Master Project Proposal

Paul Spencer: 11721677
University of Amsterdam
paul.spencer@student.uva.nl

January 2021

1 Project summary

- **Project title:** Representation Mismatch Reduction for Development in Rules-Based Business Engines
- **Host organization:** Khonraad Software Engineering B.V.
- **Host supervisor:** Toine Khonraad, a.khonraad@khonraad.nl
- **Academic supervisor:** Clemens Grelck, c.grelck@uva.nl

Drools is a rules-based business engine, with rules represented in a DSL. When there are a large number of rules in a system, it becomes difficult to reason about them.

Projectional editors are editors where developers program directly into projections of the Abstract Syntax Tree (AST). Projectional editors can have multiple editable projections or representations of the same AST.

It is the intent of this project to investigate whether the use of various projections of a Drools AST can address the feedback requirements to allow developers to reason better about their business rules.

2 Problem analysis

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

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

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

This configuration is done in a business rule system. A business rules engine executes rules at runtime. "Rules specify conditions to be monitored and operations that should be executed when certain conditions are detected. Rather than continuously monitoring the simulation, experts can define and deploy appropriate rules that are automatically evaluated at runtime"[15].

Specifically, the rules engine used is JBoss Rules, more commonly known as Drools, from JBoss, a subsidiary of RedHat[2]. Drools is a framework for Rule-Based development. It is an open-source production rule system for complex event processing, using the ReteOO and Phreak implementations of the pattern matching Rete algorithm[6]. The rules are described in a Domain Specific Language (DSL). These are stored in Drools (.drl) files. An example of a DRL file can be seen in figure 1.

```
package org.drools.examples.honestpolitician

import org.drools.examples.honestpolitician.Politician;
import org.drools.examples.honestpolitician.Hope;

rule "We have an honest Politician"
    salience 10
    when
        exists( Politician( honest == true ) )
    then
        insertLogical( new Hope() );
end

rule "Hope Lives"
    salience 10
    when
        exists( Hope() )
    then
        System.out.println("Hurrah!!! Democracy Lives");
end

rule "Hope is Dead"
    when
        not( Hope() )
    then
        System.out.println( "We are all Doomed!!! Democracy is Dead" );
end

rule "Corrupt the Honest"
    when
        politician : Politician( honest == true )
        exists( Hope() )
    then
        System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );
        modify( politician ) {
            setHonest( false )
        }
end
```

Figure 1: Example drools file

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

The Drools language does not have tooling in standard IDEs to help developers to reason about the code. The problem of a lack of useful visualization for Drools has been known as far back as 2011, when Kaczor, et al[12] proposed a method of visualising Drools. There have also been a few commercial tools to help. However, these all suffer from the fact that they are not integrated and thus have parsing issues and a lack of immediate feedback.

We have observed the difficulty that developers have trying to reason about and edit collections of Drools files. We hypothesize that developers can be presented with different views on their code that will allow them to better understand the code. The problem we wish to solve - how to improve the ability to reason about large collections of Drools rules - we believe, lends itself to the technique of projectional editing.

Editing programs in a text editor means that you must match the syntax for the parsers to transform the text into an AST. Projectional editors are editors in which a user edits the abstract syntax tree directly without using a parser[27]. This potentially allows for almost unlimited language composition and flexible notations. Like MVC Pattern, changes in one projection of the AST will instantly be visible and editable in another projection[9].

By using projections to improve feedback whilst coding, we believe that this can reduce the representation impedance mismatch that hampers developer's reasoning. To build a projectional language in the time available for this project we would need a viable language workbench.

2.1 Research Questions

The research question we wish to answer is:

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

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

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

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

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

3 Research method

Figure 2 summarizes how the questions are related and the methods used to answer them.

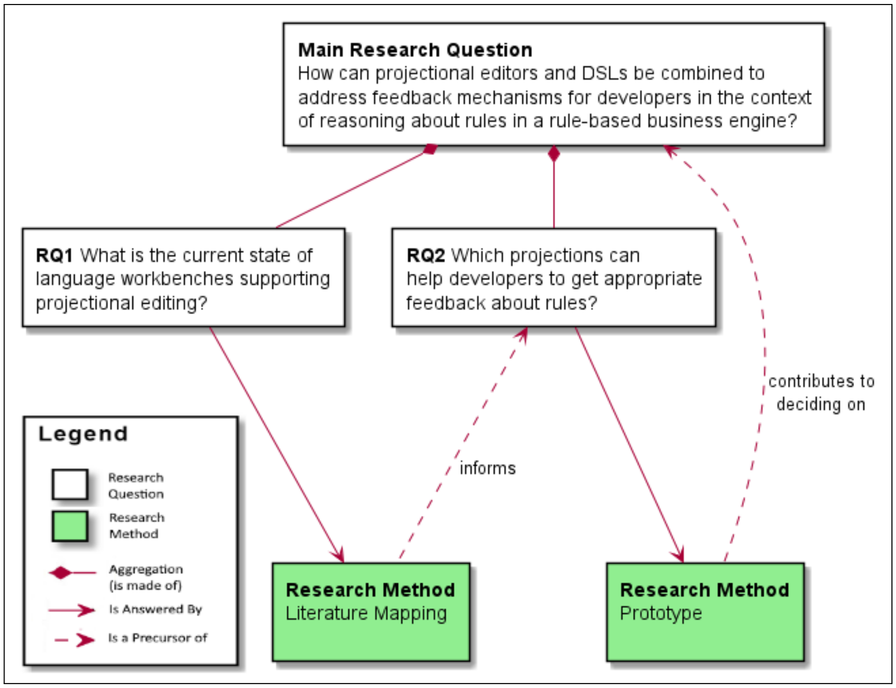

Figure 2: Research Questions and Methods

As seen in figure 2, question 1 - "What is the current state of language workbenches supporting projectional editing?", will be answered by the method of conducting a literature review of the field of language workbenches, specifically regarding those that support projectional editing. This research method will follow the recommendations of Kitchenham et al.[13]. To be clear on what we are investigating some terms should be defined first. A language workbench supports the efficient development of languages. The term caught on after a 2005 article by Martin Fowler[7]. Editing in language workbenches has two predominant editing forms - free-form text editing and projectional editing[5]. Projectional editing is a method of bypassing the need for a parser and programming directly into projections of the Abstract Syntax Tree.

Gregor[8], gives "A Taxonomy of Theory Types in Information Systems Research". For Question 3, "Which projections can help developers to get appropriate feedback about rules?", we will conduct what Gregor calls "Type V: Theory for Design and Action". The criteria for success of Type V research is that the prototype should "include utility to a community of users, the novelty of the artefact, and the persuasiveness of claims that it is effective".

For this project we will use the open-source language workbench Meta Programming System (MPS) from JetBrains[10]. MPS is built around the projectional editing paradigm. There is no existing implementation of the Drools language in MPS.

Drools is nearly 20 years old and, according to awesomeopensource[1], it is the most appreciated opensource rules engine. Despite this, it does not have strong IDE support. Mark Proctor said "for rule engines to have a future drl has to die"[20]. As the cofounder and platform architect of Drools his statement could be considered the conventional view of the state of the Drools ecosystem. We believe that this statement was motivated by the complexity of large rules files. In this project, we reject this hypothesis and instead we attempt to give the drl file a reprieve, through the use of projectional editing.

Thus, we will apply projectional editing techniques, through the MPS language workbench to the Drools language. The novelty of our approach will be to create new view types specific to the needs of a Drools programmer.

We will be relying on MPS as well as other open-source components. The reason we chose MPS is that it is the most developed of the free and open source projectional editing language workbenches, found in a study of the state of the art in Language workbenches[5].

An artefact of this master's project will be a prototype projectional editor, that will give much stronger editor support in JetBrain's IntelliJ, currently the most used Java IDE[11]. The prototype will consist of a of the Drools language, re-defined in the MPS language. The prototype will further consist of a set of projections of the DSL's AST. MPS uses the Java graphics framework Swing for the creation of graphical, as opposed to textual, projections. During the building of the prototype, we will decide upon which projections we will create. Some potential examples include:

- Visualization of order of rule execution;
- spreadsheet-like decision tables;
- or a "group-by" display on fact, query or function usage.

The major tasks in this prototype development will be:

- Modelling the Drools language;
- and developing the alternative projections.

The prototype itself will be validated by working. However, if time permits, the hypothesis of the usefulness of the projections will be further validated through developer use surveys.

4 Expected results

We expect the following from this project:

- We will be able to model Drools in MPS.
- We will create a suite of novel and useful projectional editors for the Drools language.
- We will reduce the thought to execution cycle for Drools Developers, resulting in a reduction of their "cognitive distance and representation impedance mismatch"[25].

A happy side effect of this project is that the following open-source products will become available to the public domain:

- An improved Drools editor plugin for the JetBrains IntelliJ community edition.
- An MPS implementation of the Drools DSL that can be used by other MPS language implementations for model to model generation.

5 Required expertise for this project.

Table 1 shows our expected and actual expertise levels in the technologies and practices required to complete this project.

Skill	Required	Acquired	Notes
MPS	★★★★★	★★⯨☆☆	Currently taking various courses.
Drools	★★★⯨☆	★★☆☆☆	The language is simple.
Java	★★★★☆	★★★☆☆	15 years of C#, these are similar.
Swing	★★★☆☆	☆☆☆☆☆	I have never played with this.
Language Design	★★★☆☆	★★☆☆☆	More for deconstructing Drools than creating a new language.
Rules Engines	★★★★☆	★⯨☆☆☆	This being central to the cause for the study.

Table 1: Expertise required.

6 Timeline

This prototype project consists of two main parts. First is modelling the Drools structure, behaviour, constraints, editors, and generators. The second will be creating non-standard projections of the structure.

The gathering of data to inform the design decisions for the projections will run in parallel to implementing the Drools DSL.

Time will be allocated as:

- 20 hours of my work time per week will be dedicated to design and development of the software.

- Currently estimating 4-8 hours at the weekends for research and project writing.
- There is an additional period of 4 weeks at the end allocated to the rewriting of the thesis.

This is shown in the Gantt Chart in figure 3.

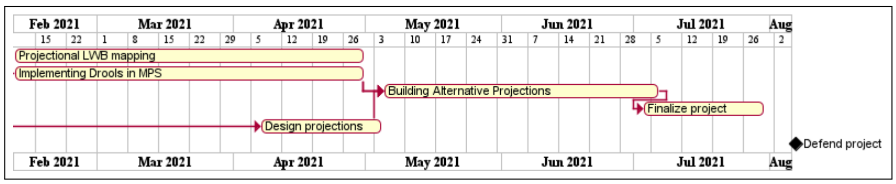

Figure 3: Predicted timeline.

7 Risks

There are various risks we will face, such as:

- Our designs of the projections, which will run in parallel to the Drools language modelling, will depend in part on the outcome of research carried out in the first period.
- Whether our design is appropriate with regards to performance and functionality.
- Whether we can achieve usefulness in our projections.

We hope to mitigate these risks through literature review and academic supervision.

What we consider our greatest risks are shown in Table 2.

Description	Risk Level	Contingency
Project goals are too ambitious	★★★☆☆	Reduce Drools implementation to a useful subset and reduce the number of projections.
MPS is not as flexible as needed	★★☆☆☆	Papers about mbeddr indicate low risk. We will limit our designs to MPS's capabilities.
MPS has too steep a learning curve	★★★★★	Currently taking training and reading lots of books. I have joined a user group and am implementing another language with a coding buddy.

Table 2: Project Risk.

8 Literature survey

To get an overview of the field we looked at MPS and Drools based papers. For MPS we started with an expert recommendation and did some forward and backward snowballing. A Google Scholar search produced one Drools papers with work on code visualization. The MPS papers and associated DSL papers covered some aspects of visual projectional editing, especially the papers relating to the product mbeddr.

Table 3 summarizes the papers and books investigated in preparation for this project.

During our research we will keep a document database, using the Zotero Personal research assistant software[23]. Also, we will create an annotated bibliography of the most relevant papers.

	Papers				
citations	Creating DSLs	How Drools works	Comparing Workbenches	How MPS works	Projectional Editing
[3]		⊕			
[4]			⊕	⊕	
[5]			⊕		
[9]					⊕
[12]		⊕			
[14]		⊕			
[16]			⊕	⊕	
[17]				⊕	
[18]				⊕	
[19]				⊕	
[21]				⊕	
[22]	⊕			⊕	
[24]		⊕			
[26]				⊕	
[27]	⊕		⊕	⊕	
[28]	⊕				
[29]				⊕	⊕
[30]				⊕	
[31]					⊕
[32]				⊕	⊕
[33]	⊕		⊕	⊕	⊕
[34]				⊕	
[35]	⊕			⊕	

Table 3: Papers about the Drools, MPS and Language workbenches

References

[1] *Awesome opensource ranking*. `https://awesomeopensource.com/projects/rule-engine`. Accessed: 2021-02-20.

[2] Paul Browne. *JBoss Drools business rules*. Packt Publishing Ltd, 2009.

[3] Erwin De Ley and Dirk Jacobs. “Rules-based analysis with JBoss Drools: adding intelligence to automation”. In: *Proceedings of ICALEPCS 2011* (2011), pp. 790–793.

[4] Sebastian Erdweg et al. "Evaluating and comparing language workbenches: Existing results and benchmarks for the future". In: *Computer Languages, Systems & Structures* 44 (2015), pp. 24–47.

[5] Sebastian Erdweg et al. "The state of the art in language workbenches". In: *International Conference on Software Language Engineering*. Springer. 2013, pp. 197–217.

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

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

[8] Shirley Gregor. "The nature of theory in information systems". In: *MIS quarterly* (2006), pp. 611–642.

[9] Stian M Guttormsen, Andreas Prinz, and Terje Gjøsæter. "Consistent Projectional Text Editors." In: *MODELSWARD*. 2017, pp. 515–522.

[10] *Jetbrains MPS Product Page*. `https://www.jetbrains.com/mps/`. Accessed: 2021-01-19.

[11] *JRebel Java Technology Report*. `https://www.jrebel.com/blog/2020-java-technology-report`. Accessed: 2021-01-27.

[12] Krzysztof Kaczor et al. "Visual design of Drools rule bases using the XTT2 method". In: *Semantic methods for knowledge management and communication*. Springer, 2011, pp. 57–66.

[13] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press, 2015.

[14] Narendra Kumar, Dipti D Patil, and Vijay M Wadhai. "Rule based programming with Drools". In: *International Journal of Computer Science and Information Technologies* 2.3 (2011), pp. 1121–1126.

[15] Hua Liu and Manish Parashar. "Dios++: A framework for rule-based autonomic management of distributed scientific applications". In: *European Conference on Parallel Processing*. Springer. 2003, pp. 66–73.

[16] Sofia Meacham, Vaclav Pech, and Detlef Nauck. "Classification Algorithms Framework (CAF) to Enable Intelligent Systems Using JetBrains MPS Domain-Specific Languages Environment". In: *IEEE Access* 8 (2020), pp. 14832–14840.

[17] Domenik Pavletic et al. "Extensible debuggers for extensible languages". In: *GI/ACM WS on Software Reengineering* (2013).

[18] Vaclav Pech, Alex Shatalin, and Markus Voelter. "JetBrains MPS as a tool for extending Java". In: *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*. 2013, pp. 165–168.

[19] Andreas Prinz. “Multi-level Language Descriptions.” In: *MULTI MoDELS*. 2016, pp. 56–65.

[20] Mark Proctor. *Drools 7.X Happenings and roadmap*. `https://www.youtube.com/watch?v=Kz7a3COtj1A`. Accessed: 2021-02-15. 2016.

[21] Daniel Ratiu, Vaclav Pech, and Kolja Dummann. “Experiences with teaching MPS in industry: towards bringing domain specific languages closer to practitioners”. In: *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2017, pp. 83–92.

[22] Daniel Ratiu, Markus Voelter, and Domenik Pavletic. “Automated testing of DSL implementations—experiences from building mbeddr”. In: *Software Quality Journal* 26.4 (2018), pp. 1483–1518.

[23] *Roy Rosenzweig Center For History And New Media. Zotero product page.* `https://www.zotero.org/`. Accessed: 2021-01-24.

[24] Kay-Uwe Schmidt, Roland Stühmer, and Ljiljana Stojanovic. “Blending complex event processing with the rete algorithm”. In: *Proceedings of iCEP2008: 1st International Workshop on Complex Event Processing for the Future Internet*. Vol. 412. Citeseer. 2008.

[25] Tijs van der Storm and Felienne Hermans. “Live literals”. In: *Workshop on Live Programming (LIVE)*. Vol. 2016. 2016.

[26] Markus Voelter. “Fusing modeling and programming into language-oriented programming”. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2018, pp. 309–339.

[27] Markus Voelter. *Generic tools, specific languages*. Citeseer, 2014.

[28] Markus Voelter et al. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.

[29] Markus Voelter et al. “Lessons learned from developing mbeddr: a case study in language engineering with MPS”. In: *Software & Systems Modeling* 18.1 (2019), pp. 585–630.

[30] Markus Voelter et al. “Shadow models: incremental transformations for MPS”. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*. 2019, pp. 61–65.

[31] Markus Voelter et al. “Towards user-friendly projectional editors”. In: *International Conference on Software Language Engineering*. Springer. 2014, pp. 41–61.

[32] Markus Voelter et al. “Using C language extensions for developing embedded software: a case study”. In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 2015, pp. 655–674.

[33] Markus Voelter et al. “Using language workbenches and domain-specific languages for safety-critical software development”. In: *Software & Systems Modeling* 18.4 (2019), pp. 2507–2530.

[34] Premysl Vysoky. "Grammar to JetBrains MPS Convertor". In: (2016).

[35] Andreas Wortmann and Martin Beet. "Domain specific languages for efficient satellite control software development". In: *ESASP* 736 (2016), p. 2.