper Title	Venue	Included	Excluded	F#	B#
Constructing optimized constraint-preserving application conditions for model transformation rules	1			X	X
Design & Evaluation of an Accessible High-Level Language for Advanced Cryptography	1	✓	grey	X	X
Domain-specific languages for modeling and simulation	1			X	X
Domain-Specific Languages in Practice	10	✓	book	X	X
DSL Based Approach for Building Model-Driven Questionnaires	1,10	✓		0	1
DSS-Based Ontology Alignment in Solid Reference System Configuration	10		unavailable	X	X
Editing Software as Strategy Value	1			X	X
Efficient editing in a tree-oriented projectional editor	1,3,7,8,9	✓		1	0
Efficient generation of graphical model views via lazy model-to-text transformation	1,4	✓		1	0
Efficient usage of abstract scenarios for the development of highly-automated driving functions	1,10	✓	unavailable	X	X
Enabling language engineering for the masses	1,3	✓		2	1
Engineering Gameful Applications with MPS	1,10	✓		0	2
Enhancing development and consistency of UML models and model executions with USE studio	1,3,7,8,9	✓		0	2
Enterprise Information Systems	10		book	X	X
Example-driven software language engineering	1,3	✓		1	2
Exploring Visual Primitives for Authoring Source Code	1		grey	X	X
FASTEN: An Extensible Platform to Experiment with Rigorous Modeling of Safety-Critical Systems	1,10	✓		0	5
FeatureCoPP: unfolding preprocessor variability	1,3			X	X
FeatureVista: Interactive Feature Visualization	1			X	X
Filling Typed Holes with Live GUIs	3	✓		0	5
First-class concepts: reifying architectural knowledge beyond the dominant decomposition	1,3	✓		0	1
FORMREQ 2020	1,2,8		book	X	X
Gentleman: a light-weight web-based projectional editor generator	1,3,4,7,8,9	✓		0	0
GPP, the Generic Preprocessor	1			X	X
Improving the usability of the domain-specific language editors using artificial intelligence	1	✓	grey	X	X
Incremental Flow Analysis through Computational Dependency Reification	1,2	✓		0	2
Incrementalizing Static Analyses in Datalog	1	✓	grey	X	X
Integrating the Common Variability Language with Multilanguage Annotations for Web Engineering	1			X	X
Integrating UML and ALF: An Approach to Overcome the Code Generation Dilemma in Model-Driven Software Engineering	10	✓		0	0

Paper Title	Venue	Included	Excluded	F#	B#
Javardise: a structured code editor for programming pedagogy in Java	1	✓		0	0
JetBrains MPS as Core DSL Technology for Developing Professional Digital Printers	1,10	✓		0	0
JetBrains MPS: Why Modern Language Workbenches Matter	1,7,10	✓		0	1
Learning Data Analysis with MetaR	1,10	✓		0	0
Lipschitz-like property relative to a set and the generalized Mordukhovich criterion	6			X	X
Macros for Domain-Specific Languages	3			X	X
Mechanizing metatheory interactively	1	✓	grey	X	X
Migrating Insurance Calculation Rule Descriptions from Word to MPS	1,10	✓		0	0
Model Driven Software Engineering Meta-Workbenches: An XTools Approach	1,5	✓		0	0
Model-based safety assessment with SysML and component fault trees: application and lessons learned	1,10	✓		8	0
Model-Driven Development for Spring Boot Microservices	1,5	✓	grey	X	X
n Challenges for Software Language Engineering	1			X	X
On preserving variability consistency in multiple models	1,3			X	X
On the Need for a Formally Complete and Standardized Language Mapping between C++ and UML	1			X	X
On the Understandability of Language Constructs to Structure the State and Behavior in Abstract State Machine Specifications: A Controlled Experiment	1			X	X
On the use of product-line variants as experimental subjects for clone-and-own research: a case study	1			X	X
PAMOJA: A component framework for grammar-aware engineering	1	✓		0	1
Programming Robots for Activities of Everyday Life	1	✓	grey	X	X
Programming tools for intelligent systems	1	✓	grey	X	X
Projecting Textual Languages	1,10	✓		0	1
Rule-based and user feedback-driven decision support system for transforming automatically-generated alignments into information-integration alignments	5		unavailable	X	X
Semi-Automatische Deduktion von Feature-Lokalisierung während der Softwareentwicklung: Master-arbeit	5		not English	X	X
Should Variation Be Encoded Explicitly in Databases?	1			X	X
SLang: A Domain-specific Language for Survey Questionnaires	1	✓		0	0
SpecEdit: Projectional Editing for TLA+ Specifications	1,4,7	✓		0	0
Specifying Software Languages: Grammars, Projectional Editors, and Unconventional Approaches	1,2,4,5,7,8,9	✓		0	6
Teaching Language Engineering Using MPS	10	✓		0	0

Paper Title	Venue	Included	Excluded	F#	B#
Teaching MPS: Experiences from Industry and Academia	1,10	✓		0	1
Teasy framework: uma solução para testes automatizados em aplicações web	1	✓	not English	X	X
The Art of Bootstrapping	10	✓		3	0
The state of adoption and the challenges of systematic variability management in industry	1			X	X
Toward a domain-specific language for scientific workflow-based applications on multicloud system	1,11			X	X
Towards a Universal Variability Language	1	✓	grey	X	X
Towards Multi-editor Support for Domain-Specific Languages Utilizing the Language Server Protocol	10	✓		5	0
Towards Ontology-based Domain Specific Language for Internet of Things	1,3	✓		0	0
Towards projectional editing for model-based SPLs	3,4,7,8,9	✓		3	0
Tychonis: A model-based approach to define and search for geometric events in space	1			X	X
Type-Directed Program Transformations for the Working Functional Programmer	1	✓		0	0
Understanding Variability-Aware Analysis in Low-Maturity Variant-Rich Systems	1			X	X
Untangling Mechanized Proofs	3			X	X
Variability representations in class models: An empirical assessment	1,3			X	X
Visual design for a tree-oriented projectional editor	1,3,4,7,8,9	✓	Duplicate	X	X
What do practitioners expect from the meta-modeling tools? A survey	1,7,8,9	✓		0	3
Cyrillic named paper 1	1		not English	X	X
Cyrillic named paper 2	1		not English	X	X

Table A.2: Systematic literature review log - search results

B

Study Quality Assessment Checklist

The Center for Evidence-Based Management (CEBMa) supports applying evidence-based practices to management and leadership. They have a collection of checklists for assessing different types of studies. We adapted these checklists from the Pocket Guide to Critical Appraisal[72]. We gave used these as the basis of our quality assessment checklists.

Critical Appraisal of a Case Study

#	Appraisal questions	Yes	Can't tell	No
1	Did the study address a focused question/issue?			
2	Is the research method (study design) appropriate for answering the research question?			
3	Are both the setting and the subject's representative concerning the population to which the findings will be referred?			
4	Is the researcher's perspective clearly described and taken into account?			
5	Are the methods for collecting data clearly described?			
6	Are the methods for analysing the data likely to be valid and reliable? Are quality-control measures used?			
7	Did more than one researcher repeat the analysis to ensure reliability?			
8	Are the results credible, and if so, are they relevant for practice?			
9	Are the conclusions drawn justified by the results?			
10	Are the findings of the study transferable to other settings?			

Table B.1: Case studies quality assessment checklist

Critical Appraisal of a Qualitative Study

#	Appraisal questions	Yes	Can't tell	No
1	Did the study address a focused question/issue?			
2	Is the research method (study design) appropriate for answering the research question?			
3	Was the context clearly described?			
4	How was the fieldwork undertaken? Was it described in detail? Are the methods for collecting data clearly described?			
5	Could the evidence (fieldwork notes, interview transcripts, recordings, documentary analysis, etc.) be inspected independently?			
6	Are the procedures for data analysis reliable and theoretically justified? Are quality-control measures used?			
7	Did more than one researcher repeat the analysis to ensure reliability?			
8	Are the results credible, and if so, are they relevant for practice?			
9	Are the conclusions drawn justified by the results?			
10	Are the findings of the study transferable to other settings?			

Table B.2: Qualitative studies quality assessment checklist

Critical Appraisal of a Survey Study

#	Appraisal questions	Yes	Can't tell	No
1	Did the study address a focused question/issue?			
2	Is the research method (study design) appropriate for answering the research question?			
3	Is the method of selecting the subjects (employees, teams, divisions, organisations) clearly described?			
4	Could the way the sample was obtained introduce (selection) bias?			
5	Was the sample of subjects representative concerning the population to which the findings will be referred?			
6	Was the sample size based on pre-study considerations of statistical power?			
7	Was a satisfactory response rate achieved?			
8	Are the measurements (questionnaires) likely to be valid and reliable?			
9	Was the statistical significance assessed?			
10	Are confidence intervals given for the main results?			
11	Could there be confounding factors that have not been accounted for?			
12	Are the findings of the study transferable to other settings?			

Table B.3: Survey studies quality assessment checklist

Critical Appraisal of a Cohort or Panel Study

#	Appraisal questions	Yes	Can't tell	No
1	Did the study address a focused question/issue?			
2	Is the research method (study design) appropriate for answering the research question?			
3	Were there enough subjects (employees, teams, divisions, organisations) in the study to establish that the findings did not occur by chance?			
4	Was the selection of the cohort/panel based on external, objective, and validated criteria?			
5	Was the cohort/panel representative of a defined population?			
6	Was the follow up of cases/subjects long enough?			
7	Were objective and unbiased outcome criteria used?			
8	Are objective and validated measurement methods used to measure the outcome?			
9	Is the size effect practically relevant?			
10	How precise is the estimate of the effect? Were confidence intervals given?			
11	Could there be confounding factors that have not been accounted for?			
12	Are the findings of the study transferable to other settings?			

Table B.4: Cohort or panel studies quality assessment checklist

C

Study Quality Assessment Results

In this appendix, we present the data for the findings of the quality assessment stage of the SLR. After the initial search engine selections, the three snowballing iterations and the final deep read for classification.

This table only reports on primary studies. When a paper reports on more than one study, the paper title appears multiple times, with the study type in parenthesis. If we had multiple papers reporting on the same study, we have already removed them.

We separated the studies into their types. When the authors self-reported a type, even if we disagreed with their categorisation, we categorised them as such. The study types were survey, case study, Design science research, and qualitative study. For the sake of table width, we refer to Design science research as DSR.

The question assessments use the assessment criteria presented in Appendix B. However, these criteria do not have a checklist for design science research. After much research, we did not find a good checklist for DSR, so we used the case study checklist. After concluding the quality assessment, we had to conclude that this checklist was not a valid interrogation of DSR studies or that all 20 DSR studies were poor.

To score the studies, we arbitrarily decided to give a +1 value for positive answers, 0 for don't know and -1 for negative answers. We understand that this is an overly simple system.

While most questions answered with "Yes" were considered positive, in the survey checklist, the question "Could the way the sample was obtained introduce (selection)bias?", the positively scored answer is "No".

Name	Type	Score	1	2	3	4	5	6	7	8	9	10	11	12
A Domain-Specific Language for Payroll Calculations: a Case Study at DATEV[73]	Case Study	3	Y	Y	Y	?	N	?	N	?	Y	Y	-	-
A Framework for Projectional Multi-variant Model Editors[74]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
A Generic Projectional Editor for EMF Models[75]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
A Model-Driven Approach Towards Automatic Migration to Microservices[76]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
AdaptiveVLE: An Integrated Framework for Personalized Online Education Using MPS JetBrains Domain-Specific Modeling Environment[54]	DSR	3	N	?	Y	?	Y	Y	N	Y	Y	?	-	-
Adding Interactive Visual Syntax to Textual Code[77]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiments[78]	Qualitative Study	8	Y	Y	Y	Y	?	Y	?	Y	Y	Y	-	-
Block-based syntax from context-free grammars[78]	Case Study	-3	N	?	?	N	N	?	N	?	Y	?	-	-
Classification Algorithms Framework (CAF) to Enable Intelligent Systems Using JetBrains MPS Domain-Specific Languages Environment[79]	DSR	3	N	?	Y	?	Y	Y	N	Y	Y	?	-	-
DSL Based Approach for Building Model-Driven Questionnaires (Design Research)[80]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
DSL Based Approach for Building Model-Driven Questionnaires (Qualitative Study1)[80]	Qualitative Study	-2	N	?	Y	N	?	?	N	?	?	?	-	-
DSL Based Approach for Building Model-Driven Questionnaires (Qualitative Study2)[80]	Qualitative Study	-2	N	?	Y	N	?	?	N	?	?	?	-	-
Efficient editing in a tree-oriented projectional editor[81]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
Efficient generation of graphical model views via lazy model-to-text transformation[82]	DSR	0	N	?	?	N	Y	Y	N	Y	?	?	-	-
Engineering Gameful Applications with MPS[83]	DSR	-6	N	?	N	N	N	?	N	?	?	N	-	-
FASTEN: An Extensible Platform to Experiment with Rigorous Modeling of Safety-Critical Systems[84]	DSR	-4	N	?	N	?	N	?	N	?	?	?	-	-
Gentleman: a light-weight web-based projectional editor generator[26]	DSR	-5	N	?	N	?	N	?	N	?	?	N	-	-
Integrating UML and ALF: An Approach to Overcome the Code Generation Dilemma in Model-Driven Software Engineering[85]	DSR	-5	N	?	N	?	N	?	N	?	?	N	-	-
Javardise: a structured code editor for programming pedagogy in Java[86]	DSR	-5	N	?	N	?	N	?	N	?	?	N	-	-
JetBrains MPS as Core DSL Technology for Developing Professional Digital Printers[87]	Case Study	-6	N	?	N	N	N	?	N	?	?	N	-	-

Name	Type	Score	1	2	3	4	5	6	7	8	9	10	11	12
Learning Data Analysis with MetaR[88]	DSR	-4	N	?	Y	N	N	?	N	?	?	N	-	-
Migrating Insurance Calculation Rule Descriptions from Word to MPS[89]	Case Study	-3	N	?	Y	N	N	?	N	?	?	?	-	-
Model-based safety assessment with SysML and component fault trees: application and lessons learned (Case study1)[90]	Case Study	-4	Y	?	N	N	N	?	N	?	?	N	-	-
Model-based safety assessment with SysML and component fault trees: application and lessons learned (Case study2)[90]	Case Study	-2	Y	?	Y	N	N	?	N	?	?	N	-	-
Model-based safety assessment with SysML and component fault trees: application and lessons learned (Design Research)[90]	DSR	-3	N	?	Y	N	N	?	N	?	?	?	-	-
Papyrus for gamers, let's play modeling[91]	DSR	-4	N	?	?	N	N	?	N	?	?	?	-	-
Projecting Textual Languages (Design Research)[92]	DSR	-5	N	?	N	N	N	?	N	?	?	?	-	-
Projecting Textual Languages (Case Study)[92]	Case Study	-6	N	?	N	N	N	?	N	?	?	N	-	-
SpecEdit: Projectional Editing for TLA+ Specifications (Design Research)[93]	DSR	-2	Y	?	?	N	N	?	N	?	?	?	-	-
SpecEdit: Projectional Editing for TLA+ Specifications (Case Study)[93]	Case Study	-5	N	?	N	N	N	?	N	?	?	?	-	-
Teaching Language Engineering Using MPS[94]	Case Study	3	N	?	Y	Y	N	?	N	Y	Y	Y	-	-
Teaching MPS: Experiences from Industry and Academia[95]	Case Study	0	N	?	Y	Y	N	Y	N	?	?	?	-	-
Tiny Structure Editors for Low, Low Prices (Design Research)[96]	DSR	-5	N	?	?	N	N	?	N	?	?	N	-	-
Tiny Structure Editors for Low, Low Prices (Case Study)[96]	Case Study	-6	N	?	N	N	N	?	N	?	?	N	-	-
Towards Ontology-based Domain Specific Language for Internet of Things[97]	DSR	-6	N	?	?	N	N	?	N	?	?	N	-	-
Type-Directed Program Transformations for the Working Functional Programmer[98]	DSR	-3	Y	Y	N	N	N	?	N	?	?	N	-	-
What do practitioners expect from the meta-modeling tools? A survey[99]	Survey	1	Y	Y	Y	Y*	?	?	Y	Y	N	N	?	N

Table C.1: Quality assessment results

D

Data Extraction Results

Study ID	1
Title of Study	A domain-specific language for payroll calculations: A case study at DATEV
Year of Publication	2021
Author(s) Names	M. Voelter, S. Koščejev, M. Riedel, A. Deitsch, and A. Hinkelmann
Source of Study	Google Scholar, SpringerLink
Type of Study	Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	generallearnings -- negative = 23, neutral = 28, positive = 13 evaluation -- negative = 45, neutral = 69, positive = 23 conclusion -- negative = 3, neutral = 10, positive = 4

Study ID	2
Title of Study	A framework for projectional multi-variant model editors
Year of Publication	2021
Author(s) Names	J. Schröpfer, T. Buchmann, and B. Westfecht
Source of Study	Google Scholar, SCOPUS
Type of Study	Design Science Research
Name of Venue	MODELSWARD
Tools in Study	EMF & Ecore
Sentiment	intro -- negative = 2, neutral = 23, positive = 2 conclusion -- negative = 2, neutral = 9, positive = 3

Study ID	3
Title of Study	A generic projectional editor for EMF models
Year of Publication	2020
Author(s) Names	J. Schröpfer, T. Buchmann, and B. Westfecht
Source of Study	Google Scholar, Microsoft Academic, SCOPUS, Semantic Scholar
Type of Study	Design Science Research
Name of Venue	MODELSWARD
Tools in Study	EMF & Ecore
Sentiment	intro -- negative = 12, neutral = 43, positive = 8 conclusion -- negative = 2, neutral = 6, positive = 1

Study ID	4
Title of Study	A model-driven approach towards automatic migration to microservices
Year of Publication	2020
Author(s) Names	A. Bucchiarone, K. Soysal, and C. Guidi
Source of Study	SpringerLink
Type of Study	Design Science Research
Name of Venue	International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 1, neutral = 15, positive = 3 conclusion -- neutral = 4

Figure D.1: Data extraction results 1 - 4

Study ID	5
Title of Study	AdaptiveVLE: An integrated framework for personalized online education using MPS JetBrains domain-specific modeling environment
Year of Publication	2020
Author(s) Names	S. Meacham, V. Pech, and D. Nauck
Source of Study	Google Scholar, IEEExplores
Type of Study	Design Science Research
Name of Venue	IEEE Access
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 3, neutral = 17, positive = 2 conclusion -- negative = 2, neutral = 3, positive = 3

Study ID	6
Title of Study	Adding interactive visual syntax to textual code
Year of Publication	2020
Author(s) Names	L. Andersen, M. Ballantyne, and M. Felleisen
Source of Study	ACM
Type of Study	Design Science Research
Name of Venue	Proceedings of the ACM on Programming Languages OOPSLA
Tools in Study	Racket and DrRacket
Sentiment	conclusion -- negative = 1, neutral = 14, positive = 4

Study ID	7
Title of Study	Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment
Year of Publication	2021
Author(s) Names	L. Addazi and F. Ciccozzi
Source of Study	Google Scholar
Type of Study	Qualitative Study
Name of Venue	Journal of Systems and Software
Tools in Study	Xtext & Ecore
Sentiment	intro -- negative = 6, neutral = 21, positive = 5 projectionalediting -- neutral = 6 discussion -- negative = 18, neutral = 26, positive = 6 conclusion -- negative = 1, neutral = 5, positive = 4

Study ID	8
Title of Study	Classification algorithms framework (CAF) to enable intelligent systems using JetBrains MPS domain-specific languages environment
Year of Publication	2020
Author(s) Names	S. Meacham, V. Pech, and D. Nauck
Source of Study	IEEExplores, CORE
Type of Study	Design Science Research
Name of Venue	IEEE Access
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 2, neutral = 19 evaluation -- negative = 3, neutral = 10, positive = 10 conclusion -- negative = 1, neutral = 6

Study ID	9
Title of Study	DSL based approach for building model-driven questionnaires
Year of Publication	2020
Author(s) Names	A. L. Furtado
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research + Qualitative Study
Name of Venue	Enterprise Information Systems: 22nd International Conference ICEIS 2020
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 4, neutral = 11, positive = 3 Implementations -- negative = 4, neutral = 16 conclusion -- negative = 3, neutral = 5, positive = 2

Figure D.2: Data extraction results 5 - 9

Study ID	10
Title of Study	Efficient editing in a tree-oriented projectional editor
Year of Publication	2020
Author(s) Names	T. Beckmann
Source of Study	Google Scholar, ACM, Microsoft Academic,SCOPUS, Semantic Scholar
Type of Study	Design Science Research
Name of Venue	Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming
Tools in Study	sandblocks (Research Project)
Sentiment	intro -- negative = 1, neutral = 7, positive = 2 design -- neutral = 9 conclusion -- negative = 1, neutral = 1, positive = 1

Study ID	11
Title of Study	Efficient generation of graphical modelviews via lazy model-to-text transformation
Year of Publication	2020
Author(s) Names	D. Kolovos, A. De La Vega, and J. Cooper
Source of Study	Google Scholar, BASE
Type of Study	Design Science Research
Name of Venue	Proceedings of the 23rd ACM/IEEEInternational Conference on Model Driven Engineering Languages and Systems
Tools in Study	Picto (Research Project using EGL, Graphviz,PlantUML)
Sentiment	intro -- negative = 3, neutral = 5, positive = 1 evaluation -- negative = 5, neutral = 43, positive = 1 results -- negative = 5, neutral = 34, positive = 5 conclusion -- negative = 1, neutral = 3, positive = 3

Study ID	13
Title of Study	Engineering gameful applications with MPS
Year of Publication	2021
Author(s) Names	A. Bucchiarone, A. Cicchetti, and A. Marconi
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 6, neutral = 20, positive = 5 engineering -- negative = 2, neutral = 34, positive = 7 mpsprojectional -- negative = 2, neutral = 18 lessonslearned -- negative = 11, neutral = 16, positive = 5 conclusion -- negative = 1, neutral = 5, positive = 2

Study ID	15
Title of Study	Fasten: An extensible platform to experiment with rigorous modeling of safety-critical systems
Year of Publication	2021
Author(s) Names	D. Ratiu, A. Nordmann, P. Munk, C. Carlan, and M. Voelter
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 18, neutral = 40, positive = 11 platform -- neutral = 13, positive = 2 discussion -- negative = 3, neutral = 21, positive = 2 discussionMPS -- negative = 14, neutral = 25, positive = 8 conclusion -- negative = 3, neutral = 11, positive = 4

Study ID	16
Title of Study	Gentleman: A light-weight web-based projectional editor generator
Year of Publication	2020
Author(s) Names	L.E. Lafontant and E. Syriani
Source of Study	Google Scholar, ACM, BASE, Microsoft Academic, SCOPUS, Semantic Scholar
Type of Study	Design Science Research
Name of Venue	Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings
Tools in Study	Gentleman (Research Project)
Sentiment	intro -- negative = 8, neutral = 12, positive = 2 editor -- negative = 5, neutral = 43, positive = 5 implementation -- neutral = 10, positive = 1 projections -- neutral = 40, positive = 6 conclusion -- neutral = 4, positive = 2

Figure D.3: Data extraction results 10 - 16

Study ID	17
Title of Study	Integrating UML and ALF: An approach to overcome the code generation dilemma in model-driven software engineering
Year of Publication	2020
Author(s) Names	J. Schröpfer and T. Buchmann
Source of Study	SpringerLink
Type of Study	Design Science Research
Name of Venue	International Conference on Model-Driven Engineering and Software Development
Tools in Study	ALF (Xtext) + Valkyrie (GMF)
Sentiment	intro -- negative = 4, neutral = 24, positive = 1 discussion -- negative = 4, neutral = 12, positive = 5 UI -- neutral = 9, positive = 2 conclusion -- neutral = 5

Study ID	18
Title of Study	Javardise: A structured code editor for programming pedagogy in Java
Year of Publication	2020
Author(s) Names	A. L. Santos
Source of Study	Google Scholar
Type of Study	Design Science Research
Name of Venue	Conference Companion of the 4th International Conference on Art, Science,and Engineering of Programming
Tools in Study	Javardise (Research Project)
Sentiment	discussion -- negative = 16, neutral = 13, positive = 4 intro -- negative = 11, neutral = 11, positive = 1

Study ID	19
Title of Study	Jetbrains MPS as core DSL technology for developing professional digital printers
Year of Publication	2021
Author(s) Names	E. Schindler, H. Moneva, J. van Pinxten, L. van Gool, B. van der Meulen, N. Stotz,and B. Theelen
Source of Study	Google Scholar, SpringerLink
Type of Study	Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	CollaborativeDSM -- negative = 5, neutral = 29, positive = 13 conclusion -- negative = 2, neutral = 22, positive = 2 intro -- negative = 1, neutral = 21, positive = 3 MPS -- negative = 9, neutral = 9, positive = 3

Study ID	20
Title of Study	Learning data analysis with metaR
Year of Publication	2021
Author(s) Names	M. Simi
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 1, neutral = 4, positive = 1 languagecompsition -- neutral = 22, positive = 3 MPS -- negative = 9, neutral = 56, positive = 6 conclusion -- negative = 2, neutral = 4, positive = 5

Study ID	21
Title of Study	Migrating insurance calculation rule descriptions from Word to MPS
Year of Publication	2021
Author(s) Names	N. Stotz and K. Birken
Source of Study	Google Scholar, SpringerLink
Type of Study	Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	intro -- neutral = 5, positive = 1 solutiontechnology -- negative = 20, neutral = 26, positive = 11 evaluation -- negative = 37, neutral = 49, positive = 10 conclusion -- negative = 9, neutral = 8, positive = 7

Figure D.4: Data extraction results 17 - 21

Study ID	22
Title of Study	Model-based safety assessment with sysml and component fault trees: Application and lessons learned
Year of Publication	2020
Author(s) Names	P. Munk and A. Nordmann
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research + Case Study
Name of Venue	Software and Systems Modeling
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 8, neutral = 11, positive = 6 realisation -- negative = 29, neutral = 46, positive = 11 discussion -- negative = 6, neutral = 4, positive = 2 conclusion -- negative = 4, neutral = 3, positive = 1

Study ID	23
Title of Study	Papyrus for gamers, let's play modeling
Year of Publication	2020
Author(s) Names	A. Bucchiarone, M. Savary-Leblanc, X. L. Pallec, J.M. Bruel, A. Cicchetti, J. Cabot,S. Gerard, H. Aslam, A. Marconi, and M. Perillo
Source of Study	snowball
Type of Study	Design Science Research
Name of Venue	Proceedings of the 23rd ACM/IEEE International Conference on Model-Driven Engineering Languages and Systems: Companion Proceedings
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 5, neutral = 8, positive = 3

Study ID	24
Title of Study	Projecting textual languages
Year of Publication	2021
Author(s) Names	M. V. Merino, J. Bartels, M. van den Brand, T. van der Storm, and E. Schindler
Source of Study	Google Scholar, SpringerLink
Type of Study	Design Science Research + Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS + Rascal
Sentiment	intro -- negative = 9, neutral = 17, positive = 3 projectionalApproach-- negative = 5, neutral = 128, positive = 4 projectionalSyntax -- negative = 4, neutral = 42 limitation -- negative = 10, neutral = 25, positive = 6 discussion -- negative = 6, neutral = 16, positive = 9 conclusion -- negative = 4, neutral = 14, positive = 5

Study ID	25
Title of Study	SpecEdit: Projectional editing for TLA+ specifications
Year of Publication	2020
Author(s) Names	R. Cuinat, C. Teodorov, and J. Champeau
Source of Study	Google Scholar, BASE,Microsoft Academic
Type of Study	Design Science Research + Case Study
Name of Venue	2020 IEEE Workshop on Formal Requirements (FORMREQ)
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 4, neutral = 16, positive = 6 projectionalEditor -- negative = 7, neutral = 78, positive = 17 lessonslearned -- negative = 4, neutral = 6, positive = 4 discussion -- neutral = 5, positive = 1

Study ID	26
Title of Study	Teaching language engineering using MPS
Year of Publication	2021
Author(s) Names	A. Prinz
Source of Study	SpringerLink
Type of Study	Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 6, neutral = 25, positive = 8 lessonslearned -- negative = 2, neutral = 12, positive = 2 metalanguages -- negative = 11, neutral = 25, positive = 11 MPSinTeaching -- negative = 14, neutral = 38, positive = 9 selectingTools -- negative = 6, neutral = 11, positive = 6 evaluation -- negative = 9, neutral = 30, positive = 20 conclusion -- negative = 2, neutral = 3, positive = 3

Figure D.5: Data extraction results 22 - 26

Study ID	27
Title of Study	Teaching MPS: Experiences from industry and academia
Year of Publication	2021
Author(s) Names	M. Barash and V. Pech
Source of Study	Google Scholar, SpringerLink
Type of Study	Case Study
Name of Venue	Domain-Specific Languages in Practice
Tools in Study	Jetbrains MPS
Sentiment	explainingprojectional -- negative = 1, neutral = 6, positive = 1 conclusion -- negative = 1, neutral = 5, positive = 1

Study ID	28
Title of Study	Tiny structure editors for low, low prices! (generating guis from toString functions)
Year of Publication	2020
Author(s) Names	B. Hempel and R. Chugh
Source of Study	snowball
Type of Study	Design Science Research + Case Study
Name of Venue	2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)
Tools in Study	Tiny Structure Editor (Research Project)
Sentiment	intro -- negative = 7, neutral = 30, positive = 2 discussion -- negative = 5, neutral = 11, positive = 1

Study ID	29
Title of Study	Towards ontology-based domain specific language for internet of things
Year of Publication	2020
Author(s) Names	E. Negm, S. Makady, and A. Salah
Source of Study	Google Scholar, ACM
Type of Study	Design Science Research
Name of Venue	Proceedings of the 2020 9th International Conference on Software and Information Engineering (ICSIE)
Tools in Study	Jetbrains MPS
Sentiment	intro -- negative = 8, neutral = 22, positive = 5 implementation -- neutral = 20, positive = 2 conclusion -- neutral = 7

Study ID	30
Title of Study	Type-directed program transformations for the working functional programmer
Year of Publication	2020
Author(s) Names	J. Lubin and R. Chugh
Source of Study	Google Scholar
Type of Study	Design Science Research
Name of Venue	10th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2019)
Tools in Study	DEUCE
Sentiment	intro -- negative = 7, neutral = 9, positive = 2 implementation -- negative = 13, neutral = 16, positive = 6 usabilityChallenges -- negative = 3, neutral = 5, positive = 2

Study ID	31
Title of Study	What do practitioners expect from the meta-modeling tools? a survey
Year of Publication	2021
Author(s) Names	M. Ozkaya and D. Akdur
Source of Study	Google Scholar, Microsoft Academic, SCOPUS, Semantic Scholar
Type of Study	Survey
Name of Venue	Journal of Computer Languages
Tools in Study	MPS, MetaEdit+, WebGME, GEMS, sirius,Xtext, MS DSL ToolsMelange, GME*
Sentiment	intro -- negative = 11, neutral = 62, positive = 8 editorservice -- negative = 8, neutral = 20, positive = 6 lessonslearned -- negative = 10, neutral = 24, positive = 8 challenges -- negative = 5 toolusage -- negative = 1, neutral = 14, positive = 2 conclusion -- negative = 10, neutral = 20, positive = 3

Figure D.6: Data extraction results 27 - 31

E

Drools Concept hierarchy

The concept hierarchy presented on the following pages was extracted and interpreted from Drools railroad diagrams.

The diagram in Figure **??** represents the file level and can be considered the root of concept hierarchy. This hierarchy represents the concepts that are available to the rule file. As the only concept that we will examine in depth is the rule, we show some shared concepts or children of, for example, function, query and type declaration.

In our final implementation, the only children of File we implemented were the Import, Global and Rule concepts.

The diagram in Figure E.1 shows the children of a rule. Each attribute has a different Behavior and Structure and are thus all represented separately.

In these diagrams, we do not show a concept diagram for the RHS. This choice was because it would be more or less the concept diagram for Java Statements, with the addition of Rule Variables and some special Drools functions. The concept diagram for a General Purpose Language would be orders of magnitude more extensive and more complex than we wish to show here. Luckily, as MPS allows for almost seamless extension and integration of different languages, we can import JetBrains implementation of Java for the RHS.

We show the hierarch for the LHS in the diagram in Figure E.2. Because of the number of concepts being represented, it may be a little hard to read.

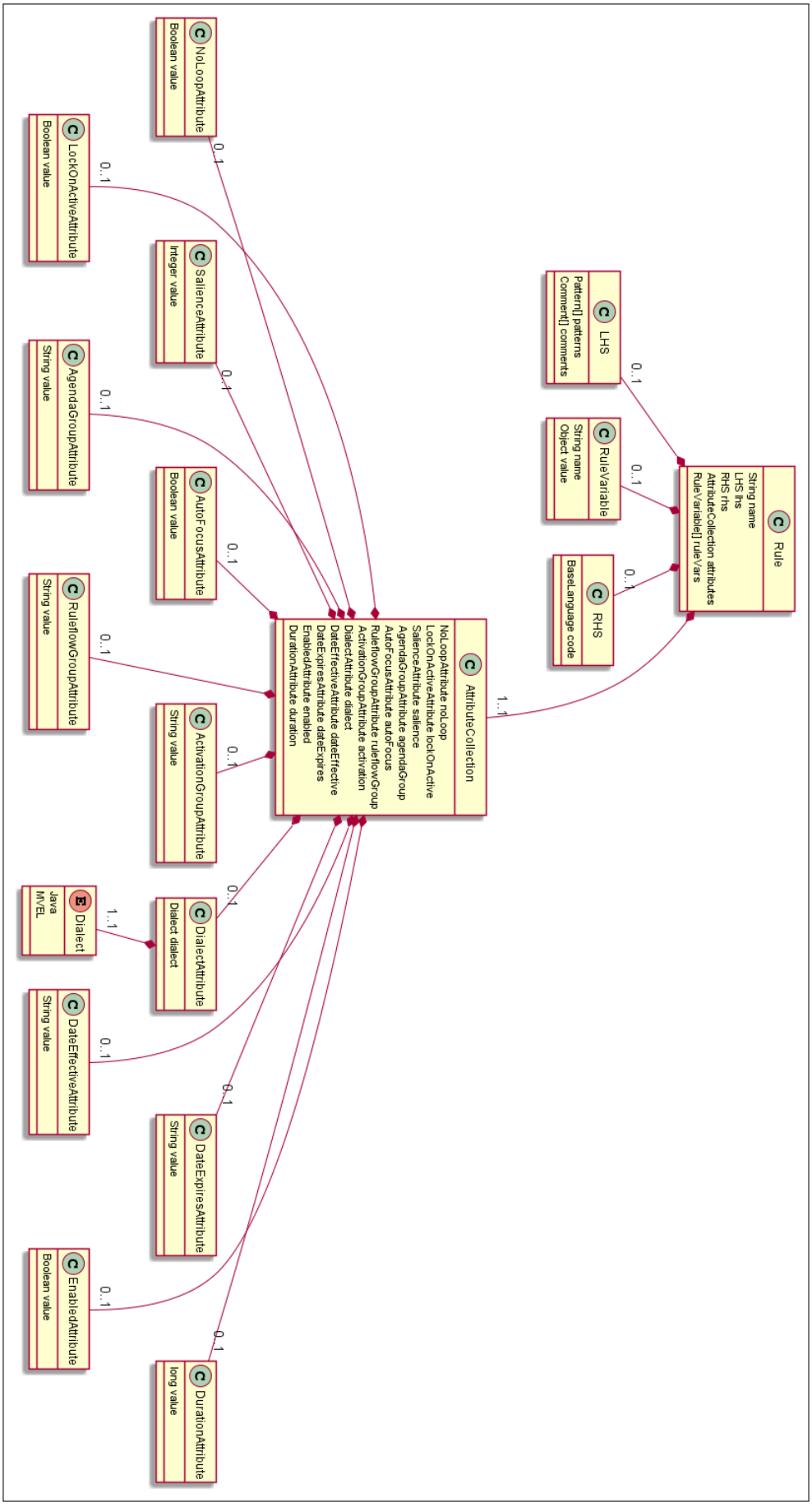


Figure E.1: Rules concept hierarchy diagram

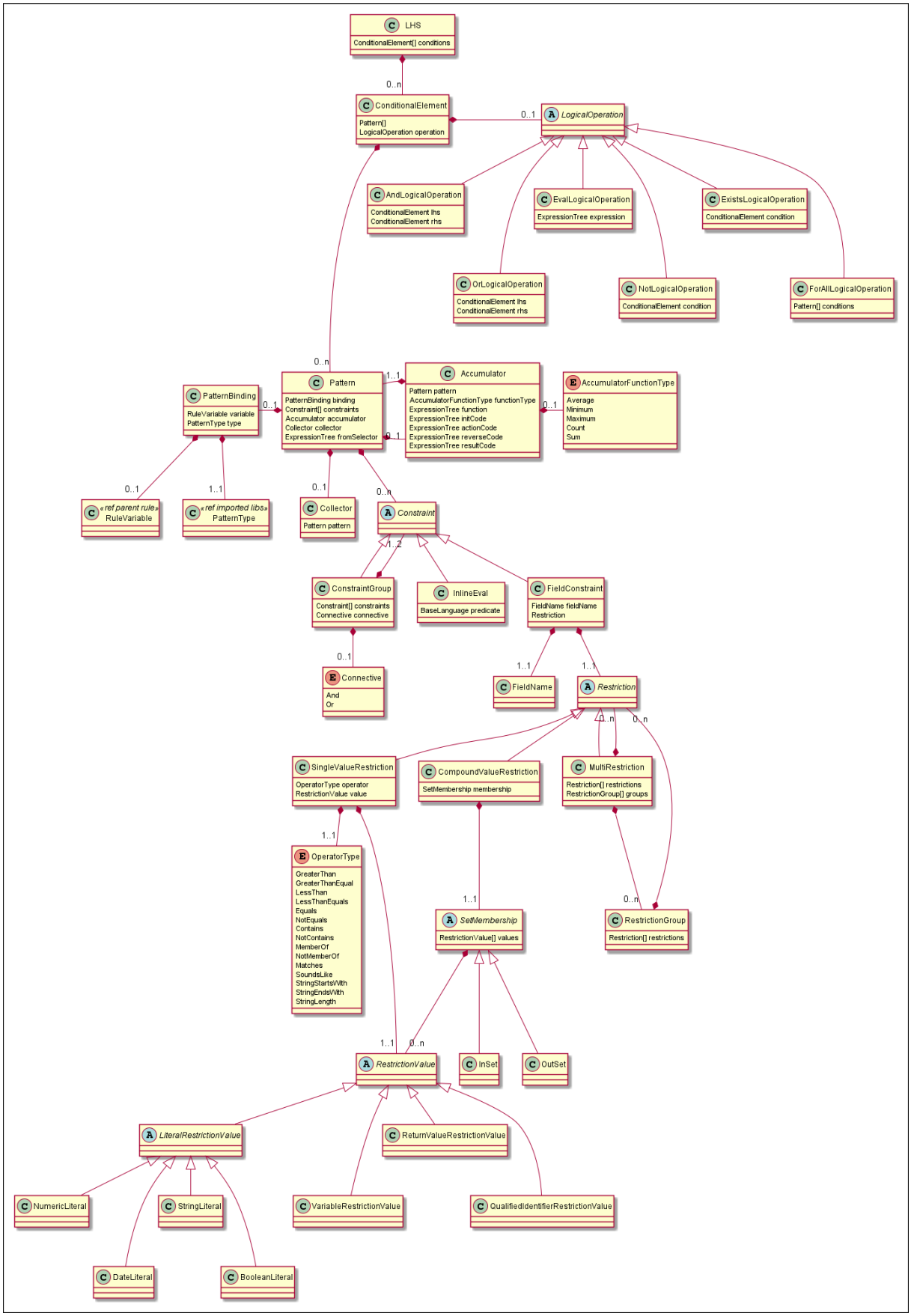


Figure E.2: Rule LHS concept hierarchy diagram

F

Questionnaire Text

Note: This questionnaire was presented on SurveyMonkey and thus the text here is a best approximation of their paging system.

Page 1 - Introduction

Thank you for taking part in this research.

According to SurveyMonkey, this survey should take 6 minutes to complete, when we tested it, the average was closer to 10 minutes.

This survey is for the validation section of a research master's 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.

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 to this survey by someone who previously answered this survey. (please feel free to forward this survey to anyone you know with Drools experience).

It is therefore assumed you are aware of what Drools is.

Projectional editing is a form of writing computer programs directly rather than writing text and having that parsed to create the program. This allows the developer multiple views and editors for the same code.

In this survey, we will present you with a few of these views.

On the following page, there is an animated GIF that will give a small demonstration of what this means.

Figure F.1 shows how this is presented to the subject.

Page 2 - Example of Projectional editing in Drools

Below is an animated GIF showing an example of a projectional implementation of Drools.

The top section is a tabular projection of the program.

The bottom part is a textual projection of the same program shown at the same time.

In this recording, we are editing in the tabular projection, which automatically updates the textual projection.

Here is placed an animated GIF of a demonstration of our prototype

Question: What is your first reaction to this mode of code editing?

Options: Very positive, Somewhat positive, Neutral, Somewhat negative, Very negative

the order of the options will be randomly presented as either "Very positive" to "Very negative" or "Very negative" to "Very posit

Figure F.2 shows how this is presented to the subject.

Page 3 - Positive about projectional editing

This page is only selected if the user chose very positive or somewhat positive

This question is optional.

you may use the Green "PREV" button to review the previous page.

Question: how would this coding style be useful to your interactions with Drools?
This is an open question with a text box.
Figure F.3 shows how this is presented to the subject.

Page 4 - negative about projections

This page is only selected if the user chose very positive or somewhat positive
This question is optional.
you may use the Green "PREV" button to review the previous page.
Question: What do you find negative with this style of coding
This is an open question with a text box.

Page 5 - Testing a projection

In questionnaire version A & D page 5 will be Testing a projection
In questionnaire version B & C page 5 will be Testing textual projection
On this page, we present you with an example projection of a collection of Drools rules, in this case, as a sort of decision table.
We will ask you to describe what you think it does, if you can't that is also good data for us.
A brief description of how this projection works follows:
for the decision table the following text:
1) each row is a rule
2) each column is a fact, or, when indented, a selection criteria of that fact
3) smiley faces indicate that a fact has been selected for a rule
4) if a fact has been selected and a variable is bound to it then the variable name appears instead of the smiley face.
5) the "Then" part of the rule appears in the "Actions" column
for the other table the following text:
1) each row is a rule
2) each column is for a variable or a property of a fact
3) if a property is selected then the selection criteria is in the appropriate cell
4) unselected cells are indicated by a grey/beige colour
5) the "Then" part of the rule appears in the "Actions" column
depending on the version of this questionnaire the respondent will see one of the following pictures
Version A - decision table showing rule set 1 (FNWI)
Version B - decision table showing rule set 2 (LAW)
Version C - new table showing rule set 1
Version D - new table showing rule set 2
Question: Please describe what you think this group of rules does
This is an open question with a text box.
Question: How easy or difficult was it to describe this rule set?
Options: Very easy, Somewhat easy, Neutral, Somewhat difficult, Very difficult
the order of the options will be randomly presented as either "Very easy" to "Very difficult" or "Very difficult" to "Very easy"
Figure F.4 shows how this is presented to the subject.

Page 6 - Testing textual projection

In questionnaire version A & D page 6 will be Testing textual projection
In questionnaire version B & C page 6 will be Testing a projection
Here we present you a textual projection of Drools rules.
[Note: These are not the same rules as on the previous page]
depending on the version of this questionnaire the respondent will see one of the following pictures
Version A & C - a text projection of rule set 2 (LAW)
Version B & D - a text projection of rule set 1 (FNWI)
Question: Please describe what you think this group of rules does
This is an open question with a text box.

Question: How easy or difficult was it to describe this rule set?
Options: Very easy, Somewhat easy, Neutral, Somewhat difficult, Very difficult
the order of the options will be randomly presented as either "Very easy" to "Very difficult" or "Very difficult" to "Very easy"
Figure F.5 shows how this is presented to the subject.

Page 7 - Comparing projections 1

In this question, we ask to compare a new projection to a previously shown projection, on the page named "Testing a projection".

If you wish to reacquaint yourself with the previous projection, you can use the Green "PREV" button at the bottom of this page.

A brief description of how this new projection works follows:
for the decision table the following text:
1) each row is a rule
2) each column is a fact, or, when indented, a selection criteria of that fact
3) smiley faces indicate that a fact has been selected for a rule
4) if a fact has been selected and a variable is bound to it then the variable name appears instead of the smiley face.
5) the "Then" part of the rule appears in the "Actions" column
for the other table the following text:
1) each row is a rule
2) each column is for a variable or a property of a fact
3) if a property is selected then the selection criteria is in the appropriate cell
4) unselected cells are indicated by a grey/beige colour
5) the "Then" part of the rule appears in the "Actions" column
depending on the version of this questionnaire the respondent will see one of the following pictures
Version A - new table showing rule set 1
Version B - new table showing rule set 2
Version C - decision table showing rule set 1
Version D - decision table showing rule set 2
Question: How does the above projection compare to the first projection you described?
Options: Much easier to understand, Somewhat easier to understand, Neutral, Somewhat harder to understand, Much harder to understand
the order of the options will be randomly presented as either "Much easier to understand" to "Much harder to understand" or "
Figure F.6 shows how this is presented to the subject.

Page 8 - Comparing projections 2

In this question, we again ask to compare the new projection, this time to the textual projection, on the page named "Testing textual projection".

If you wish to reacquaint yourself with the textual projection, you can, of course, use the Green "PREV" button at the bottom of this page again.

depending on the version of this questionnaire the respondent will see one of the following pictures
Version A - new table showing rule set 2
Version B - new table showing rule set 1
Version C - decision table showing rule set 2
Version D - decision table showing rule set 1
Question: How does the above projection compare to the text Drools rules you described?
Options: Much easier to understand, Somewhat easier to understand, Neutral, Somewhat harder to understand, Much harder to understand
the order of the options will be randomly presented as either "Much easier to understand" to "Much harder to understand" or "
Figure F.7 shows how this is presented to the subject.

Page 9 - Single rule helper 1 - Truth table

In questionnaire version A & D page 9 will be the Truth Table

In questionnaire version B & C page 9 will be the Circuit Diagram

Below we present another projection. This is a truth table projection. It highlights the conditions that have to be true for a rule to be selected.

The GIF shows the rule selected and the developer pressing the up and down arrow keys to step through the different true (highlighted in green) and false (highlighted in red) fact selections that result in a true outcome.

An animated GIF of the truth table example

Question: Would this help you with understanding your Drools rules?

Options: It would really help understanding, it would somewhat help understanding, Neutral, It would add a little confusion, It would add a lot of confusion

the order of the options will be randomly presented as either "It would really help understanding" to "It would add a lot of con

Figure F.8 shows how this is presented to the subject.

Page 10 - Single rule helper 2 - Circuit Diagram

In questionnaire version A & D page 10 will be the Circuit Diagram

In questionnaire version B & C page 10 will be the Truth Table

This is a circuit diagram of the selection conditions. choosing a different condition highlights how they are related to each other.

The GIF shows the rule selected and the developer pressing the up and down arrow keys to step through the different fact selections (highlighted in yellow) and shown in the circuit diagram, thus showing how the facts relate to each other.

An animated GIF of the Circuit Diagram example

Question: Would this help you with understanding your Drools rules?

Options: It would really help understanding, it would somewhat help understanding, Neutral, It would add a little confusion, It would add a lot of confusion

the order of the options will be randomly presented as either "It would really help understanding" to "It would add a lot of con

Figure F.9 shows how this is presented to the subject.

Page 11 - The Statistics page

Here we ask for data that we can use to slice and dice results.

Question: How long was/is your career as a developer?

Options: 0-1 year, 1-3 years, 3-10 years, greater than 10 years, none of the above

Question: When was the last time you had a coding interaction with Drools?

Options: during this week, some time after July 1st 2021, some time after Jan 1st 2021, some time after 2016, some time before 2016

Question: how long did you work with Drools?

Options: for years and intensely, for years but occasionally, not for long but intensely, I barely touched it

Question: Which tools have you used to edit Drools rules?

Checkboxes: Drools workbench, eclipse (with Drools plug-in), IntelliJ IDEA (with Drools plug-in), IDE or text editor without Drools assistance, other (please specify) *has textbox*, none of the above

Figure F.10 shows how this is presented to the subject.

Page 12 - So long, and thanks for all the fish

Thank you for your time. We leave you with a box where you can put in any thoughts about this if you feel like it.

Question: Do you have any thoughts or opinions you would like to share about what you have seen in this questionnaire?

This is an open question with a text box.

Figure F.11 shows how this is presented to the subject.

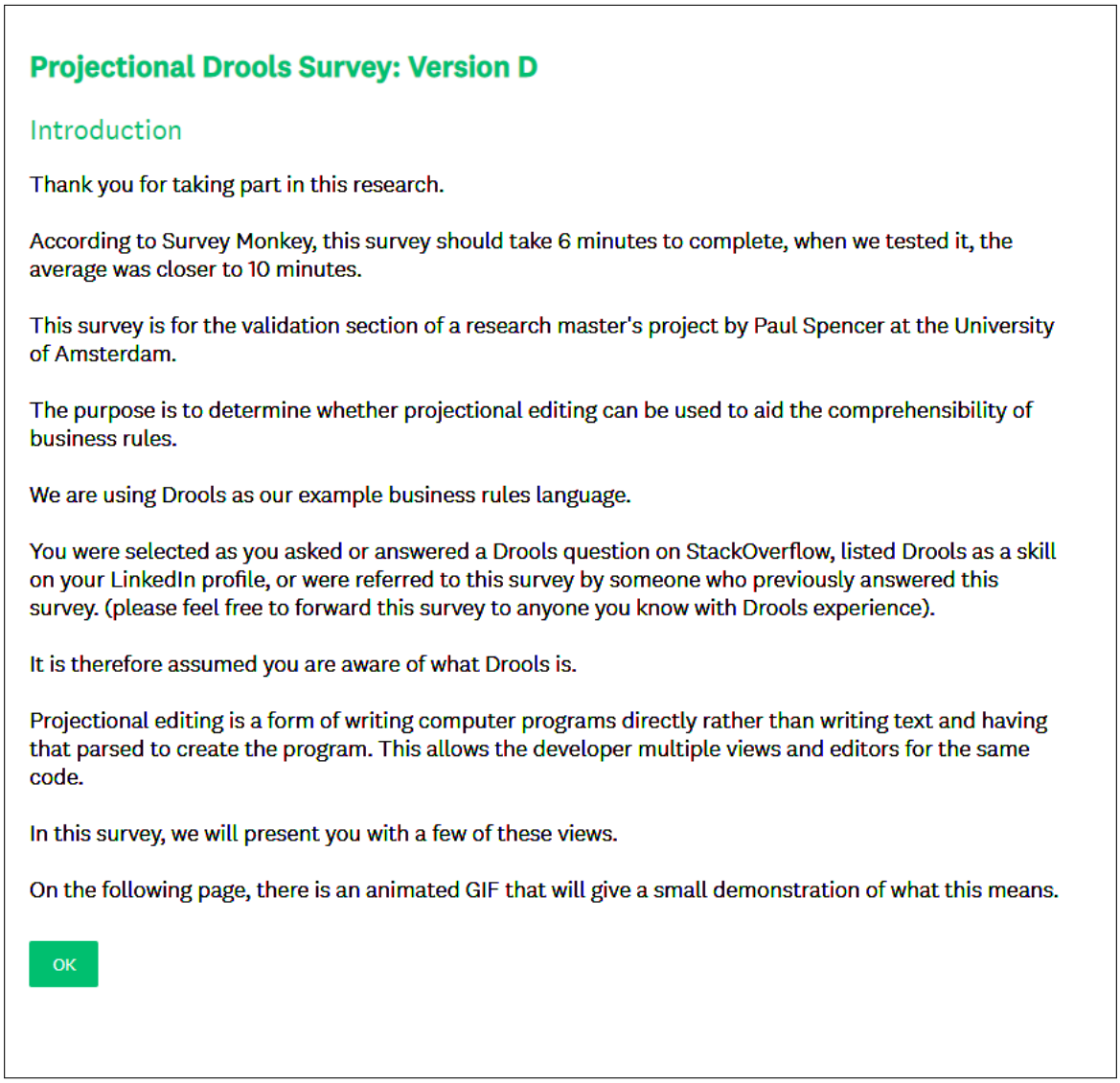


Figure F.1: Screen 1 - introduction text

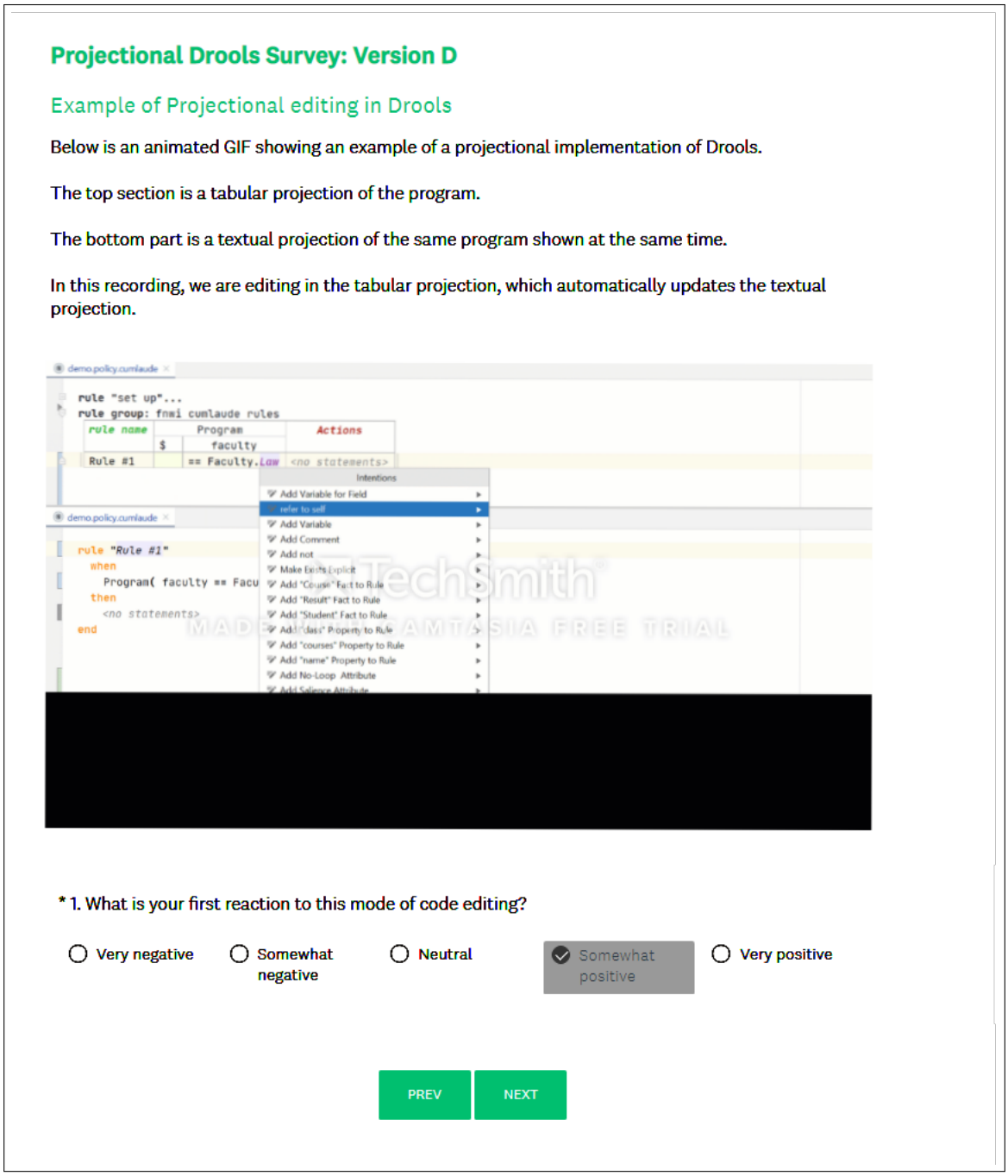


Figure F.2: Screen 2 - first impression

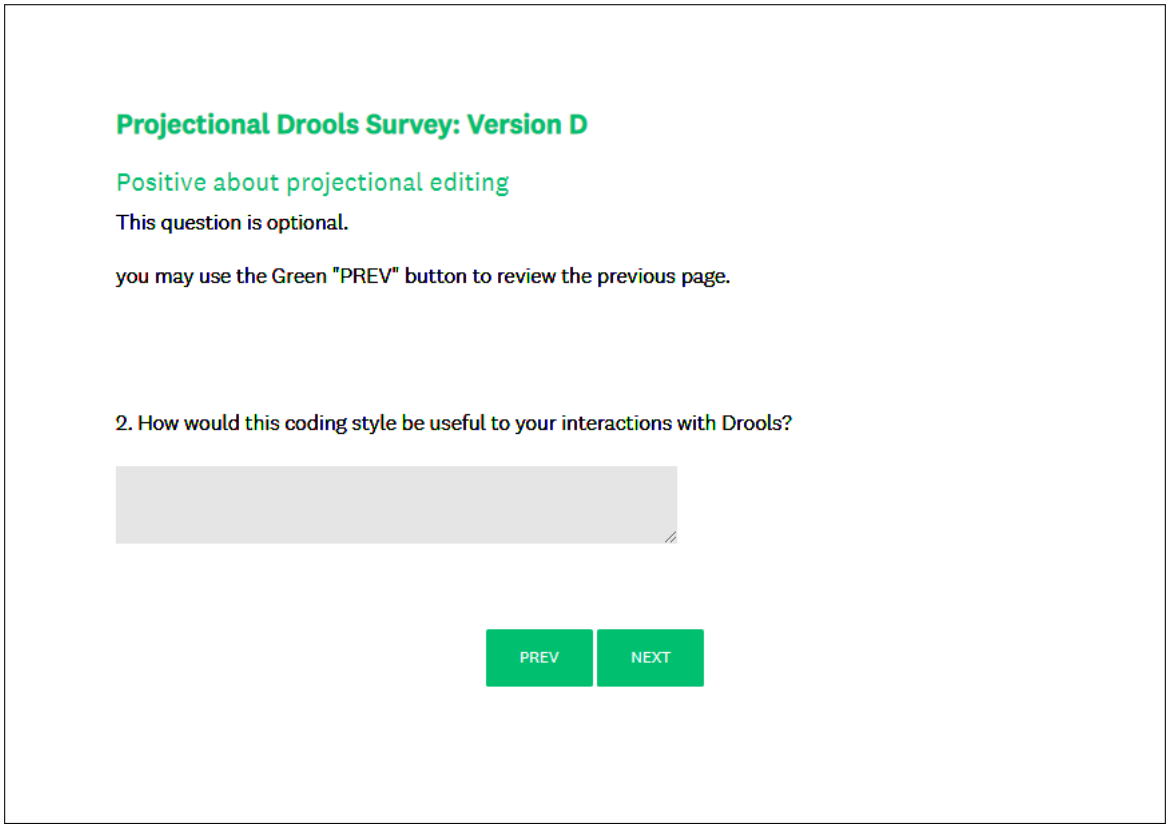


Figure F.3: Screen 3 - positive response

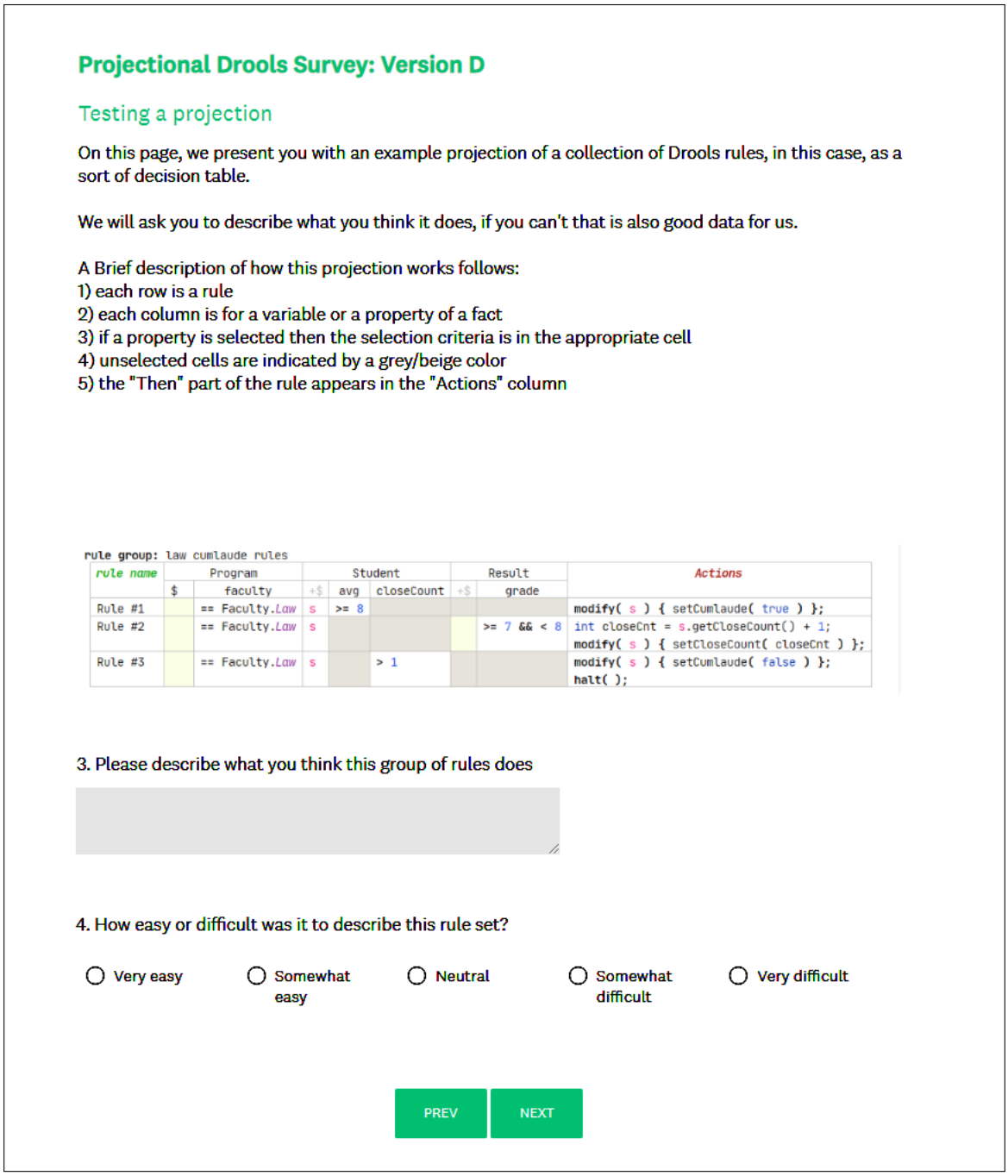


Figure F.4: Screen 4 - describe projection

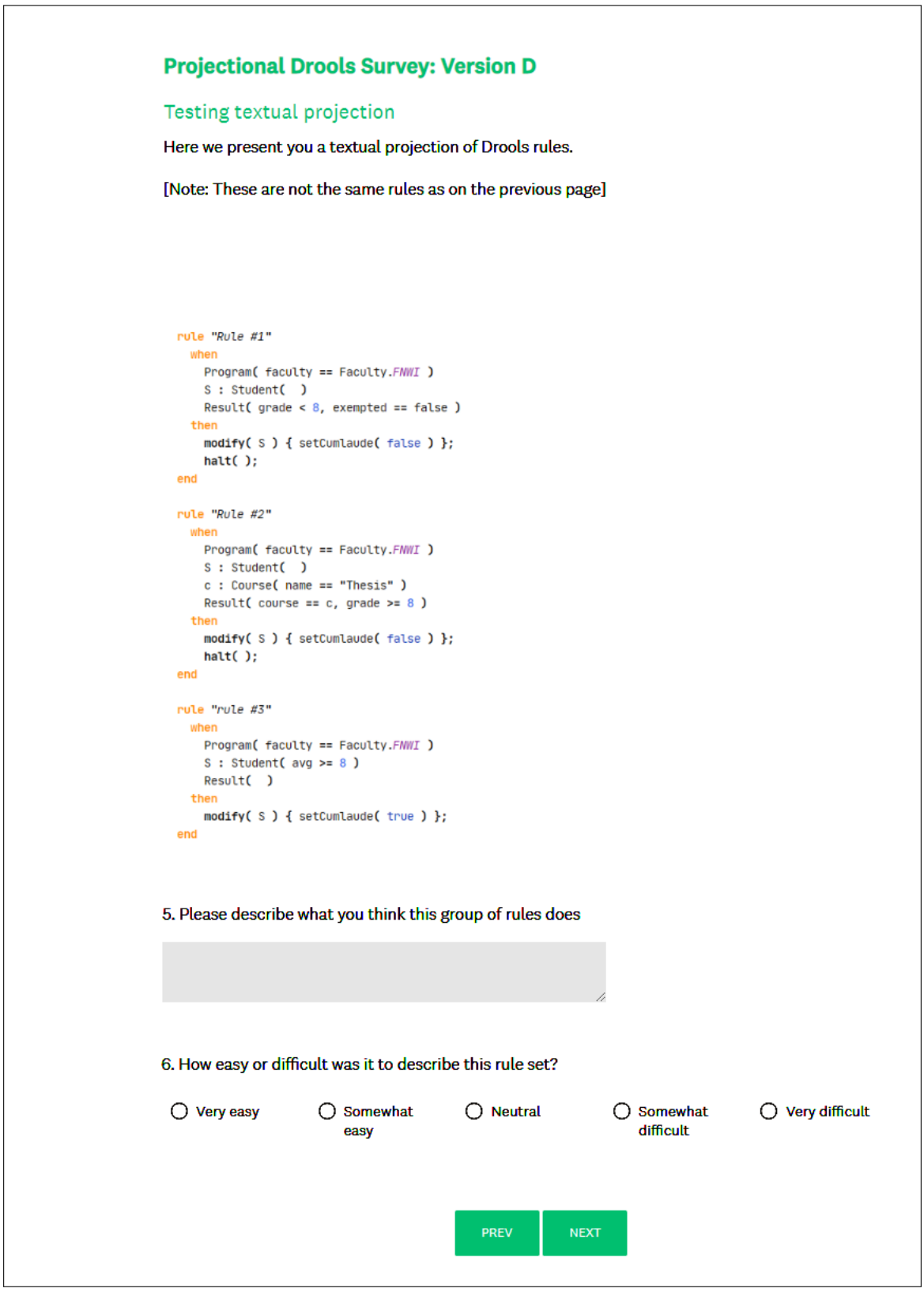
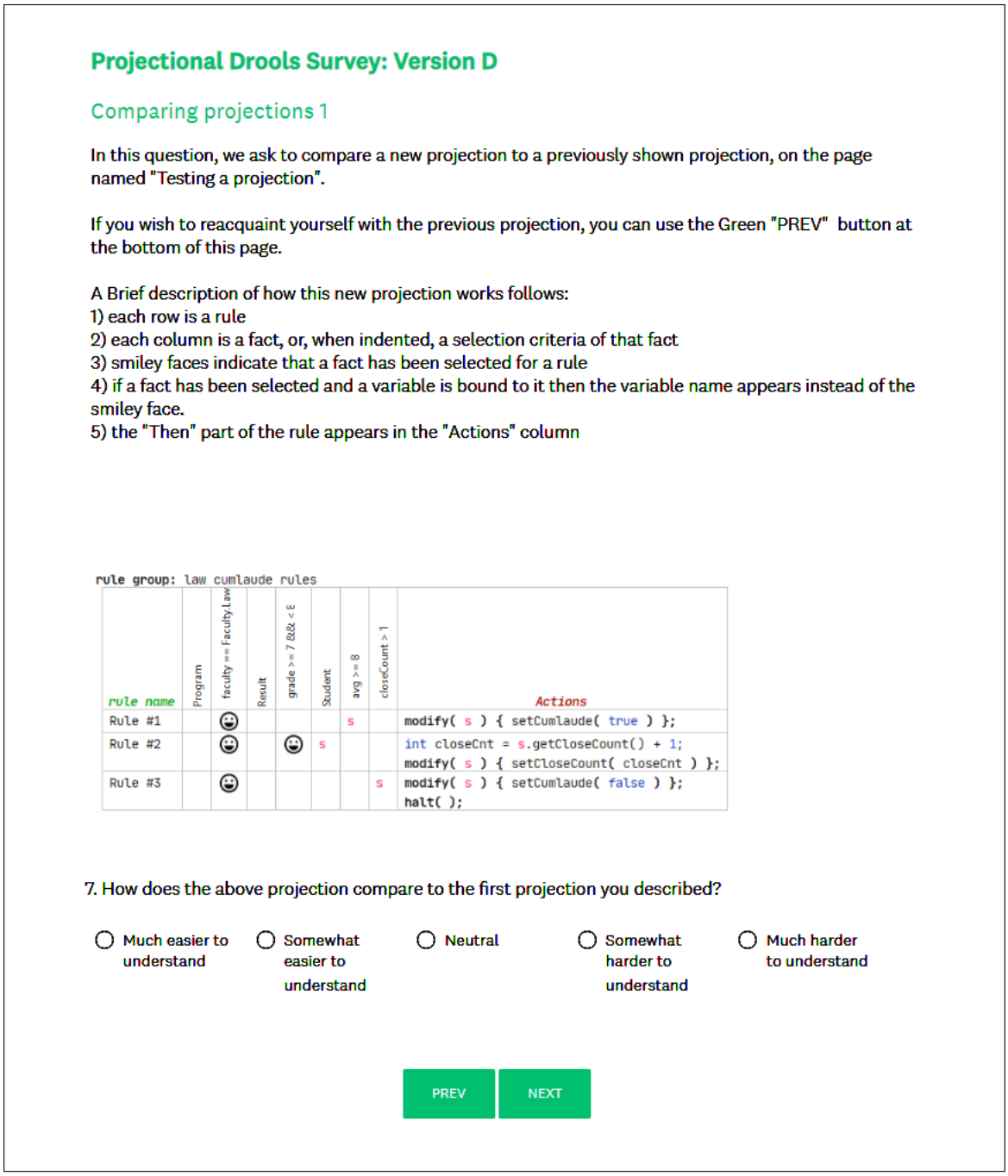


Figure F.5: Screen 5 - describe text



Projectional Drools Survey: Version D

Comparing projections 1

In this question, we ask to compare a new projection to a previously shown projection, on the page named "Testing a projection".

If you wish to reacquaint yourself with the previous projection, you can use the Green "PREV" button at the bottom of this page.

A Brief description of how this new projection works follows:
1) each row is a rule
2) each column is a fact, or, when indented, a selection criteria of that fact
3) smiley faces indicate that a fact has been selected for a rule
4) if a fact has been selected and a variable is bound to it then the variable name appears instead of the smiley face.
5) the "Then" part of the rule appears in the "Actions" column

rule group: law cumlaude rules

rule name	Program	faculty == Faculty.Law	Result	grade >= 7 && < 8	Student	avg >= 8	closeCount > 1	Actions
Rule #1		😀				s		modify(s) { setCumlaude(true) };
Rule #2		😀		😀	s			int closeCnt = s.getCloseCount() + 1; modify(s) { setCloseCount(closeCnt) };
Rule #3		😀					s	modify(s) { setCumlaude(false) }; halt( );

7. How does the above projection compare to the first projection you described?

○ Much easier to understand ○ Somewhat easier to understand ○ Neutral ○ Somewhat harder to understand ○ Much harder to understand

PREV NEXT

Figure F.6: Screen 6 - compare projections

Projectional Drools Survey: Version D

Comparing projections 2

In this question, we again ask to compare the new projection, this time to the textual projection, on the page named "Testing textual projection".

If you wish to reacquaint yourself with the textual projection, you can, of course, use the Green "PREV" button at the bottom of this page again.

rule group: fnwi cumlaude rules

rule name	Course	name == "Thesis"	Program	faculty == Faculty.FNWI	Result	course == [Course Variable]	exempted == false	grade < 8	grade >= 8	Student	avg >= 8	*Actions*
Rule #1				😀			😀	😀		S		`modify( S ) { setCumlaude( false ) }; halt( );`
Rule #2		c		😀		😀			😀	S		`modify( S ) { setCumlaude( false ) }; halt( );`
rule #3				😀	😀						S	`modify( S ) { setCumlaude( true ) };`

8. How does the above projection compare to the text Drools rules you described?

- ○ Much easier to understand
- ○ Somewhat easier to understand
- ○ Neutral
- ○ Somewhat harder to understand
- ○ Much harder to understand

PREV NEXT

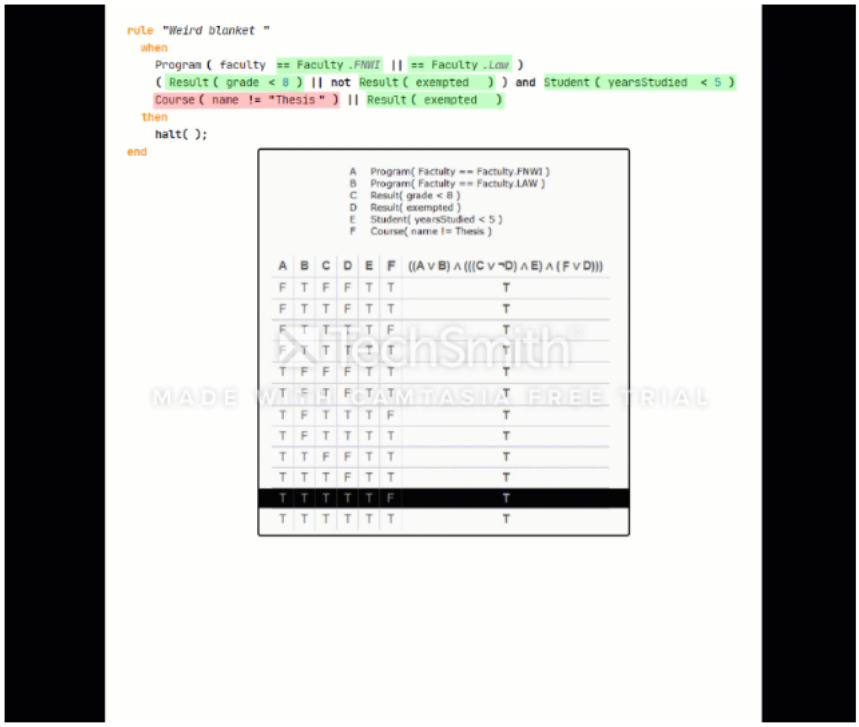
Figure F.7: Screen 7 - compare projection to text

Projectional Drools Survey: Version D

Single rule helper 1 - Truth table

Below we present another projection.
This is a truth table projection.
It highlights the conditions that have to be true for a rule to be selected.

The GIF shows the rule selected and the developer pressing the up and down arrow keys to step through the different true (highlighted in green) and false (highlighted in red) fact selections that result in a true outcome.



9. Would this help you with understanding your Drools rules?

- ◯ It would really help understanding
- ◯ it would somewhat help understanding
- ◯ Neutral
- ◯ It would add a little confusion
- ◯ It would add a lot of confusion

PREV NEXT

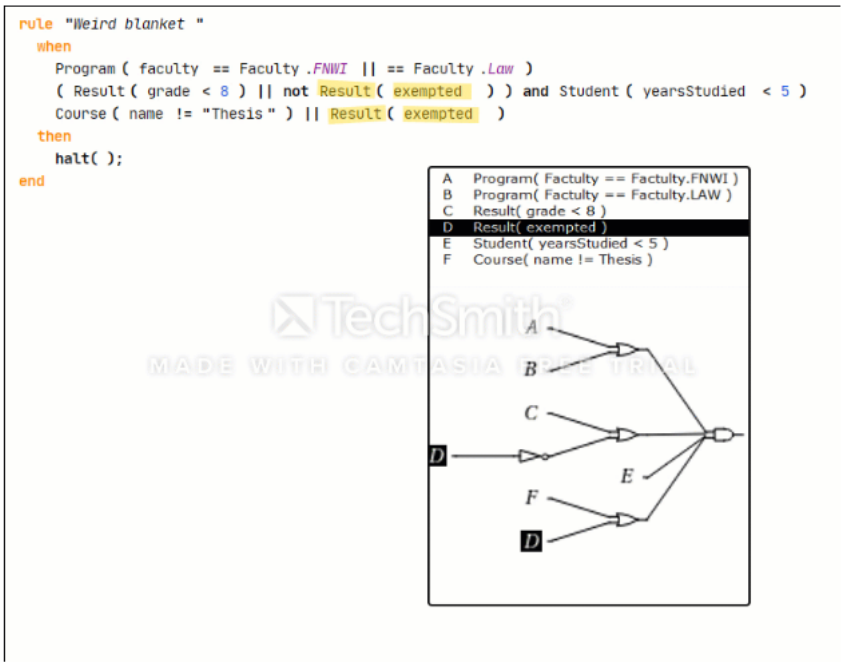
Figure F.8: Screen 8 - truth table

Projectional Drools Survey: Version D

Single rule helper 2 - Circuit Diagram

This is a circuit diagram of the selection conditions.
Choosing a different condition highlights how they are related to each other.

The GIF shows the rule selected and the developer pressing the up and down arrow keys to step through the different fact selections (highlighted in yellow) and shown in the circuit diagram, thus showing how the facts relate to each other.



10. Would this help you with understanding your Drools rules?

- ○ It would really help understaning
- ○ it would somewhat help understanding
- ○ Neutral
- ○ It would add a little confusion
- ○ It would add a lot of confusion

PREV NEXT

Figure F.9: Screen 9 - circuit diagram

Projectional Drools Survey: Version D

The Statistics page

Here we ask for data that we can use to slice and dice results.

11. How long was/is your career as a developer?

- ○ 0-1 year
- ○ 1-3 years
- ○ 3-10 years
- ○ greater than 10 years
- ○ None of the above

12. When was the last time you had a coding interaction with Drools?

- ○ during this week
- ○ some time after July 1st 2021
- ○ some time after Jan 1st 2021
- ○ some time after 2016
- ○ some time before 2016

13. how long did you work with Drools?

- ○ for years and intensely
- ○ for years, but occasionally
- ○ not for long, but intensely
- ○ I barely touched it

14. Which tools have you used to edit Drools rules?

- ☐ Drools workbench
- ☐ eclipse (with drools plugin)
- ☐ IntelliJ IDEA (with drools plugin)
- ☐ IDE or text editor without Drools assistance
- ☐ None of the above

Other (please specify)

PREV NEXT

Figure F.10: Screen 10 - personal details page

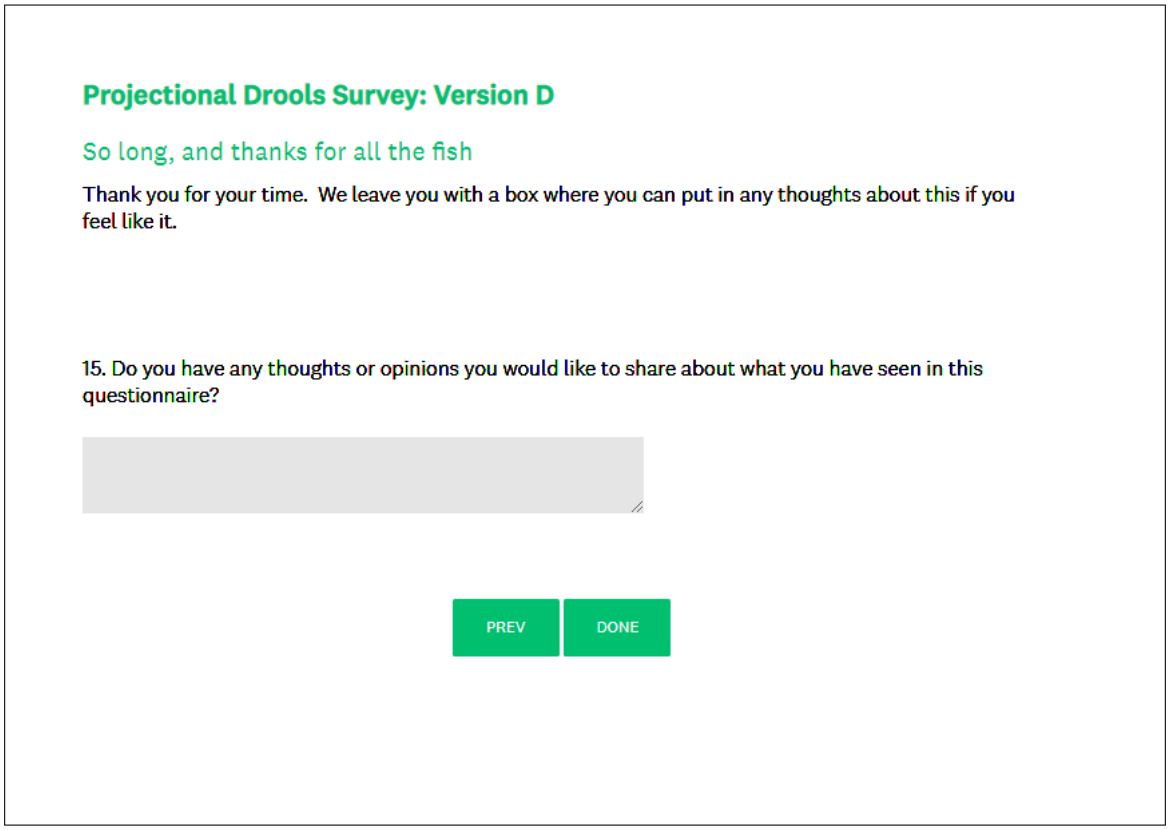


Figure F.11: Screen 11 - further comments