Université de Montréal

Gentleman: A Lightweight Web-based Projectional Editor

par

Louis-Edouard Lafontant

Département d'informatique et recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

November 26, 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Gentleman: A Lightweight Web-based Projectional Editor

présenté par

Louis-Edouard Lafontant

a été évalué par un jury composé des personnes suivantes :

Houari Sahraoui

(président-rapporteur)

Eugene Syriani

(directeur de recherche)

Marc Feeley

(membre du jury)

Résumé

Lors de la conception et la manipulation de logiciel par modélisation, il est avantageux de bénéficier d'un grand degré de liberté au niveau de la présentation afin de comprendre l'information et prendre une action en exerçant peu d'effort cognitif et physique. Cette caractéristique doit aussi s'étendre aux outils que nous employons afin que ceux-ci augmentent nos capacités, plutôt que les restreindre. En génie logiciel, nous travaillons présentement à rehausser encore le niveau d'abstraction afin de réduire le rôle central du code décrit avec un langage de programmation à usage général. Ceci permettrait d'inclure les experts non techniques dans les activités de développement de logiciel. Cette approche, centralisée sur le domaine et l'expert, s'inscrit dans l'ingénierie dirigée par les modèles (IDM), où un modèle est produit et manipulé par divers experts et utilisateurs. Le modèle est alors décrit avec un langage créé spécifiquement pour un domaine d'application ou une tache, appelé langage dédié (DSL). Une technique utilisée pour créer ces modèles et leurs DSL est le *projectional editing*, qui permet d'utiliser des notations diverses interchangeables et d'étendre et composer facilement un langage. Toutefois, les solutions actuelles sont lourdes, spécifiques à une plateforme, et manquent considérablement d'utilisabilité, limitant ainsi l'usage et l'exploitation de cette approche. Pour mieux refléter les avantages du paradigme IDM avec le style projectionnel, nous introduisons dans cette thèse Gentleman, un éditeur projectionnel léger sur le web. Avec Gentleman, le développeur crée un modèle en combinant des concepts utilisés pour définir la structure du modèle et des projections pour les manipuler dans l'éditeur. Nous avons évalué Gentleman à travers une étude basée sur un groupe d'utilisateur. L'étude a confirmé sa capacité à créer et manipuler des modèles efficacement. Les participants ont noté qu'il est facile de prendre en main Gentleman et que l'interface est très intuitive comparativement aux éditeurs existants. Nous avons aussi intégré Gentleman avec succès à une plateforme web, démontrant ainsi ses capacités d'interopérabilité et l'avantage d'une solution web.

Keyword: Ingénierie dirigée par les modèles, édition projectionnelle, application de langage, langage dédié

Abstract

In software activities and, more specifically, when modeling, the modeler should benefit from as much freedom as possible to understand the presented information and take action with minimal cognitive and mechanical effort. This characteristic should also apply to the tools used in the process so that they extend our capabilities rather than limit them. In the field of software engineering, current work aims to push the level of abstraction past general-purpose programming language into domain-specific modeling. This enables domain experts with various backgrounds to participate in software development activities. This vision is central to model-driven engineering (MDE) where, instead of code, various experts and users produce and manipulate domain-specific language (DSL). In recent years, projectional editing has proven to be a valid approach to creating and manipulating DSLs, as it supports various easily interchangeable notations and enables language extension and composition. However, current solutions are heavyweight, platform-specific, and suffer from poor usability. To better support this paradigm and minimize the risk of accidental complexity in terms of expressiveness, in this thesis, we introduce Gentleman, a lightweight web-based projectional editor. With Gentleman, a developer creates a model by combining concepts used to define its structure and projections to interact and manipulate them in the editor. We have evaluated Gentleman through a user study. The evaluation confirmed its capacity to create and manipulate models effectively. Most participants noted that the editor is very user-friendly and intuitive compared to existing editors. We have also successfully integrated Gentleman into a web application, demonstrating its interoperability and the benefit of a web solution.

Keyword: Model-driven enginneering, projection editing, language workbench, domain-specific language

Contents

Résumé 5

Abstract 7

List of tables 15

List of figures 17

List of abbreviations and acronyms 19

Aknowledgements 21

Chapter 1. Introduction 23

1.1. Context 23

1.2. Problematic and thesis proposition 24

1.3. Contribution 26

1.4. Outline 26

Chapter 2. Background and State of the art 29

2.1. Model-Driven Enginneering 29
2.1.1. Modeling 29
2.1.2. Domain-Specific Language 29
2.1.3. Metamodeling 31
2.1.4. Concrete syntax definition 32
2.1.5. Editor generation 35

2.2. Language workbench 35
2.2.1. Free-form editors 36
2.2.2. Syntax-directed editors 36
2.2.3. Projectional editors 37

2.3. Projectional editing 38

2.3.1. MPS 38
2.3.1.1. Structure 38
2.3.1.2. Editor 39
2.3.1.3. Type System 40
2.3.1.4. Code generation 40
2.3.2. The Whole Platform 40
2.3.3. Scratch 41
2.3.4. Literature review 41
2.3.5. Motivation for Gentleman 44

Chapter 3. Gentleman 45

3.1. Rationale 45

3.2. Running example 46

3.3. Architecture 47
3.3.1. Editor module 48
3.3.2. Concept module 51
3.3.3. Projection module 52

3.4. Implementation 52
3.4.1. Integration 53
3.4.2. Configuration 54

3.5. Editor services 55
3.5.1. Instantiation 55
3.5.2. Navigation 55
3.5.3. Reusing values 55
3.5.4. Undo/Redo 56
3.5.5. Code assistance 56
3.5.6. Search and filter 56
3.5.7. State 56
3.5.8. Import/Export 57
3.5.9. Bootstrapping 57

Chapter 4. Concept 59

4.1. Structure 60
4.1.1. Attribute 60

4.1.2. Property 61

4.2. Primitive 61

4.2.1. String 61

4.2.2. Number 63

4.2.3. Boolean 63

4.2.4. Set 64

4.2.5. Reference 64

4.3. Complex 65

4.3.1. Concrete 65

4.3.2. Prototype 65

4.3.3. Derivative 66

4.4. Comparison with OO 66

4.4.1. Modelling in EMF 66

4.4.2. Mapping 67

4.4.3. Transformation 67

Chapter 5. Projection 69

5.1. Structure 69

5.2. Layout 70

5.2.1. Flex Layout 71

5.2.2. Table Layout 71

5.3. Interaction points 71

5.3.1. TextField 72

5.3.2. BinaryField 72

5.3.3. ChoiceField 72

5.3.4. ListField 73

5.4. Static element 73

5.4.1. Text 73

5.4.2. Image 73

5.4.3. Link 74

5.4.4. ViewSwitch 74

5.4.5. Button 74

5.5. Relation function 75

5.6. Template 75

5.7. Styling 75

Chapter 6. Evaluation 77

6.1. User study 77

6.2. Objectives 77

6.3. Study design 78

6.3.1. Setup 78

6.3.2. Participant 78

6.3.3. Experiment process 79

6.3.4. Feedback survey 83

6.4. Metrics 83

6.4.1. Independant variables 83

6.4.2. Dependant variables 84

6.4.3. Survey variables 85

6.4.4. Null hypothesis formulation 85

6.4.5. Data collection 85

6.5. Results 86

6.5.1. Descriptive statistics 86

6.5.2. Statistical signifiance 87

6.5.3. Hypothesis validation 90

6.6. Discussion 92

6.7. Threats to validity 93

6.8. Integration with ReLiS 94

Chapter 7. Conclusion 97

7.1. Summary 97

7.2. Outlook 98

References 101

Appendix A. Ecore to Gentleman transformation 109

A.1. Ecore models 109

A.2. ATL transformation 109

A.3. EGL transformation 113

Appendix B. Gentleman TodoList artefacts 115

B.1. Concept schema 115

B.2. Projection schema 120

Appendix C. User study data 123

C.1. Participant profile 123

C.2. Survey 129

C.3. Results 139

C.3.1. Task 1 results 139

C.3.2. Task 2 results 145

C.3.3. Task 3 results 151

C.3.4. Survey results 157

List of tables

4.1 Mapping between Ecore and Gentleman concepts ... 67

6.1 Requirements for the TL model of task 1 ... 80

6.2 Requirements for the TL model of task 2 ... 82

6.3 Requirements for the TL model of task 3 ... 83

6.4 Task 1 ANOVA results ... 88

6.5 Task 2 ANOVA results ... 89

6.6 Survey ANOVA results ... 90

List of figures

2.1 Language Features (from [**57**]) 30
2.2 Four-layered architecture 32
2.3 Mind map metamodel using UML class diagram 33
2.4 Editor services for a web language (from [**46**]) 36
2.5 AToMPM user interface showing a domain-specific environment for modeling Mind Maps (from [**28**]) 37
2.6 MPS environment overview 39
2.7 WholePlatform environment overview 41
2.8 Scratch environment overview 42

3.1 TodoList metamodel 47
3.2 TodoList model in Gentleman 48
3.3 Gentleman architecture showing some key interactions 49
3.4 Gentleman editor menu giving access to the loaded concepts and projections 58

4.1 Gentleman concept metamodel 59
4.2 TodoList concept in Gentleman 62

5.1 Gentleman projection metamodel 69
5.2 TodoList projection in Gentleman 70

6.1 Traffic Light metamodel 80
6.2 Gentleman Traffic Light editor 81
6.3 MPS Traffic Light editor 81
6.4 Traffic Light initial metamodel: task 1 82
6.5 ReLiS DSL Forge editor 94
6.6 ReLiS editor: initialization 95

6.7 ReLiS editor: Screening . . . 95

A.1 Ecore metamodel . . . 110

A.2 Gentleman Ecore model . . . 111

List of abbreviations and acronyms

AST	Abstract syntax tree
DSL	Domain-specific language
EMF	Eclipse Modeling Framework
GUI	Graphical user interface
IDE	Integrated developement environment
JS	Javascript
LW	Language workbench
MDE	Model-driven engineering
OCL	Object Constraint Language
OO	Object-oriented

UML	Unified Modeling Language
UI	User Interface
UX	User Experience

Aknowledgements

I want to express my deepest gratitude to my supervisor and mentor Prof Eugene SYRIANI. Since our first meeting, I have discovered and learned so much with him and from him. In a way, he opened the path I am walking today. I would also like to take this opportunity to thank my colleagues and friends at the GEODES lab for their great support, shared insights, and the energy that is nurtured in the lab, which makes our activities more pleasant. I dedicate this thesis to my family, who gave me unyielding support and love throughout my life. The fire is burning strong today thanks to the warmth which you have embraced me with, "*un grand Merci à chacun d'entre vous* ". Lastly but surely to all my friends I have met traveling this life, I am grateful to have crossed your path. You made this adventure much richer and interesting.

Chapter 1

Introduction

This chapter introduces the context of the work presented in this thesis and the tackled problems. We also outline our contribution and the structure of the thesis.

1.1. Context

We create and use programs to interact with computers as a software layer of instructions compiled and executed by the computer. To create those programs, technical experts, such as software engineers and developers, have developed tools to organize their work and assist them in their tasks. As most programs are expressed using a text-based language, the essential tool used by developers is the editor to read and write the source code in a programming language. Most editors, such as Vim [**88**] and Visual Studio Code [**18**] manipulate general-purpose programming languages (GPL) such as C++, Python, Java, and Javascript. These languages offer a textual syntax with keywords and constructs that make programming more approachable to software developers. Most editors offer features such as syntax highlighting, indentation, code assistance, and the ability to select the characters to copy, cut, and paste. However, when creating a program, additional tools are needed to compile, execute and debug the program, ensure configuration management project, and manage all the files involved in the program. Therefore, most developers rely on an integrated development environment (IDE), such as Eclipse [**70**] and Visual Studio [**42**], that groups all these tools in a uniform and standard user interface, integrating them in a centralized environment.

Raising the level of abstraction to GPLs has been highly beneficial to software engineers and developers, as it reduced the cognitive effort needed to describe the solution. However, GPLs still impose a language gap between the problem domain (known by domain experts) and the solution domain (known by technical experts). This gap has motivated the creation of domain-specific languages (DSL) to capture precisely the semantics of an application domain [**37**]. However, those tools built with a generic approach for software engineers to exploit GPLs, are inadequate for DSLs used by domain experts of diverse backgrounds, such as banking, insurance, or healthcare. The domain may be restricted to a device (embedded system), a specific project such as a “meeting scheduler application” or it may be common to a larger group of experts and practitioners, such as the hardware description language VHDL [**5**] or the web languages CSS and HTML [**20**]. DSLs are central to model-driven engineering (MDE) [**48**], a software engineering methodology that relies heavily on the use of models, a high-level abstraction, designed using a DSL as the building artifact. This modeling practice characterized as domain-specific modeling (DSM), allows the solution to be specified directly using problem domain concepts, with the final products generated from these high-level specifications [**47**]. This approach makes it possible to leverage the expertise and knowledge of domain experts of various backgrounds instead of relying solely on technical experts, thus making software development more accessible and inclusive.

Language workbenches were created to support these modeling and language engineering activities. They are tools that support the efficient definition, reuse, and composition of languages and their IDEs [**21**]. Language workbenches offer a modeling language used by language engineers to create DSLs. In this process, the engineer works closely with the domain experts, as the language is an abstraction of their own domain. This promotes agility where the domain experts express the solution with concepts from their domain and also more clearly identify the changes that are needed for their use, which is propagated more quickly with code generation [**27**]. By working at a higher level of abstraction and using generators to create the executable code, the application of DSM results in significant productivity and quality improvement, reported by numerous companies [**118**]. However, the current state of tooling and practices have some limitations that slow down its adoption [**38, 119**]. Many challenges revolve around the modeling languages [**109**] and the tools used in the process. The vast majority of modeling approaches are developed without an appreciation for how people and organizations work. As a result, developers and organizations are forced to operate in a way that fits the approach instead of the approach fitting their process.

1.2. Problematic and thesis proposition

In the following, we identify four key issues regarding how current modeling editors are constructed in MDE and what this entails to domain experts using them.

The main modeling language used in the practice of MDE is the Unified Modeling Language (UML), considered the de facto standard in software engineering. It proposes several diagrams to describe different aspects of the software and enables actors of different areas to communicate cohesively. However, a recent study surveyed 50 software designers and found that these designers either did not use UML at all or used it only selectively and informally [**78**]. Yet, popular frameworks, such as the Eclipse Modeling Framework (EMF) [**30**], considered the de facto standard in the MDE community [**52**], describe model concepts using UML notations. This suggests that a universal modeling language such as UML is inadequate for the task or the target audience. A key characteristic of UML is its root in object-oriented programming (OOP). It facilitates code generation by presenting an abstraction of the code. However, this introduces considerable friction when a domain expert, such as an accountant or a healthcare practitioner, attempts to describe the concepts of his domain, which is usually unrelated to OOP concepts. Therefore, the modeling language used in current language workbenches hinders the modeling process and restricts the expression of a concept to the scope and principles of OOP.

Regarding the modeling IDEs, they have been repetitively noted to suffer in terms of usability [**24, 66**]. Modeling tools are typically designed and implemented by software engineers with little input from end-users. This has worked well with previous tools, such as IDEs and text editors since the end-user is also a technical expert. However, this approach fails in the context of DSM, where the domain expert (end-user) is left with an unfamiliar environment, leading to a poor user experience. Every aspect of the tool must be designed with the user in mind to ensure a good user experience. This requires doing a considerable amount of research on the users, using techniques such as personas [**1**], and giving them a central focus in the development of the IDE. In this mindset, operations such as code generation are considered as a support to the activity instead of driving it.

Most language workbenches use a parser-based approach, where an abstract syntax tree (AST) is progressively built by scanning the input and validated against a grammar. They either support a textual syntax, such as Xtext [**23**], or a graphical syntax, such as MetaEdit+ [**96**]. Since various experts may use a model with different needs, it should be possible to present the model in different forms, equivalent to one another. In contrast, with *projectional editing* the AST is modified directly; thus enabling support for multiple notations easily interchangeable and recomposable. It is an evolution of the structural editor where following each interaction with the program, the editor incrementally compiles and executes the resulting AST. Presently, the most promising projectional editor in the MDE community is Jetbrains MPS [**15**]. However, it is a heavy-weight language workbench that cannot be easily integrated with other tools and suffers from usability [**113**].

Lastly, a much-requested feature by practitioners, according to Agner and Lethbridge [**2**], is to have a web-based solution. It enables ease of distribution as no installation is required from the user, ease of integration with other tools, and greater cross-platform compatibility between operating systems and devices [**41, 49**]. The attraction of the web has led most tool builders to develop their new tools to run directly on the web or by leveraging web technologies through a platform such as Electron [**50**]. Currently being developed in the Eclipse community is the IDE Eclipse Theia and the diagramming framework Eclipse Sprotty [**90**], both with web technologies. However, none of the web-based language workbenches, such as AToMPM [**105**] and WebGME [**65**], support projectional editing. The only attempt at web-based projectional editor was Màs [**17**], which has been abandoned since 2014 [**9**].

To address these four issues, this thesis proposes a **web-based modeling editor that offers a user-friendly environment, easily integrable through projectional editing**. In our approach, the domain concepts are defined with structures and languages unrelated to any programming paradigm, considering modeling as an activity of its own. Every functionality and action reachable through the user interface is justified and conceived to ease the user in his task. We are not aiming for a fully-featured solution, such as a framework or an IDE. Instead, we propose a small library that can easily be integrated into a larger web application. This proposition is materialized in **Gentleman, a lightweight web-based projectional editor**, detailed in this thesis.

1.3. Contribution

The work presented in this thesis aims to provide a solution to make modeling more accessible to domain experts and practitioners by providing them with a user-friendly modeling tool using projectional editing. It builds on the work presented at the MODELS 2020 conference [**55**], where Gentleman was first showcased. Gentleman is developed as an open-source project available on GitHub[1]. The contributions of this thesis are:

- A concept-based approach, unrelated to any programming paradigm, to define the model structure. The approach aims to reduce accidental constraints induced by depending on a programming paradigm.
- A component-based approach to create projections, which offers specialized elements that help the user understand the content and manipulate the model.
- Gentleman, the implementation of our solution, as the proof of concept.
- A user study that evaluates the effectiveness of our approach in comparison with a state-of-the-art tool, Jetbrains MPS.

1.4. Outline

This thesis is structured as follows. First, in Chapter 2, we introduce the core notions used throughout the thesis with an emphasis on projectional editors. Following that, in Chapter 3, we present Gentleman, its design philosophy, architecture, and some implementation details. Having covered the high-level notions and the design of Gentleman, Chapter 4 and Chapter 5, explore in-depth the two main components of Gentleman, namely concept and projection respectively. In Chapter 6, we show the effectiveness, usefulness, and applicability of our approach based on a user study, and its interoperability demonstrated with its integration

[1] `https://github.com/geodes-sms/gentleman`

into the web application ReLiS. Finally, we conclude in Chapter 7 and present an outlook of areas that could be explored following the work presented in this thesis.

Chapter 2

Background and State of the art

In this chapter, we expose and familiarize ourselves with the key notions of MDE relevant to the work developed in this thesis, namely modeling languages and modeling editors. Particular attention is given to projectional editors and the core concepts of projectional editing, which is central to our solution. We end the chapter by reviewing some related work.

2.1. Model-Driven Enginneering

The practice of MDE relies heavily on the use of models and DSLs [**118**]. A model can be defined as an abstraction of an aspect of a system with respect to a specific intention [**69**]. A DSL can be defined as a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [**110**].

2.1.1. Modeling

As the systems that we develop get more complex [**75**], we use models to analyze them at different levels of abstraction. However, to provide an accurate picture of the system, diverse models aimed at different tasks and used by people of different backgrounds will be combined. Therefore, an essential activity of software engineering is to work on notations and methods for developing such concrete models [**60**]. A standard in that regard is UML, a family of graphical notations that help in describing and designing software systems [**26**].

2.1.2. Domain-Specific Language

A DSL provides a notation tailored towards an application domain and is based on the relevant concepts and features of that domain [**111**]. It is useful for a limited set of tasks, in contrast with GPLs that are supposed to be useful for much more generic tasks [**44**]. Being small in scope, hundreds of DSLs exist today, with a subset prescribed to software

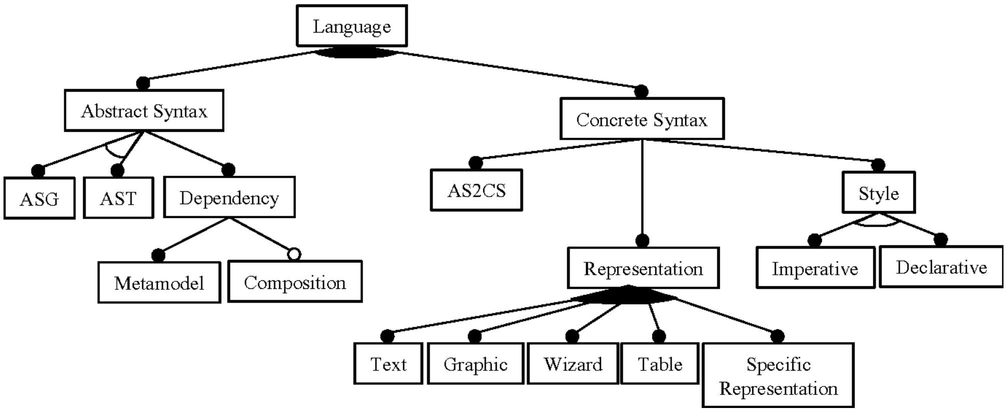


Fig. 2.1. Language Features (from [**57**])

engineering activities. On the programming side, we have classic examples like PIC, CHEM, LEX, YACC, and Make, described in [**6**]. At a higher level, well-known examples include SQL used to query databases [**68**], ATL used to define a model transformation [**45**], and HTML used to publish information on the web [**83**]. As with any formal language, void of ambiguities, defining a DSL requires a formal syntax, maintained by an abstract syntax and a concrete syntax as presented in Fig. 2.1. Semantics will be associated with that syntax but are left out of the diagram as only static semantic, included in the abstract syntax, are considered for the work presented in this thesis.

Abstract syntax. The abstract syntax is a data structure that can hold the semantically relevant information expressed by a model [**114**]. It defines the concepts of the DSL and their relations and may define constraints of the domain. It can be expressed either by a context-free grammar [**51**] or with a metamodel as described in the following section.

Concrete syntax. The concrete syntax defines the notation with which we can express models in a human-usable form [**57**]. It is only concerned with the appearance (representation) of the syntax, such as textual and graphical, as seen in Fig. 2.1. As an example, Listing 2.1 and Listing 2.2 represent the encoding of the same phonebook instance, with the first presenting it using XML and the second using an easily readable custom notation. However, although they have different concrete syntax, they both share the same abstract syntax to describe an address book. Thus, we have a one-to-many relationship between abstract syntax and concrete syntax, such that multiple concrete syntaxes may be defined using the same abstract syntax.

```
    <Phonebook>
        <Group name="Favorite">
            <Person name="Renault" number="3430993253" />
            <Person name="Anne" number="7492450903" />
        </Group>
        <Group name="Recent">
```

```xml
        <Person name="Pierre" number="5323130950" />
        <Person name="Bob" number="5723456902" />
    </Group>
</Phonebook>
```

Listing 2.1. XML notation of an address book

```
Phonebook

Group - Favorite:
 - Renault: 3430993253
 - Anne: 7492450903

Group - Recent:
 - Pierre: 5323130950
 - Bob: 5723456902
```

Listing 2.2. Custom notation of an address book

2.1.3. Metamodeling

The abstract syntax is usually specified by a metamodel, which may be defined as a model of a model [**54**]. To understand this relationship, we present in Fig. 2.2 the four-layered architecture defined by the OMG. The layers in the four-layered architecture are called M0, M1, M2, and M3. Every layer is an instance of the layer above except for layer M3. M3 is specified reflexively and, therefore, does not need layers above. At the lowest level, M0, we have the element that is manipulated by the end-user, such as a Mind map instance open on the screen, to brainstorm some ideas. Above this level, at M1, we have the model representing the objects of the program. It may be presented as an Object diagram. Above this level, we have the model labeled as M2, which is very often represented using UML class diagrams notations [**82, 4**]. Lastly, at the top, at M3, we have the self-defined model, which is used to create M1 models, such as the UML language. It is to be noted that this architecture may be generalized to n-level, such as M_i, M_{i+1}, ... M_n.

Example. To illustrate this process, let us consider a Mind map example. A mind map is a structure used to organize information *topics* linked to and arranged around a *central topic*. The metamodel depicts a tree-like organization of main topics and sub-topics. Any topic can be assigned a *marker*. If we are to create a metamodel, we would usually employ the UML notation, as seen in the Mind map metamodel presented in Fig. 2.3. There, we define types, relations, and static semantics of the language (encoding well-formedness rules). Classes represent the entities of the language such as *Mindmap* or *Topic*. They can contain attributes to retain relevant characteristics of the class, such as the *title* of the mind map

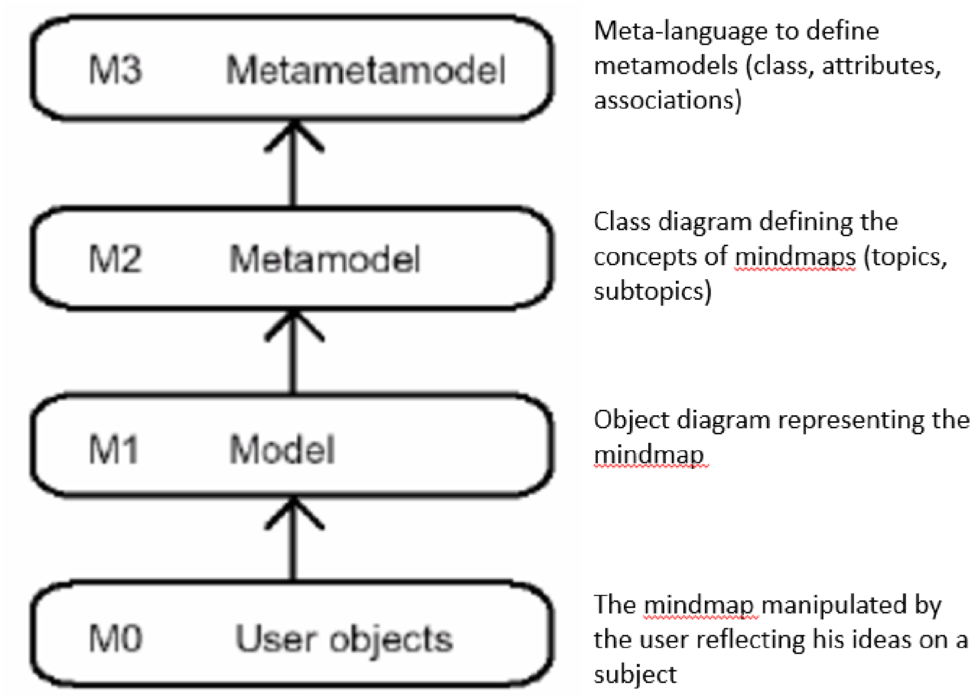

M3 Metametamodel
Meta-language to define metamodels (class, attributes, associations)
M2 Metamodel
Class diagram defining the concepts of mindmaps (topics, subtopics)
M1 Model
Object diagram representing the mindmap
M0 User objects
The mindmap manipulated by the user reflecting his ideas on a subject

Fig. 2.2. Four-layered architecture

or the name of a topic. Classes can be related using different kinds of relations to indicate the nature of the relationship. It might be a simple association, such as the one between a *topic* and a *marker*, a composition to indicate a containment, such as between *central topic* and *main topic*, and it might be a generalization, such as the relation between *topic* and all three other specific topics.

Class diagrams present the structure of the metamodel and restrict the allowed types, but complex constraints are typically expressed as Object Constraint Language (OCL) constraints [**13**]. For example, constraints may specify that the *symbol* of a marker must contain at least two characters, or that a topic cannot refer to a marker used by its parent. Alternatives to UML include Ecore [**102**] and KM3 [**44**]. Ecore, used in EMF, uses a standard textual notation: emfatic. KM3 offers a lightweight textual metamodel definition language that makes it easy to create and modify metamodels.

2.1.4. Concrete syntax definition

As seen in Fig. 2.1, the concrete syntax may be defined using various visuals, such as text or graphic. In all cases, it is built in relation with the abstract syntax.

Textual syntax. A textual syntax is defined using a grammar. It is the formal definition for a concrete syntax. A grammar is composed of *production rules* that define how valid textual

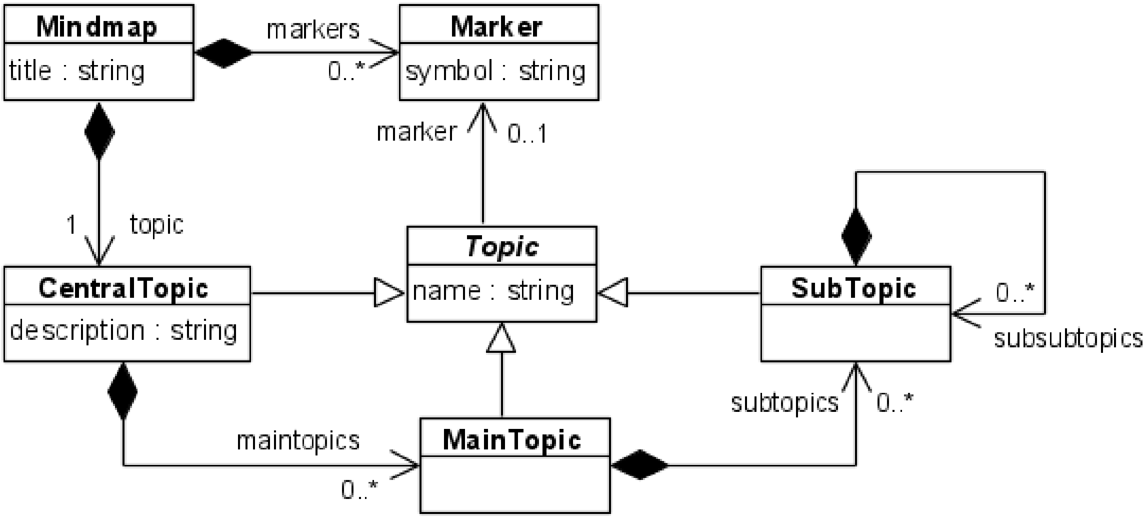


Fig. 2.3. Mind map metamodel using UML class diagram

input look like. The grammar may then be used to create or generate the parser, which will validate the entry of the end-user against the grammar, using a framework such as ANTLR [**76**]. To build such a concrete syntax, we could use Xtext, a framework that makes it possible to quickly develop tooling for a textual language [**23**]. As an example, Listing 2.3 presents a grammar developed in Xtext to represent the Mind map metamodel described in the previous paragraph. With Xtext, terminal rules are described using Extended Backus-Naur Form-like (EBNF) expressions.

```
grammar org.udem.mmap.MMap with org.eclipse.xtext.common.Terminals

generate mMap "http://www.udem.org/mmap/MMap"

MindMap:
    'Mindmap' title=STRING
    ('(' markers+=Marker (',' markers+=Marker)* ')')?
    'Topic' topic=CentralTopic
;

Marker:
    'marker' name=ID
;

Topic:
    (marker=[Marker])?
    subject=STRING
;

CentralTopic:
    topic=Topic
    (’includes’ ’{’
        maintopics+=MainTopic+
    ’}’)?
;

MainTopic:
    ’-’ topic=Topic
    (’includes’ ’{’
        subtopics+=SubTopic+
    ’}’)?
;

SubTopic:
    ’-’ topic=Topic
    (’includes’ ’{’
        subsubtopics+=SubTopic+
    ’}’)?
;
```

Listing 2.3. Textual concrete syntax for creating Mind map using Xtext

Each production rule target a class define in the metamodel. Keywords are presented as terminal symbols distinguished by a quotation mark. The attributes declared in the underlying class are accessible in the production rule and presented as non-terminals. The containment reference is expressed using the notation « variable+=ClassName ».

Graphical syntax. A graphical concrete syntax (GCS) is defined with three elements: graphical symbols (e.g., rectangles, circles), composition rules (e.g., nesting of elements) and the mapping of the graphical symbols to the elements of the abstract syntax [**10**]. Current graphical modeling editors use modeling canvas, which allows the positioning of model elements in a two-dimensional raster. There are three approaches to develop a GCS: mapping-based, annotation-based, API-based.

A **mapping-based** approach defines an explicit mapping model between abstract syntax, and concrete syntax. This approach is followed by the Graphical Modeling Framework (GMF) [**31**], where the language engineer has to define: a .gmfgraph model which defines the graphical symbols, a .gmftool model which specifies the tool palette, showing which icons are used to produce which model elements; and finally, a .gmfmap model which actually defines the mapping.

In an **annotation-based** approach, the metamodel is annoatated with concrete syntax information. This approach is supported by EuGENia [**53**], which allows us to annotate an Ecore-based metamodel with GCS information by providing a high-level textual DSML. This annotated information is used by a dedicated transformation component to generate the aforementioned GMF models.

An **API-based** approach describes the concrete syntax using a programming language, which uses a dedicated API for graphical modeling editors. This approach is taken by Graphiti [**100**], which provides a powerful programming framework for building graphical modeling editors. A language engineer has to extend the provided base classes of Graphiti to define the concrete syntax of a modeling language.

2.1.5. Editor generation

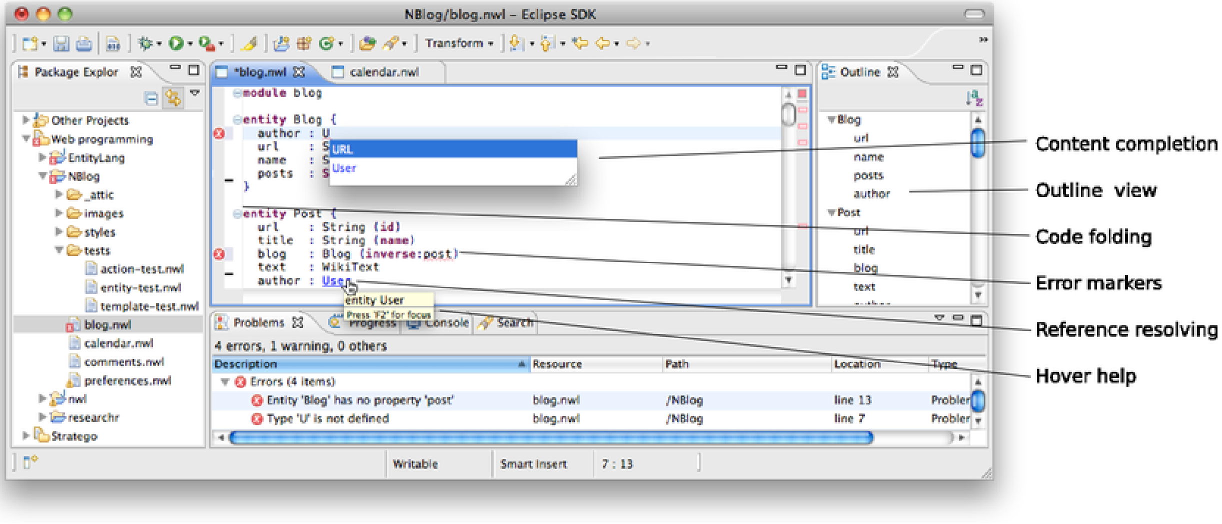
Having now fully defined a DSL, we need an editing environment to exploit it. For this purpose several tool have been introduced. The model editor is generated from the abstract and concrete syntax, with all the necessary artefacts to validate the entry of the user. As an exmaple, it could generate an analyzer that tokenizes the elements, a parser that exposes the underlying structure in a form of an abstract syntax tree (AST), a serializer that converts the model into a persistent format, an API that describes the language, and code generators to transform the model in a target language.

2.2. Language workbench

The activities described in the previous section have inspired a category of tools, commonly labeled as *language workbench* [**25**]. They are tools that support the efficient definition, reuse, and composition of languages and their IDEs [**21**]. A language workbench should present the following characteristics:

- Users can define languages, which are fully integrated with each other.
- The primary source of information is a persistent abstract representation.
- Language designers define a DSL in three main parts: schema, editor(s), and generator(s)
- Language users manipulate a DSL through a projectional editor.
- A language workbench can persist incomplete or contradictory information in its abstract representation

One key aspect of their differences lies in their modeling editors, which may support a subset of concrete syntax types. We distinguish the following three types of editors: free-form, syntax-directed and projectional.



```
module blog

entity Blog {
  author : U
  url    : S
  name   : S
  posts  : S
}

entity Post {
  url    : String (id)
  title  : String (name)
  blog   : Blog (inverse:post)
  text   : WikiText
  author : User
```

4 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Errors (4 items)				
Entity 'Blog' has no property 'post'	blog.nwl	/NBlog	line 13	Probler
Type 'U' is not defined	blog.nwl	/NBlog	line 7	Probler

Fig. 2.4. Editor services for a web language (from [**46**])

2.2.1. Free-form editors

A free-form editor uses a parser-based approach that comes with no perceived restriction when editing the program. Users can edit their programs on a blank canvas freely. However, without any restriction, or guidance, the user must be familiar with the grammar and will likely introduce syntactic and semantic errors. Most free-form editors are designed for modeling languages with a textual concrete syntax, though some graphical editors also permit it. In these (textual) editors, users enter sequences of characters into a text buffer, which is parsed to check whether the sequence conforms to the associated grammar. The parser ultimately builds an abstract syntax tree (AST), updated after each input [**112**]. Elements that do not conform to the AST are signaled to the user with visual cues, such as underlining the point of failure in red. Throughout the years, free-form editors have been greatly improved and have been augmented with helpful services. This approach has been successfully implemented in many IDEs for programming, and modeling editors, such as Xtext presented in the previous section, and Spoofax [**46**], which is shown in Fig. 2.4. The figure presents many common editor services, such as syntax coloring, error markers, content completion, as well as the general organization of the environment.

2.2.2. Syntax-directed editors

A syntax-directed editor is language-dependent and can perform syntax analysis during editing [**16**]. It is an improvement upon the free-form editor, where the syntax analysis comes late after the input. While the user is editing the program (or model), the editor can detect invalid edits that violate the AST and semantical constraints instantly, which makes it less likely to have syntactic errors. Typically, syntax-directed editors rely on incremental

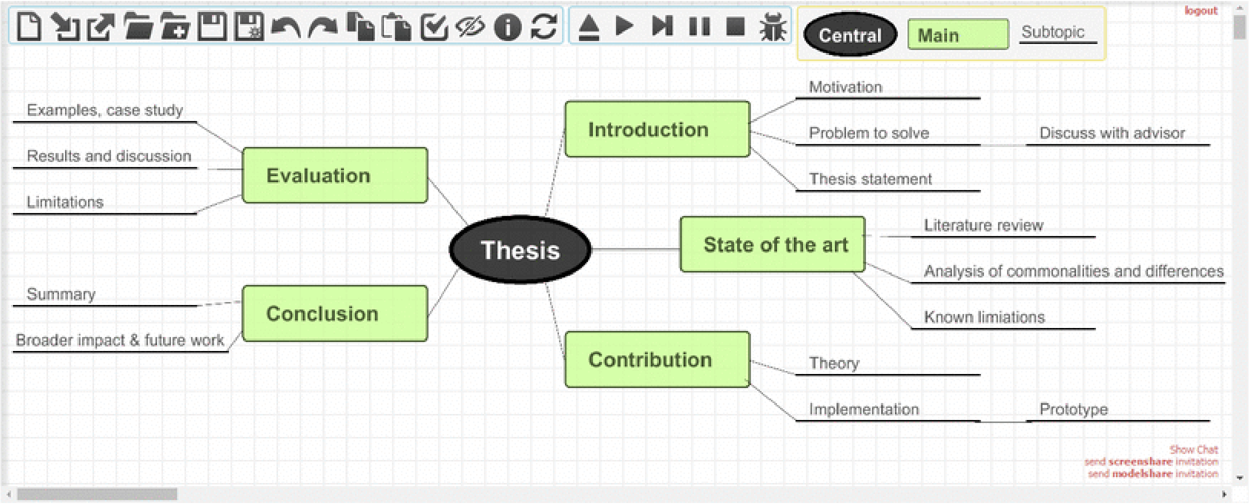


Fig. 2.5. AToMPM user interface showing a domain-specific environment for modeling Mind Maps (from [**28**])

parsing to only parse the modified portion of the program or model. This approach has been successfully implemented in many graphical editors, such as MetaEdit+ [**96**] and AToMPM [**105**], shown in Fig. 2.5. As this is a graphical editor, there is a panel at the top, exposing the different forms and shapes used in the syntax.

2.2.3. Projectional editors

Both of the previously mentioned approach are parser-based, differing mostly on the parsing technique. However, as long as there is a strong concrete syntax that requires parsing, there will always be room for errors (although small) and it will be difficult to evolve the language as the concrete syntax has value in this approach. A projectional editor, on the opposite side, does not rely on parsers. As a user edits a program, the AST is modified directly. Projection rules are used to create a representation of the AST with which the user interacts, reflecting the resulting changes. [**112, 113**]. Therefore, defining a projectional editor invovles the definition of projection rules that map language concepts to a notation. Without a parser, it enables the support of notations that cannot be easily parsed, such as tables or mathematical formulas, and the composition of any language without introducing syntactic ambiguities. As demonstrated in [**7**], this is much harder to achieve with parser-based tools. In projectional editors, every program element is stored as a node with a unique ID (UID). Since the model is stored independent of its concrete notation, it is possible to represent the same model in different ways simply by providing several projections. Different viewpoints of the overall program can be stored in one model, but editing can still be viewpoint-specific. References are established during program editing by directly selecting reference targets from the code completion menu. This is in contrast to parser-based environments in which a reference is expressed as a string in the source text and a separate name resolution phase resolves the target AST element.

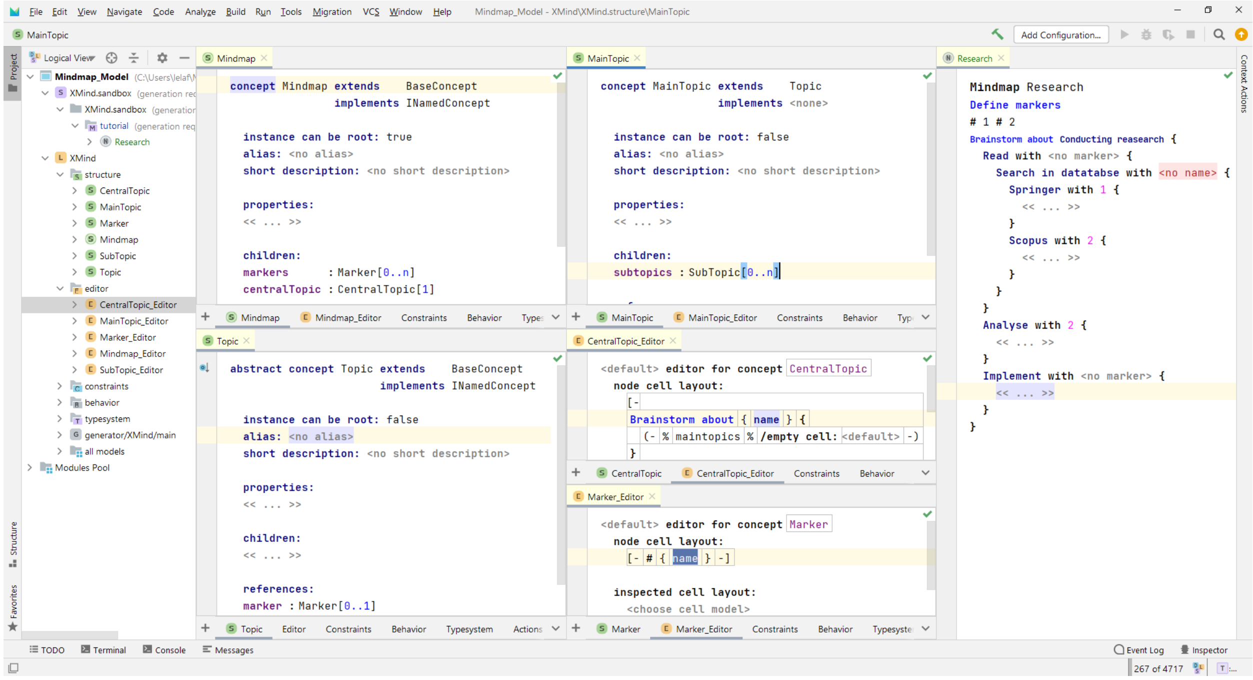
2.3. Projectional editing

Projectional editing is perceived today as a novel idea, bringing new attractive solutions to modeling editors. However, it was conceived in the 1980s, and initially introduced with the Incremental Programming Environment (IPE) [**67**]. IPE used a structural editor for programmers to interact with the program and then incrementally compiled and executed the resulting AST. The programmer communicated with the editor in terms of language constructs, like an "if" statement, which would add a (fillable) template to the code. However, as a text-based approach, it suffered in performance and usability. Other early works, such as the GANDALF project [**73**] and the Synthesizer Generator [**85**], improved slightly on the approach by generating projectional editors, but either had the same usability problems as IPE or avoided the use of projectional editing at the fine-grained expression level [**7**]. The idea was revisited with intentional programming [**94**], implemented in Intentional Domain Workbench, which inspired the rise of language workbenches where projectional editing resurfaced. Current solutions include Jetbrains MPS [**15**], the Whole Platform [**97**], and Scratch [**86**], which are explored in the following sections.

2.3.1. MPS

The Meta Programming System is a projectional language workbench developed by Jetbrains and available as an open-source software under the Apache 2.0 license. It has accumulated a fair share of success and is considered, in recent times, as the prime example of projectional editing. As a complete language workbench, it is possible to define and use a DSL, with the behavior added programmatically through a Java-based language. However, both the language definition and its manipulation is closed within MPS. Fig. 2.6 gives us an overview of the editing environment of MPS, in this case, to create a DSL to manipulate Mind maps. In the left panel, the IDE presents the organization of the project and gives the user access to the different aspects of the DSL, such as the structure, the editor and the constraints. In the right panel is a sandbox editor, which allows the language engineer to test the language as he is building it. The four central panels show the structure of some of the language concepts and the definition of their corresponding editor. To define a language, MPS divides a language into distinct aspects, with some aspects detailed below.

2.3.1.1. Structure. The definition of a language concept starts with its structure, as all other aspects refer to the structure of concepts. In MPS, the structure is equivalent to the metamodel of the language. A language concept consists of a name, child concepts, references to other concepts and primitive properties, such as integer, boolean, string, or



```
concept Mindmap extends    BaseConcept
                implements INamedConcept

  instance can be root: true
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  markers      : Marker[0..n]
  centralTopic : CentralTopic[1]
```

```
abstract concept Topic extends    BaseConcept
                       implements INamedConcept

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  marker : Marker[0..1]
```

```
concept MainTopic extends    Topic
                  implements <none>

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  subtopics : SubTopic[0..n]
```

```
<default> editor for concept CentralTopic
  node cell layout:
    [-
    Brainstorm about { name } {
      (- % maintopics % /empty cell: <default> -)
    }
```

```
<default> editor for concept Marker
  node cell layout:
    [- # { name } -]

  inspected cell layout:
    <choose cell model>
```

```
Mindmap Research
Define markers
# 1 # 2
Brainstorm about Conducting reasearch {
  Read with <no marker> {
    Search in datatabse with <no name> {
      Springer with 1 {
        << ... >>
      }
      Scopus with 2 {
        << ... >>
      }
    }
  }
  Analyse with 2 {
    << ... >>
  }
  Implement with <no marker> {
    << ... >>
  }
}
```

Fig. 2.6. MPS environment overview

enumeration. A concept may also extend another concept and implement any number of concept interfaces, like in OO. Similar to UML, external relations, such as children and references, are defined with a multiplicity to signify whether they are optional or represent a collection, in which case the multiplicity can act as a cardinality constraint. In the mind map example presented in Fig. 2.6, a Mind map concept implements the interface *INamedConcept*, which defines a name property. Therefore any Mindmap instance will possess an assignable name property. The concept also has two children: *markers* which may hold multiple Marker and *centralTopic* which must possess a CentralTopic. Like the concept *MainTopic*, it extends the abstract concept *Topic*, and therefore inherits its structure, which defines the optional *marker* reference.

2.3.1.2. Editor. In MPS, projections, which play the role of the concrete syntax, are called editors. They define how an instance of a concept is visually represented and how to interact with it. Each concept has exactly one editor unless a concept inherits the editor from a parent concept. An editor is organized as a collection of editor cells, which may contain static textual elements, fields linked to the underlying concept properties, and children cells. The editor aspect also defines some actions and keymaps, which overrides the IDE behavior when a specific key is pressed in a given cell or when a node is deleted. The example presented in Fig. 2.6 shows the definition of the editor for the CentralTopic and Maker concepts. Each element, such as the text "Brainstorm about", the character "{" or the property *name* occupy a single cell. We note that the environment and the notation strongly encourage the creation of a textual syntax, which hides the potential of projections, as a syntax agnostic approach.

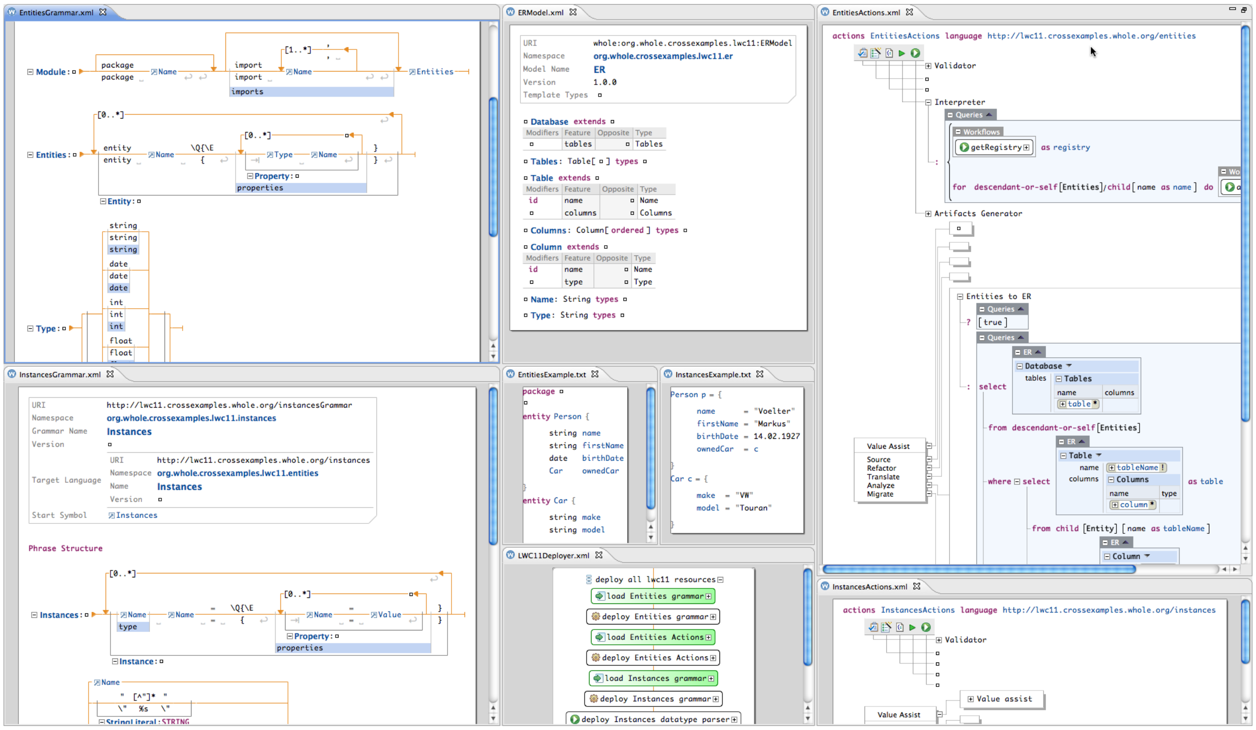
2.3.1.3. Type System. In MPS, the type system is used to specify typing rules for concepts. These include inference rules (the type of a variable reference is the type of the variable referenced by it), subtyping rules (int is a subtype of float), and checking rules. The latter are essentially Boolean expressions that evaluate any part of the model. The type system aspect may also contain quick fixes that can be used to resolve errors reported by type system rules.

2.3.1.4. Code generation. Languages in MPS also define transformation rules to lower-level languages or plain text. The generation process in MPS consists of two phases. Phase one uses a template-based model-to-model transformation engine to reduce the program code into the target language, based on reduction rules specified in the generator. The target language may be further reduced based on its own reduction rules, and so on. When no further transformation is applicable to a model, the second phase uses text generators to convert that final model into a textual program text that can then be fed to a compiler. [**77**].

2.3.2. The Whole Platform

The Whole Platform is a mature projectional language workbench distributed as Eclipse Plugin under the GPL license. It is mostly used to engineer software product lines in the financial domain due to its ability to define and manage both data formats and pipelines of model transformations over big data. The Whole Platform aims to reduce the use of monolithic languages and leverages grammar-based data formats for integrating with legacy systems. In the Whole Platform, both the built-in meta-language and newly created languages can be debugged using the same infrastructure, which has support for conditional breakpoints and variable views [**21**]. Fig. 2.7 gives us an overview of the editing environment of WholePlatform. Panels on the left show the definition of a grammar, which is used to parse the text files to create the AST and serialize the AST back into the text format. This allows a user to import an existing document by only defining the corresponding grammar. However, it also suggests that the platform only supports textual projections, more so than MPS. Panels on the right show the definition of actions, a set of model operations that can be invoked by the user, in the generated editor. They will be displayed on the editor toolbar or available in contextual menus. The central panels show examples of the generated editors.

Unfortunately, due to a lack of documentation, we cannot dive deeper into the Whole Platform.



ERModel.xml

```
URI             whole:org.whole.crossexamples.lwc11:ERModel
Namespace       org.whole.crossexamples.lwc11.er
Model Name      ER
Version         1.0.0
Template Types  □

□ Database extends □
```

Modifiers	Feature	Opposite	Type
□	tables	□	Tables

```
□ Tables: Table[ □ ] types □

□ Table extends □
```

Modifiers	Feature	Opposite	Type
id	name	□	Name
□	columns	□	Columns

```
□ Columns: Column[ ordered ] types □

□ Column extends □
```

Modifiers	Feature	Opposite	Type
id	name	□	Name
□	type	□	Type

```
□ Name: String types □

□ Type: String types □
```

InstancesGrammar.xml

```
URI              http://lwc11.crossexamples.whole.org/instancesGrammar
Namespace        org.whole.crossexamples.lwc11.instances
Grammar Name     Instances
Version          □
Target Language  URI        http://lwc11.crossexamples.whole.org/instances
                 Namespace  org.whole.crossexamples.lwc11.entities
                 Name       Instances
                 Version    □
Start Symbol     Instances
```

EntitiesExample.txt

```
package □
□
entity Person {
    string name
    string firstName
    date   birthDate
    Car    ownedCar
}
entity Car {
    string make
    string model
```

InstancesExample.txt

```
Person p = {
    name      = "Voelter"
    firstName = "Markus"
    birthDate = 14.02.1927
    ownedCar  = c
}
Car c = {
    make  = "VW"
    model = "Touran"
}
```

LWC11Deployer.xml

```
deploy all lwc11 resources
    load Entities grammar
    deploy Entities grammar
    load Entities Actions
    deploy Entities Actions
    load Instances grammar
    deploy Instances grammar
    deploy Instances datatype parser
```

Fig. 2.7. WholePlatform environment overview

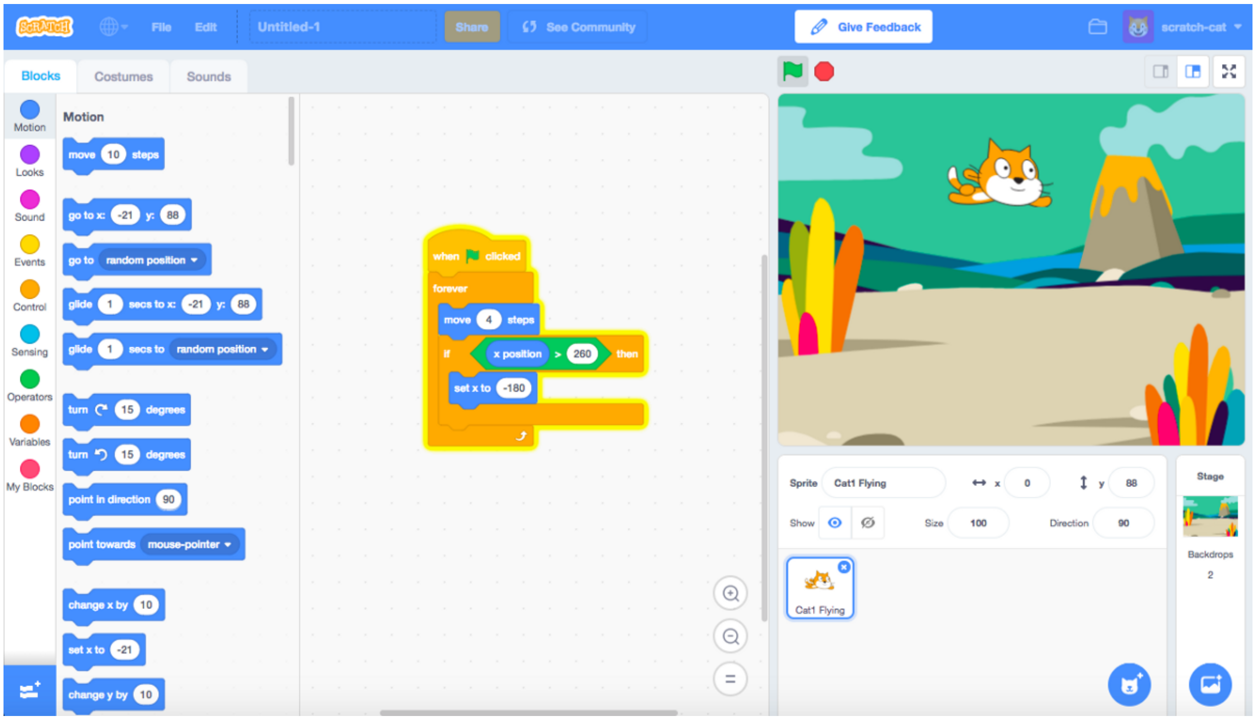
2.3.3. Scratch

Projectional editing presents an opportunity to make programming more approachable as the language no longer needs to present a cryptic syntax mostly familiar to programmers. This has tremendous educational value, notably introducing programming to a young audience. This opportunity is captured in the language workbench Scratch, a visual programming environment, which has achieved success as a tool for teaching children how to program. With Scratch, youngsters can design their own interactive media, including stories, games, animations, and simulations, by snapping together programming-instruction blocks [**11**]. Fig. 2.8 gives an overview of the editing environment of Scratch. In the left panel, the editor presents the palette and blocks that can be used to create a script, as seen in the central area. The execution of the script can be previewed and staged, as shown on the right. Multiple scripts may be created and assembled to create a more elaborate program.

As it is intended for learning programming and targets children, we will not go into the details.

2.3.4. Literature review

As an ongoing solution with over ten years of development, MPS is often discussed in the literature. To gain further insights into current projectional editing implementations, we review some of the literature around MPS.



```
when [flag] clicked
forever
    move 4 steps
    if x position > 260 then
        set x to -180
```

Sprite Cat1 Flying x 0 y 88
Show Size 100 Direction 90

Fig. 2.8. Scratch environment overview

mbeddr. The most prominent application of MPS is mbeddr, a set of integrated languages for embedded software engineering [**116**], developed with MPS. Embedded software presents several challenges, such as runtime efficiency while using adequate abstractions to provide higher expressive and reliable construction mechanism to provide safety in the program. These challenges are addressed by mbeddr which provides an extensible version of the C programming language (C99). In mbeddr, different abstractions and notations can be used in the same program. Since extensions are embedded in C programs, users can mix higher-level abstractions, using tabular or graphical notations, with low-level C code, which help significantly with managing the complexity of the developed software. However, it suffers from usability, as to end users MPS "looks" complicated, especially when we consider that many are not language engineers. It also lacks integration with legacy tools such as Eclipse EMF and MS Excel [**115**].

Notation. As a projectional editor, MPS supports multiple notations discussed in [**112**]. The first notation support is the textual notation. This allows MPS to support the syntax used by programming languages such as Java, C, or HTML. However, the textual notation causes some confusion for end-users who expect the editor to behave as a free-form editor but find themselves restricted because a projectional editor is less permissive. MPS also supports mathematical symbols, such as fraction bars and square roots, using a plugin that adds a set of new layout primitives. The plugin contributes only to the editor cells, explained in Section 2.3.1.2, so they can be integrated into arbitrary languages. Similarly, MPS supports tabular notations to represent collections of structured data or to represent two-dimensional concerns. Tables come in several flavors, such as row-oriented, where a table is built with a fixed set of columns and a variable list of rows. mbeddr makes great use of the tabular notation to produce decision tables. Lastly, MPS also supports customs cells, allowing users to create and use their own cell implementation.

Bootstrap. Bootstrapping is connected to a circularity in definition. In the context of language workbenches, this means that one language is defined using itself, or a set of languages is defined using that same set [**81**]. MPS has several meta-languages to define different aspects of a concept, and they are defined using MPS itself, making the platform bootstrapped. Many MDE language workbenches rely on bootstrapping to generate editors like eMOFLON [**58**] and AToMPM.

Efficiency. The authors of [**7**] have evaluated the efficiency of MPS with a user study. They conducted a controlled experiment with 19 graduate computer science students and industrial developers. The participants performed various code-editing activities, based on C, in the projectional editor MPS and the parser-based editor Eclipse CDT. To evaluate whether the observed differences are significant, they conducted an ANOVA or, if its preconditions were violated, a non-parametric Kruskal-Wallis test for each task. For basic editing, measured with editing operations, such as insertion and deletion, and editing errors, it is possible to be as efficient with a projectional editor as with a parser-based editor. As a note, projectional editing relied much more on code completion and yielded fewer mistakes and typos. However, experience played a factor in performing editing operations, such as selecting code. Advance editing, segmented between AST conformance and code modification and refactoring, required the user to be aware of the AST in the case of a projectional editor. When refactoring, as the participants could not rely on the visuals, they had to conceive new editing strategies, such as moving instead of copy-pasting. As a result, with MPS, advanced tasks require significantly more experience and understanding of the underlying concepts.

Usability. In terms of usability, MPS has been thoughtfully analyzed in [**113**], as a case study. The study centered around efficiently entering (textual) code, selecting and modifying code, as well as infrastructure integration. Regarding the editing experience, most of the usability issues, such as making references to non-existing nodes, selecting a block of code, copy-pasting and commenting, come from the use of a textual notation, which a user expects to behave like free-form editors. When selecting, the user usually feels misled, as the selection is not based on the visible elements but rather on the tree structure. However, MPS addresses some of them, such as the use of *intention*, to provides the user with suggestions to avoid having a dangling reference, or code completion and aliases to mitigate ambiguities introduced when composing languages.

2.3.5. Motivation for Gentleman

Projectional editing is still lacking in terms of adoption despite the current solutions described in the previous section. To better understand what is missing from projectional editors, we analyze MPS, the most documented and most discussed solution in recent papers and on forums.

Looking at the editor in itself, which is the first barrier of entry for a user, the UI of the environment looks too complex [**115**]. This unnecessarily encumbers the user as the DSL could itself be simple. Additionally, for a language engineer, there are lots of moving pieces. Simply opening a new project (i.e., creating a language), the user is presented with seven folders dedicated to different aspects of the language.

MPS concepts definition is heavily influenced by Java and the OO paradigm [**104**], such as the inheritance mechanism. Similar to UML, the structure is defined with implicit reference and multiplicity on relations. This causes friction when the domain expert tries to convert his conception into actual concepts in the editor. With this approach, the domain expert has to maintain two abstractions as well as possess knowledge about a programming paradigm that is likely to be outside of his domain of expertise.

The cell-based projections are convenient as the definition is close to the rendered result. However, stacking cells in this way results in a flat layout which might not be sufficient to represent containment intuitively. As mentioned in the previous section, although cells can accept a variety of elements, it strongly favors text by its design and features such as code folding and the sequence pattern of text.

Finally, MPS editors are desktop applications, not web applications. Therefore, the editor must be installed and updated on the machines of each user. This characteristic also impacts the interoperability factor of the editor, as the editors built with MPS cannot be integrated out of the box with web applications. Some of these concerns, such as simplifying the UI or making it easier to work with the web, could be addressed. However, we also identified some fundamental issues, such as the very definition of a concept, deeply rooted in Java and the cell-based system, much more challenging to solve. In response to these, we introduce Gentleman.

Chapter 3

Gentleman

In this chapter, we introduce Gentleman, the main contribution of this thesis. We describe the design, going from the high-level components of the architecture to the low-level artifacts of the implementation. We close this chapter by presenting the integration process and the editor services offered by Gentleman.

3.1. Rationale

Gentleman is a lightweight web-based projectional editor that provides an environment tailored to the domain expert requirements, making it possible to model at the right level of abstraction with a familiar language. It was developed with the mindset of making modeling more accessible to domain experts and practitioners. In Gentleman, a model or metamodel is structured using *concepts* and visualized and interacted with using *projections*. An environment is generated by loading a set of concepts and projections compatible with each other into the editor. The editor itself (shell and services) is also configurable to provide a more personalized experience and deeper integration into an existing application.

Metamodeling. In the MDE paradigm, current approaches to metamodeling found in popular frameworks and tools, such as EMF and AToMPM, use the UML formalism, a code-oriented approach. In this formalism, modeling is object-oriented (OO), a programming paradigm, such that a model is conceived in terms of objects [**4, 39**]. These abstractions are then translated, through a generator, into code using an OO programming language that will be completed. However, such an approach creates an abstraction of the domain by using an abstraction of the programming language, thereby requiring the domain concepts to be transformed into OO concepts such as *classes*, *attributes*, *references*, and *operations* [**79**]. This additional mapping between the perceived concepts and their expression increases the cognitive effort for the domain expert. As the domain expert might be a lawyer or a physician, he should not be expected to know a programming paradigm. To minimize this cognitive effort, a metamodel in Gentleman is defined purely with concepts unrelated to any code representation. A metamodel is then simply a graph of concepts parameterized in their relationships with each other.

Language. Creating models and interacting with their concepts requires a modeling language. As discussed in Section 2.1.4, most tools used to create a DSL use either a textual syntax or a graphical syntax. The textual approach, found in prominent tools such as Xtext sums itself as a manipulation of characters, assisted by an editor that helps in the presentation and construction of the model using techniques such as syntax coloring and context assistance. As noted previously, this approach is error-prone and does not communicate intuitively the information that it captures, requiring a considerable amount of cognitive effort. The graphical approach, found in mature tools such as MetaEdit+[**108**], shifts to manipulation of shapes and forms, making it more intuitive to the expert, albeit requiring learning the semantics of those shapes [**92**]. Furthermore, as the model gain in complexity, it becomes harder to construct and comprehend as graphic elements are spatial and therefore are constrained by the viewing area [**103**]. Additionally, the language is tied to the metamodel in both of these approaches, making it hard to change. Thus the language quickly becomes a burden for the domain expert, who has to use an increasingly inadequate language. Therefore, to embrace the diversity found in experts and the dynamic nature of their domain, Gentleman uses projections that map a visual (mixing text and graphics) to an underlying concept.

This approach gives us a dynamic syntax. The projections may be changed at any moment to visualize the concepts with a different view and may evolve independently of the underlying concept to better fit the domain expert and the task at hand.

3.2. Running example

In the remainder of this thesis, we will use a *TodoList application*, as a running example, to illustrate our presentation. Fig. 3.1 presents the metamodel of the DSL for building a *TodoList* in a UML class diagram and Fig. 3.2 presents an instance of the metamodel in Gentleman. The artifacts used by Gentleman to produce the editor can be found in Appendix B.

Metamodel. A *TodoList* is identified by a *title* and consists of at least one task. A *Task* is identified by a *name* and *description*, can be marked as *completed*, may be assigned a *due date*, and a *priority* limited to P1 to P4. A *Single* task occurs once, whereas a *Recurring* tasks may occur multiple times parametrized by a *start date*, an *end date*, and a *recurrence day.* A task may create multiple references to *labels* defined in its parent *todoList.* A *Label* is identified by a *name* and may also be assigned a *priority.*

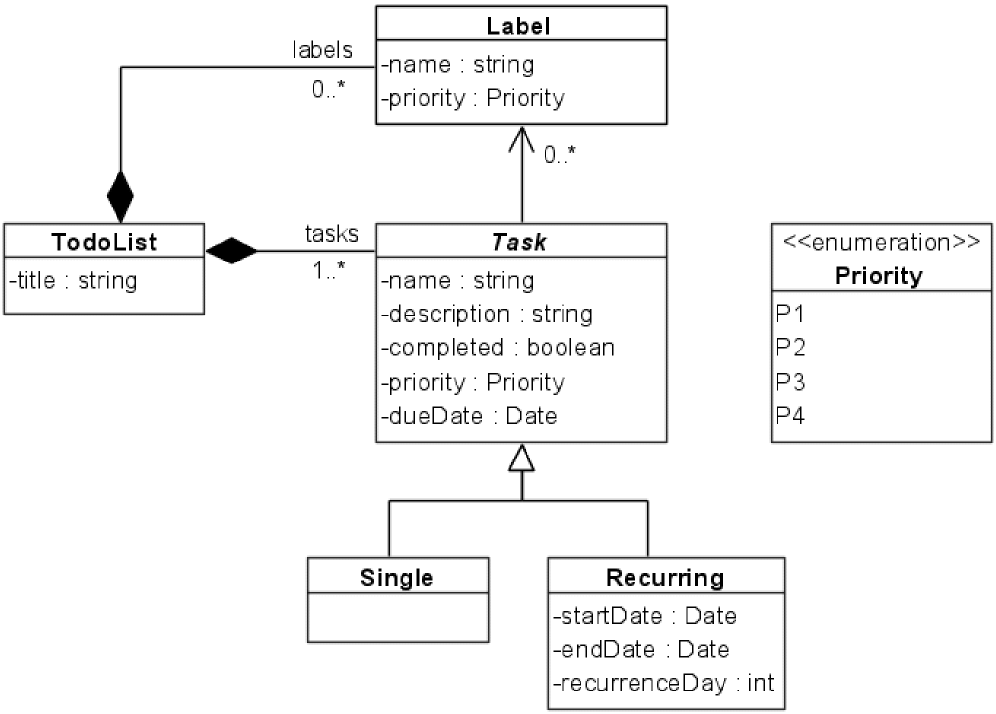


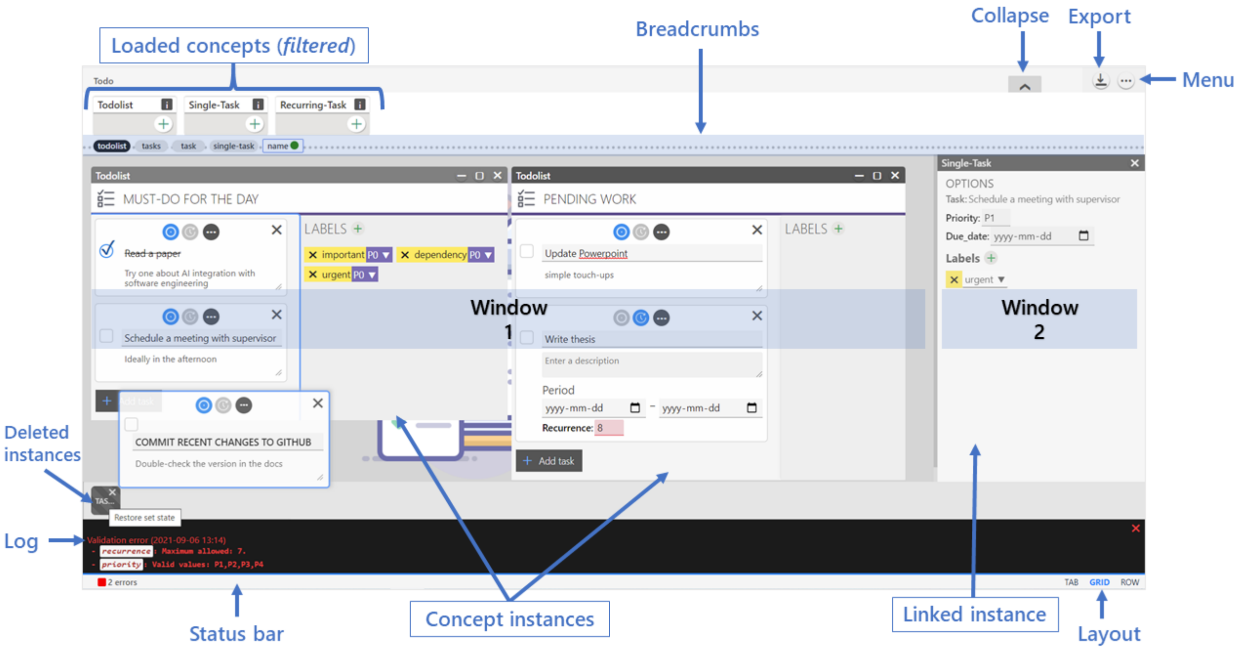
Fig. 3.1. TodoList metamodel

In terms of constraints, we have the following: the length of the *title* must be between 1 and 50, the length of the *name* and *description* must be greater than 2, and the *recurrence* must be between 1 and 7 to map correctly to a day of the week.

Model. In our model, presented in Fig. 3.2 with a grid layout, we have two *TodoList* titled "Must-do for the day" (left) and "pending work" (right), respectively. The first *TodoList*, on the left, contains two single tasks and defines three labels. The first task is marked as completed resulting in the mark shown inside the checkbox. The second list contains a single task and a recurring task with an error in the recurrence day value, which is greater than 7. Lastly, on the right, we have the task options, which are associated with the first todo list task. We also note the presence of errors indicated in the status bar and detailed in the log and a deleted task in the footer.

3.3. Architecture

The architecture of Gentleman is presented in Fig. 3.3. Gentleman is mainly composed of three modules structured as a Model-View-Controller architecture [**59**]: the *Editor module* (EM) acts as a Controller, the *Concept module* (CM) acts as a Model and the *Projection module* (PM) acts as a View. This section only presents an external view of concepts and projections in the last two modules. They are explored in much more detail in Chapter 4 and Chapter 5, respectively.



```
Validation error (2021-09-06 13:14)
 - recurrence: Maximum allowed: 7.
 - priority: Valid values: P1,P2,P3,P4
```

Fig. 3.2. TodoList model in Gentleman

The domain expert, which we will refer to as *User*, initiates a Gentleman instance which we will call *App*, by loading into the editor the *concept schema* and the *projection schema*, which hold the definition of the concepts and projections respectively. Optionally, a configuration file may also be provided to the editor for additional configuration. The editor, now initialized, keeps track of CM and PM, which are in constant interaction to keep the projections and concepts in sync with each other.

3.3.1. Editor module

The *Editor* is the entry point of the App and, as the main controller, it is the first to respond to incoming requests and messages. Incoming requests may be categorized as an API call, where a method declared in the Editor is directly invoked, or an Event call, which triggers one of the event listeners. As seen in Fig. 3.3, examples of API calls include a call to create a concept, create a projection, export the session, or load a resource, which are mapped directly to a method declared in the Editor API. Event calls are messages, internal or external, received by the Editor. External events, such as a keystroke event (key up, key down) or a mouse event (click), originate from the webpage, as opposed to Internal events, such as editor events (log created) and model events (value changed, view changed), which originate from the App itself. The Editor implements the *Facade pattern* [**29**], and as such most of the requests will be coordinated with its specialized units or delegated entirely to other modules.

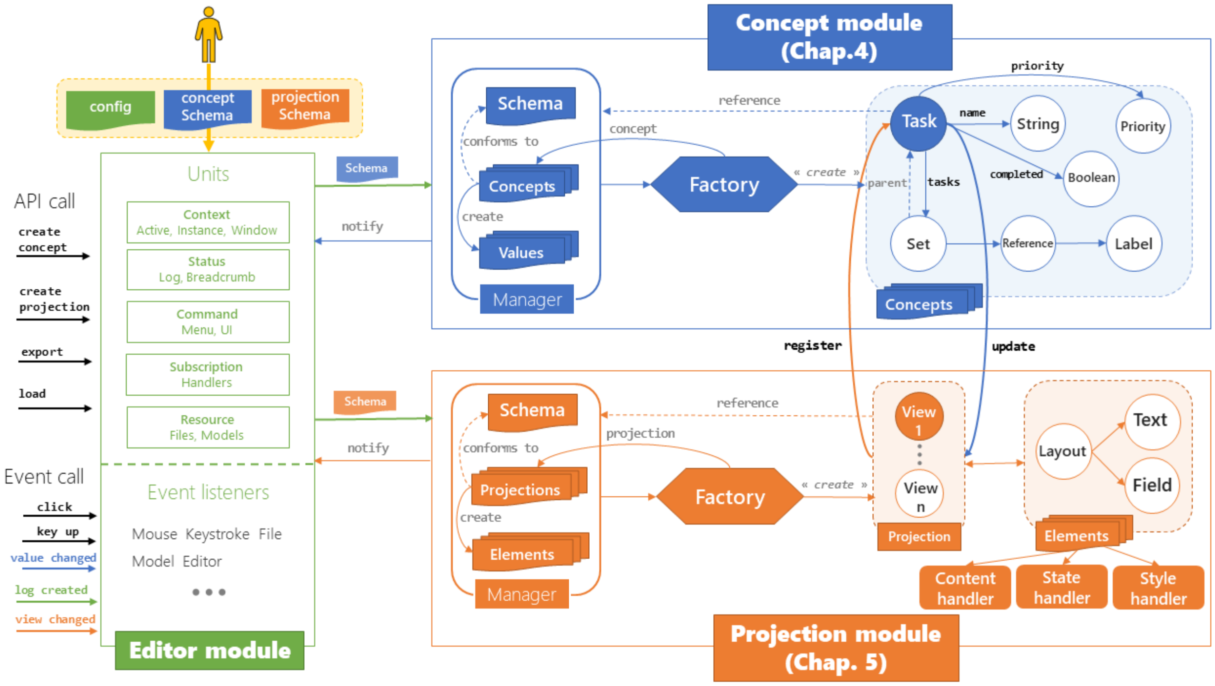


Fig. 3.3. Gentleman architecture showing some key interactions

The Editor units encapsulate distinct elements and operations, thus avoiding unnecessary coupling. They provide a *Manager*, for the Editor to communicate with, that is responsible of the interaction with the elements (data structures or UI components) controlled by the unit. A Manager has common methods to create, retrieve, update or delete (CRUD) an element as well as element-specific methods.

Context. The modeling experience truly begins when we start creating instances of the concepts loaded in the editor, the highest context of the App which comprises the whole of Fig. 3.2. As instances are created, our context becomes fragmented, illustrated here by the smaller windows within the editor. Jumping from one instance to another the Editor needs to follow along so that each operation can target the instance intended by the User. In Gentleman, there are three levels of context: instance, active , and window. An instance context is bound by the concept (root) with which it was created and its projection. This allows the concept and projection to be accessed from a closer parent than the Editor. From within the instance, numerous concepts are instantiated as seen in the TodoList context, where we find a collection of task which will receive focus at different moments of the editing activity. However, the instance context is not specific enough for some operations such as copying or removing which targets the element directly. The Active context is therefore introduced to track the active element. A common feature found in modern editors is the ability to create high-level contexts and split the view of the model, allowing the user to stay focus on parts of the model and thus be more effective. In Gentleman, windows that will contain instances may be created to fragment the App context, as shown in Fig. 3.2 where a task is displayed in the main window (window 1), and its options in a secondary window (window 2).

Navigation use-case. As we navigate through the structure, it will become increasingly difficult to keep track of our position in the model. In Gentleman, a breadcrumb is used to track the user's position, as a method recommended by [**93**]. It is located at the top of the editor as shown in Fig. 3.2. The breadcrumb is window-specific, begins at the root of an instance, and ends with the active element, thereby illustrating all three context levels.

Status. For any system with which a person interacts with, in order to provide good usability, it is critical that the User feels in control and is informed of the state of the system [**71**]. Continuing with our previously introduced breadcrumb, it informs the User of the state of the active element found at the end of the trail but indicates very little regarding the state of the other instances and the general state of the App. The Status Unit is dedicated to this very task. It monitors the state of all the instances and their elements, as well as the state of the editor. Therefore, it is the first to respond to an error found in the model or a warning signal by the editor.

Status-bar use-case. To communicate this state to the user, a dedicated status bar is presented in the footer of the Editor. In the TodoList example, while the breadcrumb indicates no errors, the status bar indicates the presence of two errors, showcasing its usefulness. It also serves as a layout manager for the user who has access to different layouts listed on the right edge of the bar. Lastly, as messages such as errors and notifications are emitted during the interaction with the App, the User should not be required to remember it all. To that effect, the status bar presents an entry to the log center where all the messages are gathered, as seen in Fig. 3.2 where it lists the errors detail.

Command. Some interactions with the App should be followed by actions, such as clicking on the menu icon which should be followed by the menu actually opening. However, many triggers may result in the same action, such as exporting the model, which can follow a click on the export icon of the App or hitting a key combination. To process this efficiently, the command unit has been dedicated to the task, as an implementation of the *Command pattern.*

Subscription. During the interaction with the App, many events may be triggered as shown in Fig. 3.3, near the bottom half of the Editor module. The ones on the left-hand side originate from outside the App and therefore are simply handled internally. However, those on the right originates from within as explained earlier, and thus are not automatically broadcast to the page environment, where another application might have reacted to. In order to provide some flexibility and control over the resulting behavior, in Gentleman, it is possible to subscribe to these events, following a *Publish-subscribe pattern.* The Handler manager provides CRUD methods to manage the subscribed handlers and defines additional methods to register or unregister a handler and trigger an event, which may be parameterized.

Resource. In a static system, everything that composes it must be provided ahead of time. Modeling being a dynamic activity, some elements will only be made available during the modeling activity. These elements that we call resources will be provided by the user when needed, and lazy-loaded in the App. The Resource Unit serves as a repository for these resources and comes with a type I manager. A special distinction is made for some resources that Gentleman recognized such as a resource that represents a collection of concepts, useful to make meta-references.

3.3.2. Concept module

Turning to our second module presented in Fig. 3.3, at the top we found the Concept module (CM). It is mostly concerned with exposing the concepts received as a schema from the Editor, managing the concept instances and preserving their values. As we can see in Fig. 3.3, CM is initialized by the editor with a concept schema, which lists a collection of concept definitions. Following that transaction, the Editor may communicate with the module to retrieve or remove a concept or query the model or instance. Now that our module is presented at the high level, let us look at the interactions, following the initialization, between the CM components and with the other modules.

Manager. The manager stores and manage the schema, the created concept instances, and their values. Although the manager is responsible for the management of the instances, their creation is delegated to a factory, as other modules might also need to create an instance. An example is found when previewing an element where the instance should not be considered as part of the model.

Factory. The factory is a direct application of the *Factory method pattern*. As shown in Fig. 3.3, it is strictly concerned with the creation of concept instances. Usually called by the manager, it receives the schema of a specific concept, and using it, it produces the appropriate concept instance. In this case, the created instance is then stored in the Manager.

Concept. Found at the end of every input request is a concept instance, referred to here as an instance. Created by the factory, an instance holds a reference to the Manager as well as the Editor in order to notify the user as quickly as possible when an error is found. Through the instance unit, a projection is registered to an instance, which is notified of every change of state of the instance. Since concepts are put in relation with one another, an instance will observe the same relation with those concept instances, as shown as the parent relation between instance Todo and String in Fig. 3.3.

3.3.3. Projection module

Turning down to our third and final module presented in Fig. 3.3, we found the Projection module (PM). Representing the view in our MVC architecture, it is responsible for the creation of all the visuals, concrete and virtual, which display the state of our model. It then follows, that it is concerned with the projections, and requests that are directly concerned with the projections are transferred to PM. Similarly to CM, it is initialized by the Editor, with a schema that lists the definition of the projections loaded in the App.

Next, we present the internal structure of the module, which resembles closely that of the CM.

Manager. The manager stores the schema received by the editor, the projections, and the components created during the session. It exposes some CRUD methods to manage the projections and their elements. As with the CM manager, the creation of projection is delegated to a factory, which is also useful for the preview scenario enunciated in the previous section.

Factory. Responsible for the creation of projections, depending on the received schema, the factory produces the appropriate projection.

Projection. The workflow of a projection is where this module differs from the previous one, as a projection is much more complex than a concept as it is expected when comparing the details of a *View* to that of a *Model*. A projection might span other projections similarly to a concept instance. However, the rendering of a projection requires three dedicated Handlers to resolve the Content, the State, and the Style.

ContentHandler. The content handler is responsible for finding the kind of content that is required by the projection or its components, such as a *Field*, a *Layout* or a *Static* element. This puts it in relation proxy position between the projection the content factories. With the kind of content resolved, it selects the appropriate factory to produce the element which will be rendered following a similar process.

StyleHandler. The style handler is responsible for resolving the styling attributes that will be applied to the rendered content. This is mostly done following a mapping system to find the appropriate CSS rule to apply. However, styling may be dynamic when using a derived value, hence the need for a dedicated handler.

StateHandler. Similar to the style handler, the state handler is responsible for resolving the state of a projection. It groups methods that will apply first-order logic to resolve the state of the projection. Here also, some terms might come from a derived value.

3.4. Implementation

With a clear understanding of the architecture, we can now tackle the implementation and complete our exploration of Gentleman's design. It is distributed as a Javascript (JS) library, written entirely in Javascript. It conforms to the ECMAScript 6 (ES6) standard which has several important features that help integration and organization such as generators, modules, and classes [**95**]. As a JS library, the web is the de facto running platform for Gentleman. On the web where available resource (space) is limited [**14**], the page size, which is an aggregation of the resources size found on the page, has a correlating impact on performance, and that is paramount to improve the User Experience (UX) [**64, 120**]. Gentleman aims to be a lightweight editor, and this characteristic is observed both in the User Interface (UI) and the size of the library. The current version of Gentleman (0.4.0) is sized at 205 KB, which is much lower than *Ace* [1], the code editor used by DSLForge which is sized at 362 KB, and comparable to a modern WYSIWYG editor such as Quill [2], sized at 211 KB. Being a pure Javascript library, Gentleman runs fully on the client-side (browser), which makes offline use possible. As with any web application, HTML and CSS are used to structure the content on the page and arrange its presentation. The editor comes with very little restriction and can be adapted to various HTML/CSS layouts and integrated into any web application.

3.4.1. Integration

The next step for a library is the installation and integration into a larger application. As a web solution, Gentleman requires no installation. It can be easily integrated into any web application with a single script (`<script src="gentleman.js"></script>`). With the script loaded, the editor can be just as easily activated in one of two ways. The simplest solution is to decorate an HTML Tag in the *body* of the page with the attribute *data-gentleman*, such as `<div data-gentleman="editor"></div>` . In this case, after loading the script, every HTML element on the page found with this attribute will have a Gentleman instance attached to it with the editor rendered inside. Alternatively, and for a more controlled integration, the instance may be created dynamically in a JS script loaded on the page as shown in Listing 3.1. Using this approach, a config may be provided and handlers may be registered.

```
const EDITOR = require('[CONFIG_FILE_PATH].json');
const CONCEPT = require('[CONCEPT_FILE_PATH].json');
const PROJECTION = require('[PROJECTION_FILE_PATH].json');

let editor = Gentleman.activateEditor(".app-editor")[0];
editor.init({
    config: EDITOR,
    conceptModel: CONCEPT,
    projectionModel: PROJECTION
```

[1] https://ace.c9.io/

[2] https://quilljs.com/

```
});
```

Listing 3.1. Gentleman dynamic integration

The ability to span multiple editors allows a parent application to have multiple Gentleman instances within a single context. This enables a web-based language workbench, like AToMPM, to control and stylize the edition of numerous attributes simultaneously.

3.4.2. Configuration

The editor options can be configured directly in the code during the initialization or through a JSON file which can be loaded into the editor at any time. The configuration options can be categorized into two segments: those that target the UI and those that target the behavior.

At the UI level, the following elements are subject to customization: the header, the menu, and the layout. The header configuration allows the title and styling to be changed for a more cohesive look as the editor will be part of an existing design. The list of concepts available for instantiation may be filtered to limit the concepts displayed to the user. The menu gives some options to style the displayed elements and choose which one are displayed or disabled. The menu may also be augmented with additional elements, handled outside of Gentleman. Lastly, the layout of the editor may be set to one of the pre-defined layouts such as *Grid* or *Tab*. As seen in Fig. 3.2, the Grid layout presents the instances on a grid, which is useful to have a global view. In the Tab layout, each instance is attached to a tab and viewed one at a time. Like the menu, custom layouts may be defined and added to the list of layouts available to the end-user.

For the behavior, the most important component is the handler described in Section 3.3.1. In Listing 3.2, we see how a handler could be defined to be executed every time an instance's value is changed. A handler may also be parameterized to be called once or under certain conditions. Using handlers, we can also extend and control the flow of validation. Our last configuration option is the storage option which could be set to an external server, where additional work might be done with the output.

```
const Handlers = {
    "value.changed": function () {
        // access the editor with ‘‘this’’
        // action executed when an instance’s value has changed
    }
};
... // create or retrieve the editor
editor.init({
    ... // other arguments
```

```
        handlers: Handlers
    });
```

Listing 3.2. Gentleman handler configuration

3.5. Editor services

With the editor in place and well understood, we can visit Gentleman main editor services, which help provide a good user experience.

3.5.1. Instantiation

Modeling with Gentleman begins with the instantiation, which is done by selecting one of the exposed concepts available in the header. A concept may be instantiated multiple times, which can be useful to create a draft or temporary copy. A key advantage to the distinct separation between concepts and projections is that multiple projections may be used simultaneously against the same concept. This brings us to the special case of instantiation: the linked instance.

Linked instance. An instance can quickly become too crowded as you modify it, and you might want to focus on a small part of it. Fortunately, because the view is a composition of projections, they can each be manipulated separately as linked instances, similar to UML aliasing [**74**]. As such, any modification done to the linked instance is reflected on the original as they both share the same underlying concept instance. Fig. 3.2 presents a linked instance of a single task in window 2, focusing on its optional attributes. It is linked to one of the task listed in window 1. As a linked instance, deleting it will have no side effects on the underlying concept instance, which is still bound to the original one.

3.5.2. Navigation

Gentleman supports both mouse and keyboard navigation. With the mouse, it's simply a matter of pointing and clicking on the element you want to focus on. With the keyboard, you can use the arrow keys to access close-by elements or the `Tab` key to iterate over the elements. The ability to create rich UI in Gentleman often results in a container-based layout (layered visual) as opposed to the flat layout found in textual editors. This impacts the navigation flow since not all elements are found on the same plane.

3.5.3. Reusing values

While editing, it is very common to find patterns that suggest reusing existing structures or values to avoid manual repetition. Copying part of a structure into another part, commonly referred to as copy-pasting, has become one of the most frequent interactions when handling editors [**22**]. Gentleman enables any part of an instance to be copied and pasted into another. As a projectional editor, only values coming from a similar concept will be applied to the target concept. Interestingly, the ease that comes with the action has developed the need for multiple slots to hold our copy, especially during a refactoring operation where some elements are temporarily removed. As a response, Gentleman allows multiple copies to be preserved at the same time.

3.5.4. Undo/Redo

Another feature that has come to be expected from editors is the ability to undo an action or reverse the undo with a redo. Although more frequent with the textual editor as the probability of making an error increase with the input rate, Gentleman attempts to responsibilize the User. It offers the ability to undo one step, but it also gives the User a visual of the deleted elements, which can be restored until permanently deleted, as seen in Fig. 3.2. Additionally, with the many saving slots, Gentleman encourages the User to use them frequently to store parts that might be useful at a later time.

3.5.5. Code assistance

Another way to assist our User is with auto-completion, a well-known technique implemented by many applications, very useful when filling out forms on the web [**121, 35**]. It has been noted to help prevent spelling errors as well as boost the confidence of the User [**117**]. As a projectional editor that follows a fill-in blank flow, Gentleman offers contextual assistance in the form of autocompletion when the input is textual (characters or symbols).

3.5.6. Search and filter

As a model gain in complexity, looking for a specific element can be quite tedious. With a filtering mechanism, such a task takes much less time and effort. As a web application, Gentleman leverages the searching mechanism found in the browser to assist the User in his task.

3.5.7. State

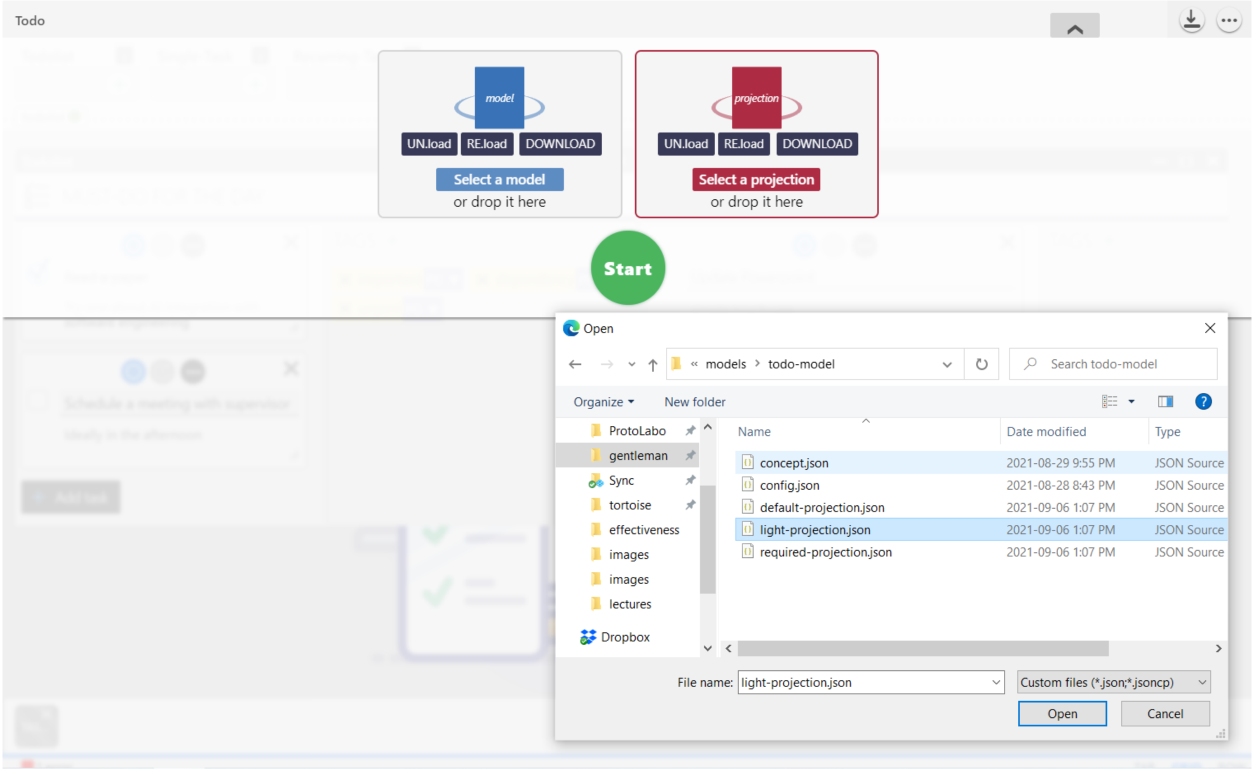
As previously described in Section 3.3.1, Gentleman offers many cues to keep the User informed of the state of the model. The breadcrumb gives the state of the active instance and, in the presence of errors, it can be queried for more details about the nature or source of the error. The element itself gives various cues such as horizontally shaking as soon an error is made or coloring the border in red. Lastly, in the footer, the status uses different icons to inform the User of the state of the model, going from a green circle to a red square when an error is found.

3.5.8. Import/Export

Gentleman does not support a storage facility for models. Storage should be handled by the application embedding a Gentleman editor. To ease this process, Gentleman offers some export options to select different formats, namely JSON, XML and Ecore (XMI).
JSON. When using Gentleman as a stand-alone application, it is possible to save the session with all the created instances as a JSON file. It is the default export format, also used to import and load a model.
XML. JSON is the preferred choice when interacting with web services. However, many applications outside the web still prefer XML for exchanging data. To interop with these applications, Gentleman offers XML as an export format.
Ecore/XMI. Lastly, we have the Ecore (XMI) format. As Ecore is widely used in the MDE community, it would be beneficial with it. As such, Gentleman offrs a migration tool to import Ecore models as well as the option to export the Gentleman model into Ecore.

3.5.9. Bootstrapping

The meta-languages used to define concepts and projections have been specified using Gentleman, making the editor bootstrapped. This technique makes it possible to create concepts and projections with a familiar language. The user may swap projections directly from the menu in Gentleman without interrupting his work or losing his created instances, as seen in Fig. 3.4. Therefore, it is essential to note that all the editor images presented in this thesis, including those used for metamodelling presented in the following chapters, are simply one possible view of the underlying concept and may be projected differently.



Name	Date modified	Type
concept.json	2021-08-29 9:55 PM	JSON Source
config.json	2021-08-28 8:43 PM	JSON Source
default-projection.json	2021-09-06 1:07 PM	JSON Source
light-projection.json	2021-09-06 1:07 PM	JSON Source
required-projection.json	2021-09-06 1:07 PM	JSON Source

File name: light-projection.json

Custom files (*.json;*.jsoncp)

Fig. 3.4. Gentleman editor menu giving access to the loaded concepts and projections

Chapter 4

Concept

In Gentleman, the structures that encapsulate the concepts used to define a model are called *concept*, inspired by MPS. As such, a model can be viewed as a graph of concepts related to one another. Gentleman only uses concepts to create the structure of a metamodel, but there are different types of concepts. Each type encapsulates different behaviors and restricts the area of action of the concept. This is similar to the differentiation found in OO regarding classification, where we might use an *interface*, an *abstract class* or a *concrete class* to communicate different intents in the model. In Gentleman, we distinguish between four types of concepts: *primitive*, *concrete*, *prototype*, and *derivative*. Complex structures, such as a task or a person, will involve many concepts. They will be put in relation in the form of *attributes* and *properties*, which will resolve to a primitive. Fig. 4.1 presents the metamodel of concepts in Gentleman, which we will explore in this chapter.

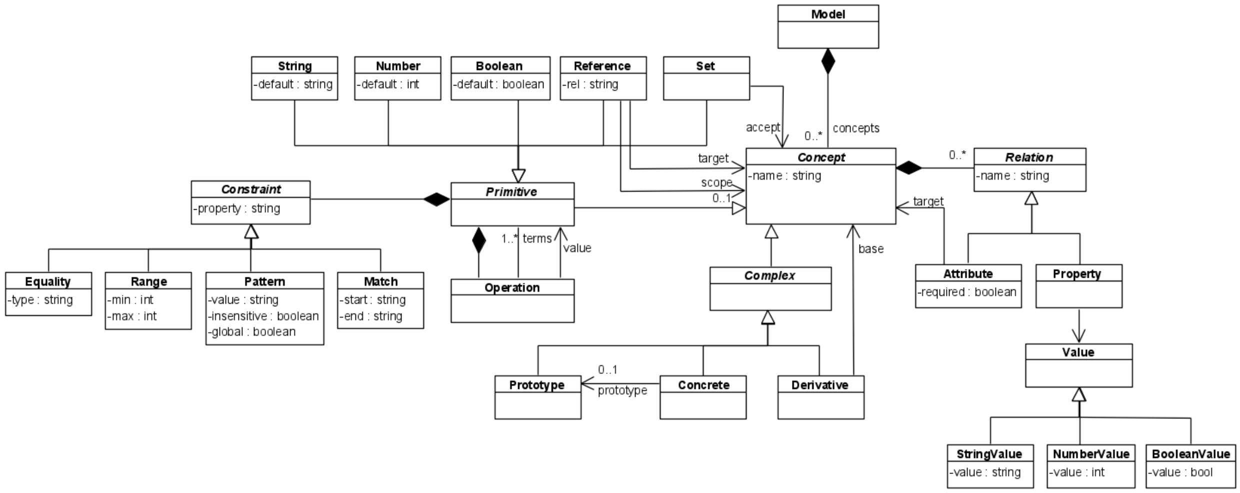


Fig. 4.1. Gentleman concept metamodel

4.1. Structure

As presented in the creation of the TodoList concepts, shown in Fig. 4.2, a concept is identified by its name and its nature, corresponding to the ones listed in the above paragraph. To allow experts to follow the naming scheme of their choice, little restriction has been put on the name, therefore allowing numbers, letters, and even special characters to be part of the name.

Relations. A concept structure is mainly defined by its relations, as shown in the metamodel in Fig. 4.1. As a concept is put in relation with other concepts, the structure becomes more complex. For any given concept, an input, like assigning a value, or a request, like verifying the presence of a value, may require the involvement of other concepts to be processed. Using the running example, if we consider the *Task* concept, which encapsulates some characteristics of an actual task when given a string representing the *name* or a boolean representing the *completed* status, it will require the primitive concepts *String* and *Boolean* respectively to be processed. To process the request of the *length* of the *title*, which is defined by a *String* concept, the *Number* concept will be needed to output the result. Thus, in Gentleman, a concept may have two types of relations. To process an input, a concept will use an external relation, named *Attribute*, to find the primitive that will process it. To process a request, a concept will use an internal relation, named *Property*, to resolve it using primitive operations.

4.1.1. Attribute

An attribute represents an extrinsic characteristic of its parent concept. The attribute of a concept describes a relation held by the parent with a target concept. It is identified by a *name* that is unique within the concept. In this relationship, the target concept may be parameterized with constraints, similar to how OCL is used to add constraints on class diagram elements. As seen in Fig. 4.2, defining constraints is simple in Gentleman as the constraints are defined within the concepts, thereby avoiding scattering the definition of a concept. As an example, in the TodoList presented in Fig. 3.1, the Priority has two associations, defined with Task and Label. However, it is likely that these two concepts use Priority differently, and they might use different default values, or one might need to restrict the priorities to P1 and P2 only. By defining the constraint in the attribute, every relation may define its own use of a concept. When using an OO-based approach to create a metamodel, a relation is defined in terms of multiplicity, which is evaluated against the value held by the object. Therefore, the relation is always present in the model, whether it is needed or not, in which case it has a *null* value. In Gentleman, an attribute may be optional, meaning that an instance of the parent concept would be valid without a relation to the target concept.

Example. In the TodoList example, every attribute and end association is represented as a concept attribute. As an example, the target concept of *name* would be the primitive concept *String*. Interestingly, the target concept of *tasks*, which is a collection, would be the primitive concept *Set*, instead of Task directly.

4.1.2. Property

A property represents an intrinsic characteristic of its parent concept. The property of a concept describes a relationship within itself. As a relation, it is identified by a unique *name*. A property draws some similarity with UML *derived attribute* and *query operation*, which is an operation that does not modify the owning class in any way [**79**]. A property may be used to define a constant value or a computed value. A constant value would take the form of a key-value pair, such as a link to the documentation or some metadata. A computed applies an operation on the current state of the concept without changing it. Whereas the attribute would make relations with any other concept in the model, a property uses only the concepts accessible by the parent in its operation. Therefore, a property will result in a primitive value.

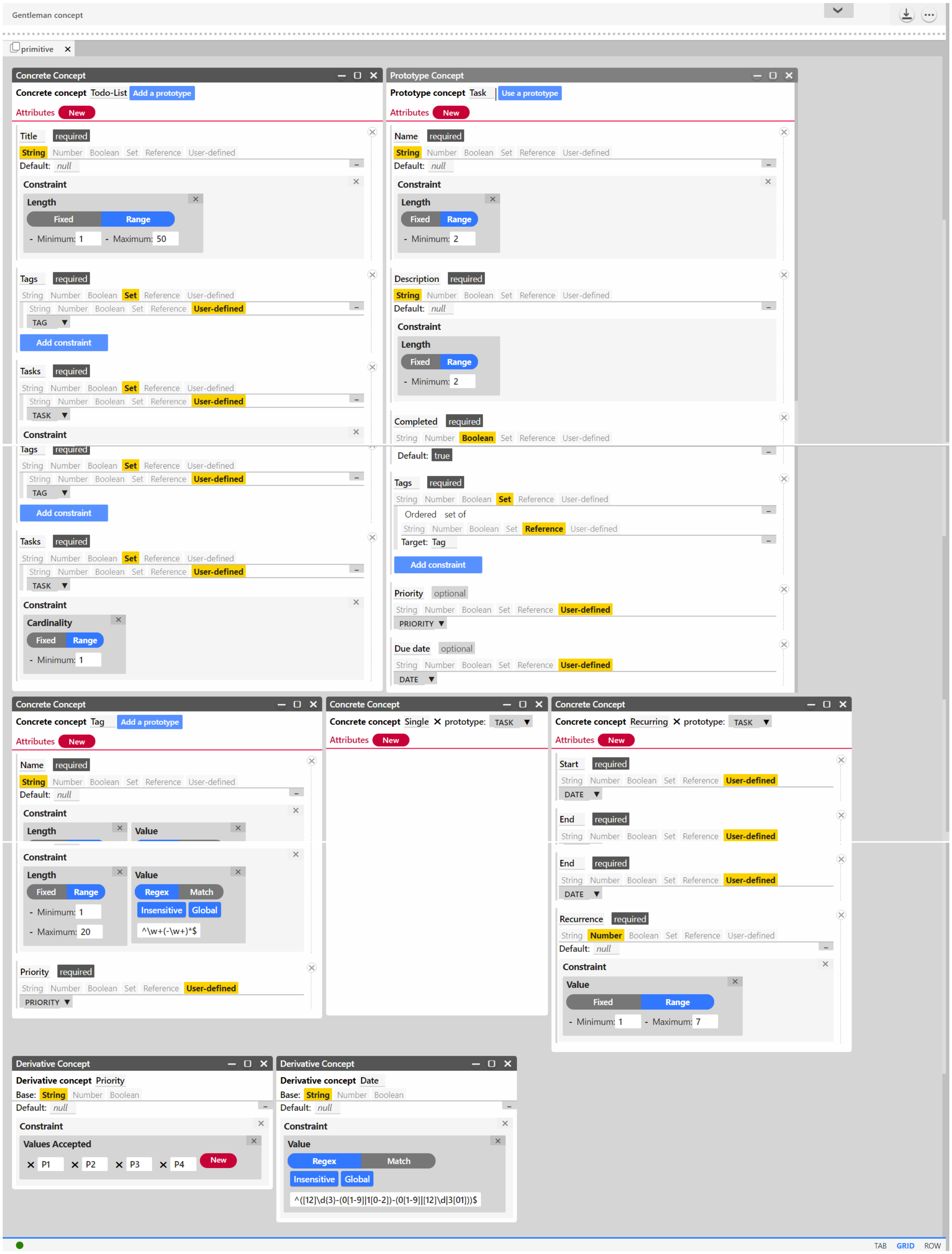
Example. In the TodoList example, we could have a *completedTasks* property, where the value is derived from the *tasks* attribute. It would return a filtered set with only the completed tasks. Another example could be found in the *Recurring* concept where we could add a *duration* property. It would return the number of days between the start date and end date.

4.2. Primitive

Primitive data types are the building blocks of most programming and modeling languages. In Gentleman, they are self-defined concepts and not related to any other concepts, i.e., they have no attributes. However, they possess properties used to query their state, such as the *length* property in the *String* concept. For better model integration, primitives are accessible globally to any model. Every concept can be resolved to a composition of primitives. Gentleman offers the following predefined primitive concepts: *String*, *Number*, *Boolean*, *Set*, and *Reference*. They contain specific properties and constraints that can be used when defining an attribute to restrict the concept.

4.2.1. String

The first primitive that is needed should give us the capacity to input characters, as this is how most computer activities are done. In Gentleman, such behavior is encapsulated by the *String* primitive concept, which gives the ability to store and manipulate characters. It



Concrete Concept
Concrete concept Todo-List Add a prototype
Attributes New

Title required
String Number Boolean Set Reference User-defined
Default: *null*
Constraint
Length
Fixed **Range**
- Minimum: 1 - Maximum: 50

Tags required
String Number Boolean **Set** Reference User-defined
String Number Boolean Set Reference **User-defined**
TAG
Add constraint

Tasks required
String Number Boolean **Set** Reference User-defined
String Number Boolean Set Reference **User-defined**
TASK
Constraint

Tags required
String Number Boolean **Set** Reference User-defined
String Number Boolean Set Reference **User-defined**
TAG
Add constraint

Tasks required
String Number Boolean **Set** Reference User-defined
String Number Boolean Set Reference **User-defined**
TASK
Constraint
Cardinality
Fixed **Range**
- Minimum: 1

Prototype Concept
Prototype concept Task Use a prototype
Attributes New

Name required
String Number Boolean Set Reference User-defined
Default: *null*
Constraint
Length
Fixed **Range**
- Minimum: 2

Description required
String Number Boolean Set Reference User-defined
Default: *null*
Constraint
Length
Fixed **Range**
- Minimum: 2

Completed required
String Number **Boolean** Set Reference User-defined
Default: true

Tags required
String Number Boolean **Set** Reference User-defined
Ordered set of
String Number Boolean Set **Reference** User-defined
Target: Tag
Add constraint

Priority optional
String Number Boolean Set Reference **User-defined**
PRIORITY

Due date optional
String Number Boolean Set Reference **User-defined**
DATE

Concrete Concept
Concrete concept Tag Add a prototype
Attributes New

Name required
String Number Boolean Set Reference User-defined
Default: *null*
Constraint
Length Value

Constraint
Length
Fixed **Range**
- Minimum: 1
- Maximum: 20
Value
Regex Match
Insensitive Global
^\w+(-\w+)*$

Priority required
String Number Boolean Set Reference **User-defined**
PRIORITY

Concrete Concept
Concrete concept Single prototype: TASK
Attributes New

Concrete Concept
Concrete concept Recurring prototype: TASK
Attributes New

Start required
String Number Boolean Set Reference **User-defined**
DATE

End required
String Number Boolean Set Reference **User-defined**

End required
String Number Boolean Set Reference **User-defined**
DATE

Recurrence required
String **Number** Boolean Set Reference User-defined
Default: *null*
Constraint
Value
Fixed **Range**
- Minimum: 1 - Maximum: 7

Derivative Concept
Derivative concept Priority
Base: **String** Number Boolean
Default: *null*
Constraint
Values Accepted
P1 P2 P3 P4 New

Derivative Concept
Derivative concept Date
Base: **String** Number Boolean
Default: *null*
Constraint
Value
Regex Match
Insensitive Global
^([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))$

Fig. 4.2. TodoList concept in Gentleman

is also found in most languages and modeling tools, such as UML where it is a primitive, SQL which has the VARCHAR type, and Excel which has the Text type.

Constraints. As a fundamental element, the domain governed by the String concept is broad. It is all the possible combinations of all the possible characters. Therefore, as we see in Fig. 4.2, through attribute, constraints may be defined on the String concept in order to limit the domain of acceptance. The constraint may target the exposed properties of the String concept, such as its value, length, or the list of values accepted in the domain.

Operations. When defining computed properties or interacting with a concept, we would like to invoke or chain operations to avoid manual computation and repetitive actions. As such, primitives, like the String concept expose some operations such as concatenation, segmentation (substring), transformation, and comparison. With concatenation, the given values are chained in the order of entry to result in a single value. With segmentation applied, a value is created by taking parts of an existing value. With transformation, a mapping is done on the characters of a value. With comparison, a pattern is used to find similarities between multiple values.

Example. In the TodoList example, we have many attributes targetting the String primitive, such as the todo list's *title*, the task's *name* and even the *Priority* (see Fig. 4.2).

4.2.2. Number

The following primitive is just as common as the String concept as it gives the ability to count and perform arithmetic operations. It is the *Number* concept, which is also defined as a primitive in UML, and found in most modeling tools. Note that a number can be any negative or positive integer or a real value of arbitrary length.

Constraints. As a foundation element, the domain governed by the Number concept is just as large. Therefore, constraints may be defined on Number to refine and restrict the accepted values. The constraint may target the exposed properties of the Number concept, such as its value, the list of accepted values, and the use of decimals.

Operations. The Number concept exposes some arithmetic operations such as those found in OCL [**63**], making it possible to perform addition, subtraction, multiplication, and division. It also gives access to comparison operations to evaluate the equality of two numbers by leveraging the JS engine.

Example. In the TodoList example, we have a single use of the Number concept, with the *recurrenceDay* attribute.

4.2.3. Boolean

The Boolean primitive is very simple when compared to the previous two in terms of domain, as it is composed of only two values. However, it becomes quite powerful when considering its operations. These operations allow us to perform boolean algebra, which is useful for creating complex properties. The boolean concept, being a small domain, defines no constraint.

Example. In our TodoList example, we have a single use of the Boolean concept, with the *completed* attribute.

4.2.4. Set

The next primitive is not as common as the previous ones and actually missing in UML and MPS as a primitive. It is the *Set* concept, which is usually left as a complex datatype in most programming languages or dealt with in execution such as in OCL. However, this creates significant friction when metamodeling with UML as the set is implicit (using cardinalities) and cannot be dealt with separately. As it is strictly defined and bound by a type, this makes it difficult for the set to evolve and accept other types, which could be dealt with by generalizing the type. However, such a generalization is not desirable as it does not reflect the domain but instead solves a technical limitation. Therefore, in Gentleman we have the *Set* primitive, defined in a similar way as *ComplexType* found in XML schema [**107**].

Constraints. The Set must have a target concept, which may be of any nature, and may be constrained to an OrderedSet, which would take the index into consideration when manipulation the set elements. It exposes a single property, the cardinality, as show in Fig. 4.2 where it ensures that the *tasks* (set) will have at least one task. It is important to note that simply having a relation, signified by the **required** attribute, does not imply the presence of a task. We could have a relationship with an empty set. The cardinality can be restricted to limit the minimum or the maximum number of elements accepted in the set.

Operations. The Set concept exposes some operations to manipulate its elements and query the set. As an example, it could be queried for the existence of an element, verify if it is empty or if it is disjointed with another set.

Example. In the TodoList example, we have two examples of Set with the list of tasks and the list of labels. As previously mentioned in the Attribute example, the TodoList concept has a relation with the Set concept which has a relation with the Task concept if we consider the task list.

4.2.5. Reference

In some programming languages, such as C [**87**], pointers are used to create references and to dynamically manipulate memory addresses. However, such ability is lost in the high level of abstraction of OO metamodeling. Similar to Set, in UML, references are left implicit, and as such, configuring them is tedious, often requiring complex OCL constraints to define their scope. This is addressed in MPS with reference constraints that allow the scope to be explicitly defined [1], to have references pointing at a restricted set of allowed targets. In Gentleman, we have a *Reference* primitive to make and manipulate references as first-class citizens, just as a string or a number. It is important to differenciate it from an Attribute, which is a relation and may not be manipulated but only navigated.

Constraints. A reference with no constraint may point to any concept, which might be helpful for annotations but not desirable for most scenarios. As we have our implicit references in a UML class diagram, we want at least the target type to be defined, which will be the first constraint to be defined. Following this is the scope, as not all elements of this type would be valid candidates. Gentleman defines the scope as a query that will be used at runtime. The query is created with the target concept and its hierarchical relationship with the reference.

Example. In the TodoList example, we have one attribute, the task *labels*, targetting the Reference concept through the Set. As enunciated in Section 3.2, we would like our tasks to refer only to label found in their parent todolist. This is achieved by simply limiting the scope to the target concept *TodoList*, which has a parent relation with.

4.3. Complex

Having seen the primitives, we can now build complex structures with them.

4.3.1. Concrete

A concrete concept represents a regular concept of the model and is comparable to a class in OO. Unlike a primitive, it is specific to a model and must have a relation with another concept.

Example. In the TodoList metamodel shown in Fig. 3.1, every declared concept, with the exception of Task and Priority, is a concrete concept, and we can see that they all present at least one relation or one attribute. The Single Task inherits the relations defined in the Task concept, as seen in Fig. 4.2.

4.3.2. Prototype

Gentleman borrows some notions from the OO paradigm, but also from prototype-based programming [**19**], as concrete concepts are the only means to model applications. A prototype creates a base skeleton to provide reusability and extension to concepts of the model. It could be seen as a non-empty starting point for another concept to build from. Any concept can reuse a prototype and would inherit its attributes, which may be redefined in the concept. Prototypes follow the *Liskov substitution principle*. If the target of an attribute is a prototype, then any concept reusing it can also be the target. In this case, any property

[1] urlhttps://www.jetbrains.com/help/mps/scopes.html

or constraint defined on the attribute would still hold. Lastly, a key advantage of having a prototype as a self-sufficient entity with its own state and behavior is to be able to use projections directly on it, which is not possible with OO-based solutions such as MPS. In this case, additional rules are needed as an abstract class cannot be instantiated.
Example. In the TodoList example, *Task* is represented as Prototype concept, reused by the *Single* and *Recurring* concept. As an example of overriding, the *name* attribute could be further constrained in *Recurring*, such that it would end with a digit representing its recurrence.

4.3.3. Derivative

We have seen how to extend and reuse complex concepts, but it is not always needed to create new relations to express a characteristic of a concept. Instead, we could specialize existing relations and combine them to create new ones. This brings us to the *Derivative* concept, in which the structure is derived from another concept, called *base*. Every value that a derivative can capture must also be valid for its base concept, a primitive or concrete concept. When the base is a primitive, it can serve as a form of specialization. As described in the previous section, constraints may be used on primitives to restrict the values that should be accepted. When the base is a concrete concept, the derivative could be used to define computed properties.
Example. An enumeration would usually be translated into a derivative. As such, in our TodoList example, *Priority* would be represented as a Derivative of String with a constraint on the accepted values.

4.4. Comparison with OO

As shown throughout this chapter, concepts in Gentleman holds some similarities to class in OO. In this section, we highlight those similarities by describing the migration of an Ecore model into a Gentleman model.

4.4.1. Modelling in EMF

First, it is important to note that EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Whereas Gentleman is being developed specifically for DSM, EMF is much larger. As such, it is missing some special metamodeling concepts such as Model. Nonetheless, Ecore is the de-facto reference implementation of OMG's Essential Meta-Object Facility (EMOF). A simplified subset of Ecore is shown in Fig. A.1 and a detailed version can be found in [**12**]. The main elements of Ecore are EClass, EReference, and EAttribute. At the root of an Ecore model is a package containing classifiers . A classifier can either be an EClass or an EDatatype. EClasses can have EReferences that express unidirectional relationships between two EClasses. An EClass can additionally have EAttributes to express properties of the EClass. The range of the attribute values is specified by a data type such as EInt, EString, or EBoolean. EFeatures can be ordered

4.4.2. Mapping

The mapping of concepts is shown in the Table 4.1. Some elements supported by Gentleman, such as properties, are not presented in the table as they have no equivalence in Ecore. An EPackage, being the root of an Ecore model, is mapped to a Model element with the same name. Note, however, that these two concepts do not share any semantic properties; they are mapped because they serve the same function as a container of the metamodel elements. An EClass can be mapped to a Prototype element, an Abstract element, or a Concrete element if it is an abstract, an interface, or neither respectively. The name and relations to the EAttributes and EReferences are preserved. They are both mapped to an Attribute element. If the EAttribute has a multiplicity higher than 1, the corresponding Attribute will have a type Set.

Ecore	**Gentleman**
EPackage	Model
EClass (abstract or interface)	Prototype
EClass	Concrete
EEnum	Derivative
EDataType	Primitive
EAttribute (many)	Set
EAttribute	Attribute
EReference	Attribute

Table 4.1. Mapping between Ecore and Gentleman concepts

4.4.3. Transformation

Gentleman supports importing a metamodel defined in Ecore into a Gentleman model. To this end, we implemented a transformation chain that can be executed directly in Eclipse. The transformation is two-fold. First, we apply a model-to-model transformation, following the mapping presented in Table 4.1, with the ATLAS Transformation Language (ATL), a hybrid model transformation language that allows both declarative and imperative constructs to be used in transformation definitions [**45**]. The transformation rules can be found in Listing A.1, with the Gentleman metamodel in Ecore presented in Fig. A.2. With the generated model, we apply a model-to-text transformation with the Epsilon Generation Language (EGL), a model-driven template-based code generator built atop Epsilon [**89**]. It serializes

the model in the JSON format supported by Gentleman. The transformation rules can be found in Listing A.2. As components of the Eclipse Modeling Project, Eclipse will be used to execute the transformations.
To apply the ATL transformation, an input and output metamodel are required. The input metamodel will be Ecore, and the output will be Gentleman Ecore model, Gentleman metamodel defined with Ecore. It will also be used in the EGL transformation.

Chapter 5

Projection

In Gentleman, as a projectional editor, the concrete syntax is defined with projections that are bound to concepts. As explained in Chapter 3, multiple projections may be defined for a concept. A projection is a light interchangeable and composable enveloppe that fits the user's view and the task requirements in order to manipulate a concept and thus a model. Fig. 5.1 presents the metamodel of projections in Gentleman, which we will explore in this chapter.

5.1. Structure

As a representation of a concept, a projection can be visualized and interacted within the GUI. Unlike concepts, which may be categorized by type, as described in Chapter 4, projections have no such distinction. A projection is the collection of views composed of

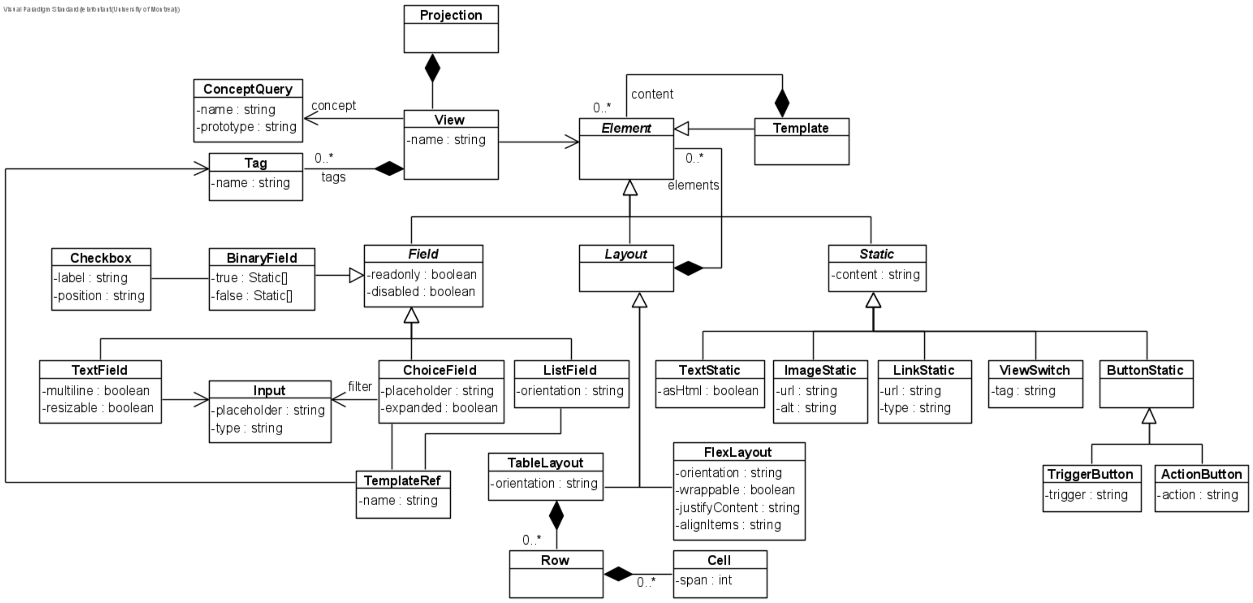
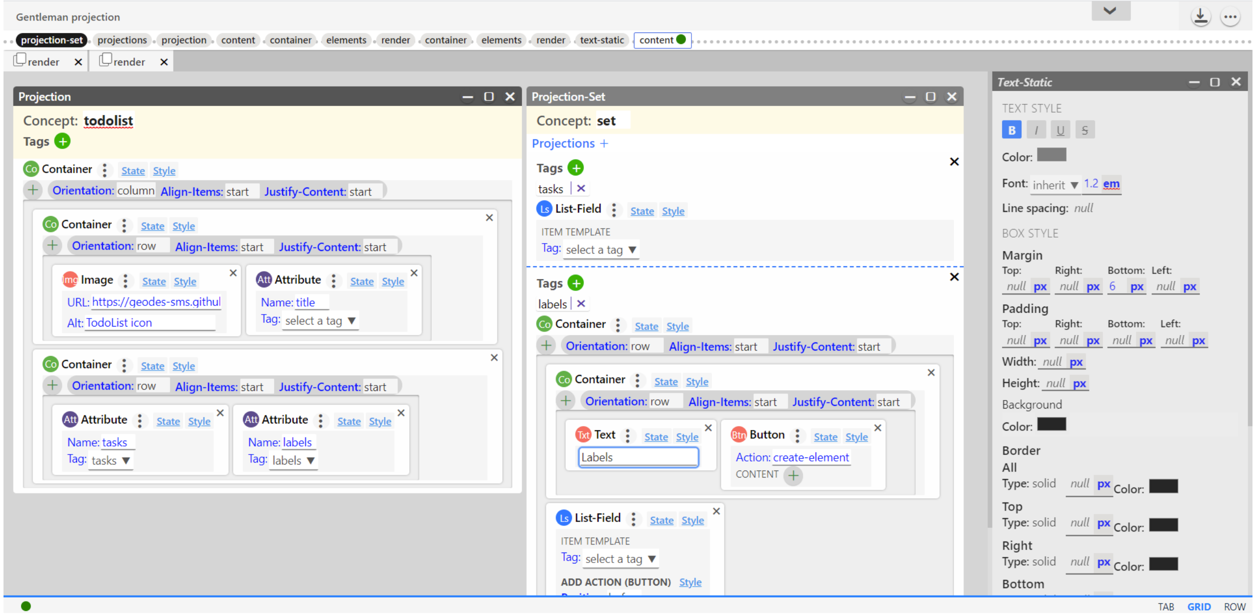


Fig. 5.1. Gentleman projection metamodel



Projection

Concept: todolist

Tags

Container State Style

Orientation: column Align-Items: start Justify-Content: start

Container State Style

Orientation: row Align-Items: start Justify-Content: start

Image State Style

URL: https://geodes-sms.githul

Alt: TodoList icon

Attribute State Style

Name: title

Tag: select a tag ▼

Container State Style

Orientation: row Align-Items: start Justify-Content: start

Attribute State Style

Name: tasks

Tag: tasks ▼

Attribute State Style

Name: labels

Tag: labels ▼

Projection-Set

Concept: set

Projections +

Tags

tasks | ×

List-Field State Style

ITEM TEMPLATE

Tag: select a tag ▼

Tags

labels | ×

Container State Style

Orientation: row Align-Items: start Justify-Content: start

Container State Style

Orientation: row Align-Items: start Justify-Content: start

Text State Style

Labels

Button State Style

Action: create-element

CONTENT

List-Field State Style

ITEM TEMPLATE

Tag: select a tag ▼

ADD ACTION (BUTTON) Style

Text-Static

TEXT STYLE

B I U S

Color:

Font: inherit ▼ 1.2 em

Line spacing: null

BOX STYLE

Margin

Top: null px Right: null px Bottom: 6 px Left: null px

Padding

Top: null px Right: null px Bottom: null px Left: null px

Width: null px

Height: null px

Background

Color:

Border

All

Type: solid null px Color:

Top

Type: solid null px Color:

Right

Type: solid null px Color:

Bottom

Fig. 5.2. TodoList projection in Gentleman

many smaller elements, put together to interact with a concept and navigate its structure, as seen in the metamodel in Fig. 5.1 . The elements that compose a projection are classified as *layouts*, *interaction points*, *static elements*, and *relation functions*. They shape the body of a projection, which may be customized with *styling* rules, added as the final layer.

Binding. In Gentleman, projections are loosely configured and use a late-binding approach to maximize their modularity and flexibility. Although there is a definite correspondence between a projection and a concept at runtime, this binding is not strictly specified in its definition. A projection declares an assertion that specifies which concepts may be used with it, using properties such as name and prototype. The editor uses that assertion when looking for a projection compatible with a given concept. This allows a projection to target a prototype concept and thus every concrete concept that implements that prototype.

Tagging. This dynamic behavior and loose coupling is also observed for projection composition. A projection follows the structure of a concept, which may have relations with other concepts, having their own projections. The name or one of the tags may be used to target a projection, similar to CSS class and id selectors, to select which projection will be displayed. As the name is unique, if used, the projection will have a single or no solution. However, using a tag will return a collection of projections as many projections may define the same tag.

5.2. Layout

A layout is essential to any graphical representation as it organizes the projection elements presented in the GUI. It indicates the initial location of its child elements and creates a flow of constraints that they follow. In Gentleman, to define the scope governed by a layout, it is associated with a container that can be configured with options, such as *movable* and *resizable* which allows a container to be repositioned in a parent container and change the rendered box size, respectively. Gentleman offers predefined *layouts* found in modern GUI technologies [**84**] and frameworks such as Xamarin [**36**] and SWT [**34**].

5.2.1. Flex Layout

The FlexLayout leverages the Flexbox layout defined in CSS [**3**]. The direct children of the associated container children elements can be laid out on one axis (either horizontal or vertical), and these child elements can automatically grow and shrink to fill the space available without overflowing the parent container [**33**]. This makes it simple to position content while retaining the flexibility to introduce additional sibling elements later.
Example. In the TodoList example, the FlexLayout is used extensively and exclusively to organize elements vertically and horizontally. As a layout may contain any element, including another layout, a Todo is structured vertically first, with a header where we find the title and a body, which shift the flow horizontally. The body is split into two vertical flex-layout to organize the tasks and the tags.

5.2.2. Table Layout

While powerful, the FlexLayout deals with only one dimension. However, some entries, such as tabular data, require a second dimension to coordinate the elements. In Gentleman, this is made possible with TableLayout, which arranges its elements in a table that can either be row-directed or column-directed and is divided into three sections: an optional *Header* and *Footer* presented at the top and bottom of the table respectively, and a *Body* that may itself be divided into subsections. They all contain cells organized in rows or columns. A cell may span across multiple rows or columns and may contain child elements.

5.3. Interaction points

An interaction point, called *Field* in Gentleman, is concerned with manipulating the value of a concept; thus, it enables data input and output. It offers data-specific controls, which provide more structure to help users quickly scan and comprehend the information presented [**43**]. It provides an abstraction for the underlying widget (control element) to promote reusability and portability. It uses a modular approach to select the right widget for the intent of the user. Gentleman offers fields that cover the most fundamental widget components in GUIs [**32**]. Fields also have generic properties to specify, for example, if they are read-only, disabled, or hidden.

5.3.1. TextField

As our interaction with computers is rooted in typing, even with the popular use of GUI, textual input is still an important interaction [**62**]. It then follows that in order to accept models that require textual inputs and in alignment with the web environment, Gentleman offers the textfield. It allows the user to input characters and is represented as single-line textbox or multi-line textbox depending on the requirements (configuration). In order to assist the user with the entry, a textfield may provide some assistance in the form of autocompletion [**72**]. Naturally, it is used to interact with a *String* or *Number* concept, both capable of parsing textual input.

Example. Let us consider the *Priority* concept, which is limited to four values and defined as a Derivative concept in Gentleman. As it is based on a String, a textfield is used to interact with the concept, as seen on the right of Fig. 3.2. During the interaction with the field, the user may query the field for the list of accepted values. Similarly, we see that the reference attribute which targets a Number is also interacted with using a textfield.

5.3.2. BinaryField

Just as essential as the textbox is the checkbox according to [**91**]. However, as a generalization of the interaction, we found that it may be presented as an element that alternates between two states. Thus in Gentleman, we have the BinaryField filling that role. As seen in Fig. 5.1, each state (true, false) may define its own representation. Generalizing the behavior like so frees the designer from the boundary of the mark to indicate a change in state. Naturally, it used to interact with a *Boolean* concept, but may also be found useful against a derivative presenting two values such as the result of a coin-flip.

Example. In our Todo example, the *completed* status of a task is a natural fit for the binary field, presented as a customized checkbox in this case.

5.3.3. ChoiceField

It is not always necessary or desirable for the user to enter any value in our model, but a binary choice might be too limited to fully express the list of possible values. For this reason, we introduce the ChoiceField, which allows the user to select one item in a predefined list. Similar to the popular PHP framework *Symfony* [1], depending on the configuration of the ChoiceField, it might resolve to a radio-button list or a combo-box. Such a field is very practical when interacting with a Prototype concept where the user needs to choose the implementation to use or a Reference concept that accepts specific candidates.

[1] `https://symfony.com/doc/current/reference/forms/types/choice.html`

Example. In Fig. 3.2, we have an example of both. At the top of every task, the concrete implementations of the prototype concept Task are listed as radio buttons. In the task options, at the right, a combo-box is presented to make a reference to an existing tag.

5.3.4. ListField

An interaction may deal with a single element as described in the previous fields, but may also deal with collection of elements. In Gentleman, such an interaction is made possible with ListField which allows the user to manipulate a collection of values. It gives the user the means to add and remove items in the list and may be provided with templates for each of its items. It is dedicated to interacting with a *Set* concept.
Example. In the todo example, we have two instances of the ListField to manage our set of tasks and labels. Though they use the same field, we can note that the layout and presentation of the buttons as well as the elements are quite different. Indeed, ListField manages the interaction with the collection while allowing its content and controls to be freely customized.

5.4. Static element

Programs and models interactions are not purely restricted to value input and action trigger, as to do those well, the context needs to be understood. Moreover, a model may use in an activity that does not require it to be modified but queried for information. As noted by [**101**], static elements, such as symbols and keywords, have a significant impact on program comprehension. Gentleman presents several types of Static elements to ease the process of program comprehension and enable rich projection.

5.4.1. Text

Textual content as decoration can play a very crucial role in understanding a model. This is demonstrated with graphic visualization, which without text can be impossible to decipher [**98**] or with forms where labels provide great insight to fill out the fields [**40**]. In Gentleman, *StaticText* allows you to display text, which may be printed as HTML for richer content.
Example. In our TodoList example, the titles and labels such as “Tags ”and “Priority ”, respectively, are provided as StaticText. This helps the user quickly understand the type of content of a section and fill in the fields with assurance.

5.4.2. Image

An image may convey information a lot quicker than text. As a decorative element, we can distinguish between icons and graphics, which are symbolic, and pictures, which create

a theme in the surrounding area. In Gentleman, *StaticImage* allows you to display any web-supported image, which may even be animated for richer content.
Example. In the TodoList example, there are several instances of StaticImages. At the top of a TodoList, we have an icon that is symbolic of a list, which sets the tone for what is coming next. The types of task to choose from have also been rendered with icons which creates a cleaner interface and reduce the cognitive required to parse the type.

5.4.3. Link

With the previous two, we can already create rich projections. However, as the editor runs on the web, we may want to create a hyperlink to a guide, documentation or any resource that could help in the creation of a model. In Gentleman, *StaticLink* allows the user to create hyperlinks to external resources.

5.4.4. ViewSwitch

A link as described above allows us to navigate to external resources, but there might be a need to navigate inside a projection. This scenario is central to multi-projections, where multiple projections are used to manipulate a concept. In order to navigate amongst the different views, Gentleman introduces *ViewSwitch*. It may target any alternate view currently found in the projection by using the tag or name to find the view.
Example. This is very useful when dealing with complex concepts. In this case, we split the attributes into separate views, which may be accessible via a tabular structure, where a tab is a ViewSwitch that switches the projection to a view. A more simple example is to provide a collapsed view where a ViewSwitch could be used to switch the projection for a much simpler one, presenting fewer elements.

5.4.5. Button

In order to encapsulate an action and give the user the ability to trigger it, a button is usually required. In Gentleman, a button is called *StaticButton* and rendered as such in the projection. A StaticButton may be used to broadcast a message on the editor, which we will trigger the subscribe handlers or invoke an operation on the underlying concept.
Example. In the TodoList example, a StaticButton is displayed next to the label title, to create a new label. When clicked, it sends a request to the Set concept used to manage the labels. The Set creates the label and notifies the registered ListField, which displays the newly added label. Another instance is found in the header of a task, where a three-dot button is used to access the task options. In this, case a message is sent to the editor, which triggers a handler that opens a window on the right to configure the options.

5.5. Relation function

One of the key features of a projectional editor is language composition as described in Chapter 2. In Gentleman, in order to connect a projection to another projection, thus making projection composition, we use relation functions. A relation function is rendered as a dynamic element that targets a relation of the underlying concept. Projections are composed together following the concept relations. As such, an attribute may be projected to connect a parent projection to another compatible with the attribute target concept. The target projection is selected based on the parameters, such as the projection tag given to the function.

Example. In our running example, the TodoList projection is composed of a Textfield for the title and a ListField for our tags. This conforms to the relations of a TodoList concept defined in the metamodel. Returning to our example, we can note different types of representation of Textfields. This showcases the selective properties of a relation function, targetting the proper projection with the underlying concept.

5.6. Template

Templates favor reuse following the principle of write once, produce many [**106**], such that a segment may be reused in various contexts. The degree of complexity of a template lies in the implementation of its dynamic part [**61**], where a template might be a block of content reuse in different places, such as the footer of a webpage, or it might define a generic behavior that will change form depending on the context. In Gentleman, a template may be defined and referred to by a projection.

Example. In the TodoList example, we see that, regardless of the task, we have a common part that contains the name, describe and status (checkbox). For this, the Single task and Recurring task use a template where those elements are defined.

5.7. Styling

Any projection can be complemented with style rules to describe its presentation. Styles can be defined directly in the Gentleman editor or imported. Style can be applied to any element of a projection. It follows the box model defined in CSS. For example, users can set the font, color, and alignment of text and the border of a table. The former can either target the text content or the container by defining a *TextStyle* or a *BoxStyle* respectively. To avoid repetition and encourage better integration, Gentleman leverages browser technologies, offering full support for CSS class selectors. Like so, changing the appearance of a projection could be achieved by changing the stylesheet loaded on the page. Following best practices of UI library such as Bootstrap [**99**] or Materialize [**80**], every element exposes a set of standard classes representing the type and state of the element. As an example, every field has a class named `field`, which for a textfield is complemented by `field-textbox`. When no value is held by the textfield, the `empty` class is assigned to it. Similarly, the class `error` is used to indicate the presence of errors. This open approach enables the designer to declare global styles through CSS and specific context-based rules in Gentleman.

Text Style. A TextStyle allows the designer to describe and format the presentation of the text. The available properties are: Bold, Italic, Underline, Strikethough, Color, Font (font family), Transform.

Box Style. A BoxStyle concerns itself with the container. The available properties are: Inner space (padding), Outer space (margin), Width, Height, Border, Background, Box shadow.

Chapter 6

Evaluation

We have evaluated Gentleman with a user study to show its effectiveness and usefulness. This study aims to show the effectiveness of modeling with a projectional editor using a structured approach, as presented in Chapter 5. To showcase Gentleman interoperability, it has been successfully integrated into a complex software developed with MDE technologies.

6.1. User study

In order to study the effectiveness of Gentleman, it will be compared to an existing projectional editor. As discussed in Chapter 2, MPS is a valid candidate and representative of the state of the art of projectional editors. MPS uses a cell-based approach to arrange its projections, where each element (presented on the UI) occupies a cell. The cells are stacked vertically and horizontally, resulting in a flat layout. Although MPS supports multiple notations, it most frequently textual, and thus the interaction is similar to a textual editor as described in Section 2.3.1. Gentleman, as described in Chapter 5, uses a container-based approach to arrange the projections and widgets to interact with them. As containers can be boxed, the layout may present multiple levels. The syntax found in a typical Gentleman editor is very diverse, as we saw in the TodoList example Fig. 3.2, which is composed of static content such as text and icons, buttons and widgets such as textbox and checkbox.

6.2. Objectives

The goal of this study is to evaluate the effectiveness of Gentleman, with respect to its efficiency and usability, from the point of view of domain experts in the context of graduate students and industry practitioners, performing modeling activities.

To do so, we formulate the following research questions.

RQ1: *"Does the approach used by Gentleman improve the user's productivity compared to existing projectional editors?"* In this question, we observe and compare the user's efficiency and the number of errors made during the activity with Gentleman and MPS.

RQ2: *"Does the approach used by Gentleman improve the ability to understand a model compared to existing projectional editors?"* In this question, we observe and compare how accurately the participant assesses the model's state and how easy it is to change it with Gentleman and MPS.

6.3. Study design

To answer the research questions stated above, we conducted a controlled experiment.

6.3.1. Setup

To perform this experiment, we used a Windows 10 machine (Dell XPS 13 9380, 8GB RAM) connected to a wide monitor (24 inches), with the developed software, as well as the support software such as the *Chrome* browser and PDF reader. In order to facilitate and accommodate the participants, we performed the experiment remotely using the software *Anydesk* to connect the participant to the machine. Every instance of the experiment was conducted by me, with one participant for each session. In this experiment, I acted as the supervisor. I introduced the experiment, explained each task, and signaled the beginning and end of a task. Although the participant had access to the web for documentation, I provided technical support during the task as a knowledgeable user of both tools. As this was a remote experiment, our interaction was done through the videoconferencing platform *Zoom*, thus enabling audio and video feedback.

6.3.2. Participant

We sent a call for volunteers to enroll participants for this study, emphasizing that no prior knowledge of either tool was required. As the tasks were about concrete modeling and not metamodeling, knowledge of MDE was also not a prerequisite. We followed a convenience sampling by sending invitations to research groups and industry practitioners of our knowledge. We had 22 volunteers responded to the call and participated in this study, with 12 from academia (researcher and graduate students) and 10 professionals from various industries, including software engineering. Appendix C.1 presents the profile of the participants. Most of them had a graduate degree, with varying years of expertise in computer science ranging from 3 to 33. Two participants had no experience with computer science, which added some diversity to the group. They had varying expertise of MDE, ranging from 1 to 18, with the majority considered as a novice in MDE. Most of the participants had almost no experience with projectional editing, with only six with more than two years of experience (with MPS). A month before the experiment, we performed a run-test with a volunteer to get some feedback regarding the experiment process and the setup. We did not consider the data from his session in the following analysis. Many coming from a programming or software engineering background are used to a type of editor. To reduce any form of cognitive bias towards the editing experience, we chose participants from various backgrounds. To remove any bias or familiarity effect, we divided the participants into two groups and had them do the experience using both tools in a different order.

Group A: First completed all tasks with Gentleman and then completed the same tasks with MPS.

Group B: First completed all tasks with MPS and then completed the same tasks with Gentleman.

6.3.3. Experiment process

The user study was conducted in individual sessions, divided into three parts: the introduction, the experiment, and the feedback. In the introduction, we presented the experiment, went through the tool's documentation, and provided a hands-on tutorial that lasted about 5 minutes. The tutorial helped them familiarized themselves with the UI and the editor main interactions. As it was a modeling activity, the projections and concepts used in the editor were prepared beforehand. The experiment was divided into a sequence of three distinct modeling tasks. The first two were concerned with creating a model, and the third focused on evaluating an existing model. After completing the three tasks with the first tool, we repeated the same procedure with the other tool, starting with the documentation and the tutorial. For the tasks, the participants used the metamodel of a traffic light (TL), presented in Fig. 6.1. As most participants were discovering new tools, we did not impose any time constraints. However, as an indication of the difficulty level, we communicated an estimate of the time required to accomplish the task. This time was based on the time taken by someone familiar with the tool but not with the projections (syntax). After completing the tasks, we asked them to complete a survey. For most participants, the experiment lasted about an hour.

Traffic light metamodel. A traffic light is modeled as a state-machine system. A *State* is identified by a *name* and is related to other states through *behaviors*, which can either be temporal or manual. A *Temporal* behaviour indicates the time to wait by specifing a *value* and a *unit*, limited to s, ms and min. A *Manual* behaviour indicates the *event* that will trigger the transition. As a traffic light system, the state will either be a light or a mode. A *Light* is a state identified by a *color*, limited to 5 colors. A *Mode* is a state composed of at least one light and indicates a *start* light.

Traffic light editor. Similar to the tutorial, the projections, and concepts used in the editor, were prepared beforehand. Fig. 6.2 and Fig. 6.3 show the editor of a participant modeling in Gentleman and MPS, respectively. The projections were created to be as user-friendly as

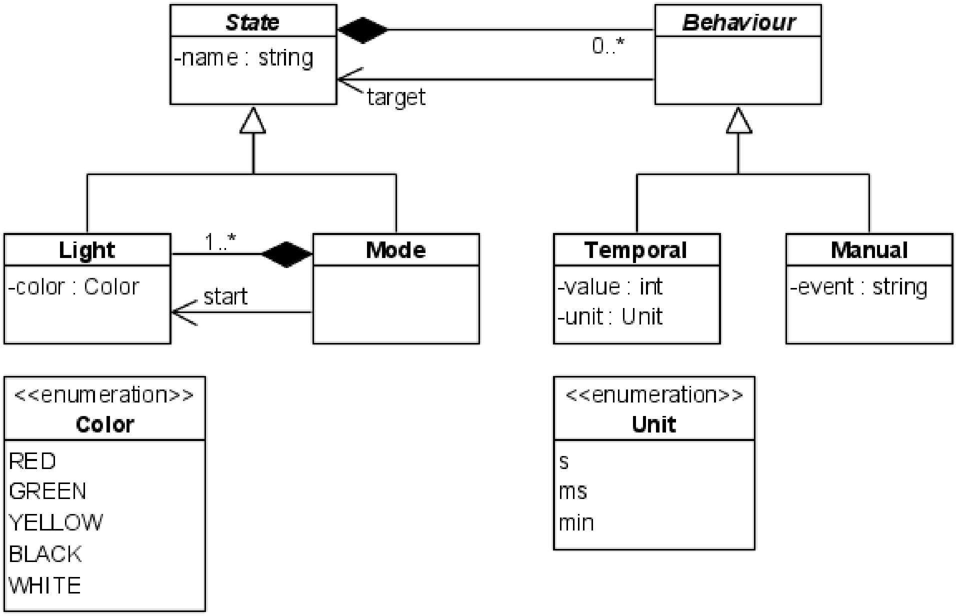


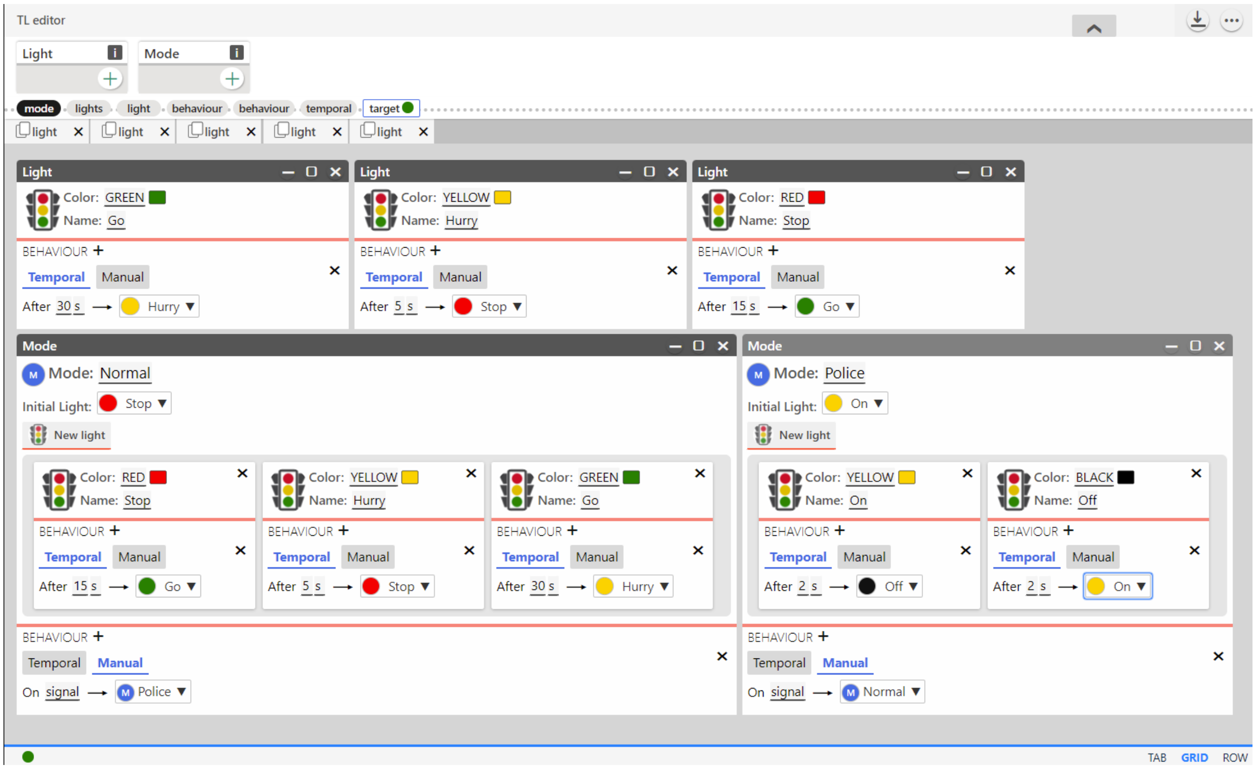
Fig. 6.1. Traffic Light metamodel

LIGHT		TRANSITION		
Name	**Color**	**Type**	**Value**	**Target**
Stop	Red	Temporal	15s	Go
Slow	Yellow	Temporal	5s	Stop
Go	Green	Temporal	30s	Slow

Table 6.1. Requirements for the TL model of task 1

possible to showcase the best features of each editor. Gentleman's editor presents a mix of notations. Each light and mode has an icon next to it to communicate the type of content. As the participant types the color, it is displayed next to it, helping them confirm the choice. The type of behaviors are exposed so that the user does not have to guess and make it easy to switch behavior. MPS' editor presents a rich textual notation. The keywords are colored to distinguish them from the entered value. Similar to Gentleman, the color typed is displayed next to the value.

Task 1. As this was their first use of the tool, after the tutorial, the first task presented a reduced version of the metamodel, as shown in Fig. 6.4. The focus here was less about the complexity of the model and more about the difficulty found in the tool to create a simple model, similar to the one used for the mind map tutorial. Therefore, at this stage, the participant was only asked to create lights and behaviors. The requirements for this task were to create three lights and assign to each light a temporal transition that would link it to another light, as presented in Table 6.1.



Light — Color: GREEN 🟩 Name: Go
BEHAVIOUR +
Temporal Manual
After 30 s ⟶ ● Hurry ▼

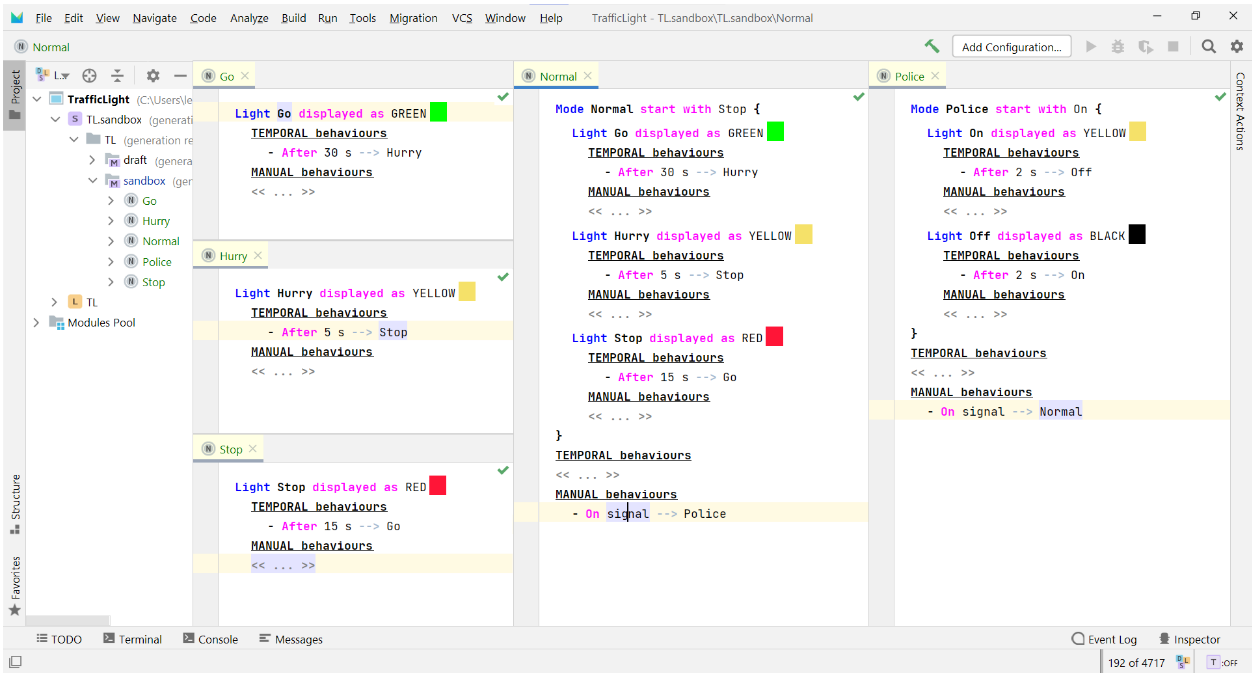
Light — Color: YELLOW 🟨 Name: Hurry
BEHAVIOUR +
Temporal Manual
After 5 s ⟶ ● Stop ▼

Light — Color: RED 🟥 Name: Stop
BEHAVIOUR +
Temporal Manual
After 15 s ⟶ ● Go ▼

Mode — Mode: Normal
Initial Light: ● Stop ▼
New light
Color: RED 🟥 Name: Stop — BEHAVIOUR + Temporal Manual — After 15 s ⟶ ● Go ▼
Color: YELLOW 🟨 Name: Hurry — BEHAVIOUR + Temporal Manual — After 5 s ⟶ ● Stop ▼
Color: GREEN 🟩 Name: Go — BEHAVIOUR + Temporal Manual — After 30 s ⟶ ● Hurry ▼
BEHAVIOUR +
Temporal Manual
On signal ⟶ M Police ▼

Mode — Mode: Police
Initial Light: ● On ▼
New light
Color: YELLOW 🟨 Name: On — BEHAVIOUR + Temporal Manual — After 2 s ⟶ ● Off ▼
Color: BLACK ■ Name: Off — BEHAVIOUR + Temporal Manual — After 2 s ⟶ ● On ▼
BEHAVIOUR +
Temporal Manual
On signal ⟶ M Normal ▼

Fig. 6.2. Gentleman Traffic Light editor



Go
```
Light Go displayed as GREEN
  TEMPORAL behaviours
    - After 30 s --> Hurry
  MANUAL behaviours
  << ... >>
```

Hurry
```
Light Hurry displayed as YELLOW
  TEMPORAL behaviours
    - After 5 s --> Stop
  MANUAL behaviours
  << ... >>
```

Stop
```
Light Stop displayed as RED
  TEMPORAL behaviours
    - After 15 s --> Go
  MANUAL behaviours
  << ... >>
```

Normal
```
Mode Normal start with Stop {
  Light Go displayed as GREEN
    TEMPORAL behaviours
      - After 30 s --> Hurry
    MANUAL behaviours
    << ... >>

  Light Hurry displayed as YELLOW
    TEMPORAL behaviours
      - After 5 s --> Stop
    MANUAL behaviours
    << ... >>

  Light Stop displayed as RED
    TEMPORAL behaviours
      - After 15 s --> Go
    MANUAL behaviours
    << ... >>
}
TEMPORAL behaviours
<< ... >>
MANUAL behaviours
  - On signal --> Police
```

Police
```
Mode Police start with On {
  Light On displayed as YELLOW
    TEMPORAL behaviours
      - After 2 s --> Off
    MANUAL behaviours
    << ... >>

  Light Off displayed as BLACK
    TEMPORAL behaviours
      - After 2 s --> On
    MANUAL behaviours
    << ... >>
}
TEMPORAL behaviours
<< ... >>
MANUAL behaviours
  - On signal --> Normal
```

Fig. 6.3. MPS Traffic Light editor

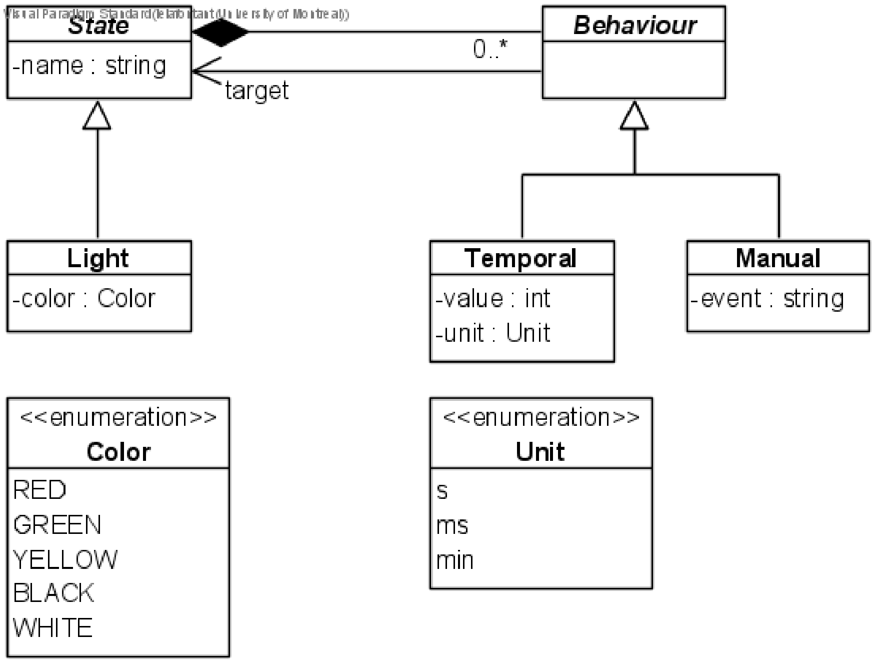


Fig. 6.4. Traffic Light initial metamodel: task 1

MODE			TRANSITION		
Name	**Lights**	**Initial light**	**Type**	**Value**	**Target**
Normal	See table 6.1	Stop	Manual	signal	Police
Police	See table below	On	Manual	signal	Normal

LIGHT		TRANSITION		
Name	**Color**	**Type**	**Value**	**Target**
On	Yellow	Temporal	2s	Off
Off	Black	Temporal	2s	On

Table 6.2. Requirements for the TL model of task 2

Task 2. With the experience from task 1 and the tutorial, we assumed that the participant gained enough familiarity with the tool and could tackle a more complex model. For the second task, we revealed to the participant the complete metamodel, as shown in Fig. 6.1. The difference with the metamodel of task 1 is the addition of the mode concept. To evaluate his performance in a refactoring operation, the participant was asked to modify the existing model. They had to change the model to conform to the new requirements, presented in Table 6.2. They had to create two modes, named *normal* and *police*. The normal mode reused the three lights created in task 1, and the police mode defined two additional lights. The two modes had a manual behavior to transition to the other.

Task 3. At this stage, with tasks 1 and 2 completed, we assumed that the participant gained enough familiarity with the syntax to evaluate an existing model using the same syntax. To avoid a bias towards the evaluated model, which might make the task too easy or too hard, we

MODE			**TRANSITION**		
Name	**Lights**	**Initial light**	**Type**	**Value**	**Target**
Automatic	See table below	GL	*Any*	*Any*	*Any*

LIGHT		**TRANSITION**		
Name	**Color**	**Type**	**Value**	**Target**
GL	Green	*Any*	*Any*	YL
YL	Yellow	*Any*	*Any*	RL
RL	Red	*Any*	*Any*	GL

Table 6.3. Requirements for the TL model of task 3

created ten different models and randomly chose one for each participant. Therefore, in task 3, they were presented with a model containing several semantic errors. The requirements used to validate the model is presented in Table 6.3. In this scenario, no constraint was defined on the type of transition, indicated by the word *Any* in the table. They were asked to identify the errors and explain the kind of the error. This activity was done orally, using the mouse to point the error in the model. After the participant confirmed that he was done reviewing the model, we asked them to apply the changes to obtain a valid model.

6.3.4. Feedback survey

At the end of the experiment, the participant was invited to complete an online survey, to gain further insight into their experience. The goal of this survey was to identify the participant background and collect subjective evaluations of our approach and gain feedback about the proposed solution. The survey consisted of 10 identification questions to create their profile, focusing on their experience with modeling and with projectional editors. Following that were 10 questions regarding their experience with Gentleman and the same questions again with MPS. Those questions were formulated as statements and nswered using a five-point Likert scale with 1 indicating strong disagreement with the statement and 5 indicating complete agreement. The survey was taken in the presence of the supervisor to clarify the questions. Appendix C.2 presents the complete survey.

6.4. Metrics

In order to measure quantitatively the effectiveness of our approach, we identified several metrics.

6.4.1. Independant variables

The independent variables are mostly the participant background identified in the survey. In addition to those, we added the task in itself as well as the tool.

6.4.2. Dependant variables

In order to gather cohesive metrics, we created four distinct categories: mechanical effort, cognitive effort, completion, and error.

Mechanical effort. The mechanical effort variables targets the perceived actions that are the direct result of a mechanical movement such as the movement of the mouse or a key being pressed.

`NB_Selection`: Number of selection. A selection is either a mouse click or the end of a pressed navigation key.

`NB_Blank`: Number of blank selection. It occurs when the selection does not change the focus or the state of model, such as clicking on an already focused textbox.

`NB_Miss`: Number of miss selection. It occurs when the wrong selection is made, such as clicking on the wrong button.

`NB_CtxtChange`: Number of tab or window change.

Cognitive effort. The cognitive effort variables target all the activity during a task that is not captured by the tool, such as idle time or taking a pause to ask the supervisor some questions.

`T_Idle`: Idle time. It begins after 2 seconds of inactivity, such that no mechanical effort is produced, and the editor is not executing an operation.

`T_QA`: Question and answer (QA) time. It begins after 3 seconds of interaction with the supervisor. This is to dismiss validating questions.

`NB_Idle`: Number of idle occurences.

`NB_QA`: Number of QA occurences.

Completion. The completion variables target the total time taken to accomplish a task and consider the help given by the supervisor that was crucial to the completion of the task.

`T_Total`: Total time taken to complete the task.

`NB_Block`: Number of occurences when the user was blocked in his task, and requested some assistance.

Error. The error variables target the mistakes of the participant, which result in a non-valid state, and his ability to notice the error and correct it.

`NB_Typos`: Number of typos. These include wrong inputs such as “YELO” instead of “YELLOW” or “red” instead of “RED”.

`NB_DesignErr`: Number of design errors. They occur when the requirements are not respected, such as selecting the wrong type of behaviors.

`RT_Detection`: Detection rate. The ratio of the total number of errors and the number of errors found.

`RT_Recover`: Recovery rate. The ratio of the total number of errors and the number of errors corrected.

`T_Recover`: Total time taken to recover from all the errors following their detection.

6.4.3. Survey variables

The variables gathered from the survey are listed in Table 6.6. They are presented in the same order as the question to which they are attached, presented in Appendix C.2.

6.4.4. Null hypothesis formulation

With our variables identified, we can formulate the null hypothesis that will help us answer our research questions.

H_{0,Mec_Effort}: The tool does not impact the mechanical effort required to accomplish a task with a projectional editor. Here we consider the following variables: `NB_Selection`, `NB_Blank`, `NB_Miss`, `NB_CtxtChange`.

H_{0,Cog_Effort}: The tool does not impact the cognitive effort required to accomplish a task with a projectional editor. Here we consider the following variables: `T_Idle`, `T_QA`.

H_{0,Com_Time}: The tool does not impact the time taken to accomplish a task with a projectional editor. Here we consider the following variables: `T_Total`.

H_{0,Err_Number}: The tool does not impact the number of errors produced during a task with a projectional editor. Here we consider the following variables: `NB_Typos`, `NB_DesignErr`.

$H_{0,Err\_Recover}$: The tool does not impact the ability to recover from errors with a projectional editor. Here we consider the following variables: `RT_Detection`, `RT_Recover`, `T_Recover`.

$H_{0,Usability}$: The tool does not impact the user experience of projectional editors with a projectional editor. Here we consider the survey variables.

6.4.5. Data collection

After the confirmation of consent from the participant at the beginning of the experiment, the session was recorded using *ScreenCast* [1], which captured the screen content as a video. This included the interaction with the PDF reader and other manipulations such as opening a browser window. During the activities, no notes were taken. Instead, we recorded the audio using *Zoom* [2], to provide better insight as to the intent behind an interaction or lack of interaction. Lastly, although Screencast distinguishes the clicking effect of a mouse, we used a dedicated tool, Teramind [3], to monitor mouse and keyboard interactions. To

[1] `https://screencast-o-matic.com/`

[2] `https://zoom.us/`

[3] `https://www.teramind.co/`

protect the privacy of the participants, all the collected data was anonymized. After every experiment was over, we analyzed the videos and collected the relevant information from reports generated by the software, and then we destroyed the recorded data as per the agreement mentionned to the participant. At the end of the session, we collected the answers to the survey to contextualize the directly collected data. We statistically analyzed the collected data for each metric using the software *SPSS 25* [4].

6.5. Results

Before we begin our analysis, it is important to consider the effect of the order of the tool on the results. This preliminary analysis was done to compare means and variance between each group, using a confidence interval of 95%. It was concluded that the order of the tool had no significant impact on the survey results nor the performance metrics, listed in 6.4. Appendix C presents a complete report of the tasks and the survey.

6.5.1. Descriptive statistics

To begin our analysis, we will first consider measures of center, such as the mean, median, and mode. In the tables presented in the Appendix, for the performance metrics safe of `RT_Detection` and `RT_Recover`, a lower number is considered better, as we are aiming for efficiency. For the survey, the opposite is observed as it reflects the degree of appreciation.
Task 1. For task 1, considering the mean, participants performed better using Gentleman than MPS, but with various degrees. The most important differences are found in the mechanical effort and completion time. On average, a participant produced 10 more `NB_Selection` with MPS than with Gentleman and 3 more `NB_Blank`. However, the difference in terms of (`NB_Miss`) is only 0.46. For `NB_CtxChange`, on average, a participant produced 8 occurrences with MPS and none with Gentleman. In terms of time, on average, a participant took 90 more seconds to complete the task with MPS than with Gentleman. Looking at the minimum, which reveals the best performance with each editor, they produced (for most variables) the same result. However, the maximum is much higher on MPS case for most variables. In terms of variability, participants produced a much larger distribution with MPS than with Gentleman, especially for the mechanical effort variables. Lastly, we note two instances where a participant needed critical help (`NB_Block`) for MPS and none for Gentleman.
Task 2. For task 2, except for `NB_Miss`, every other variable indicated a better performance with Gentleman than MPS. However, in terms of mechanical effort, the difference between the two has reduced, with 6 more selections with MPS, compared to the previous 10 and slightly more miss (`NB_Miss`) with Gentleman. The difference for `NB_CtxChange` remained

[4] `https://www.ibm.com/docs/en/spss-statistics/25.0.0`

just as important as task 1, but the difference in total time dropped to 60 seconds, still in favor of Gentleman. For this task, we note some differences for the minimum. It is lower in the case for Gentleman for the `NB_Selection` and `T_Total`, indicating that the best user performed better with Gentleman. Similar to task 1, the maximum is higher for MPS. Lastly, we note two instances where again a participant needed critical help for MPS and none for Gentleman.

Task 3. For task 3, there is very little difference between MPS and Gentleman, with most variables giving a slight edge to Gentleman. During this task, a participant triggered an internal error (`NB_Bug`) in Gentleman and none in MPS.

Survey. In the survey, for Gentleman, like the results of the tasks, most participants strongly agreed with the statements. The weakest point in the participant experience with Gentleman is the navigation with the keyboard (`NavKey`). However, it should be noted that only 10 participants were considered as the rest navigated using only the mouse. All the other variables indicate, on average, a score greater than 4.5 with a minimum between 3 and 4. For MPS, the average score was a point lower, indicating simple agreement with the statements. The result distribution, in this case, presents a lot of variability towards a lower score than its median of 4. The minimum for MPS mainly oscillated between 2 and 3, with a few participants strongly disagreeing with some statements, such as *It is easy to remember the controls and commands of the editor* and *It is easy to know what actions are expected from me or available to me at any point int time.*

6.5.2. Statistical signifiance

To see whether those differences are significant, we conducted Levene's test to assess between-group variance differences and a one-way between-subjects ANOVAs test with posthoc tests to assess between-group mean differences using the tool as the independent variable. Table 6.4, Table 6.5 and Table 6.6 presents the ANOVA for tasks 1, 2, and the survey, respectively. The significant variables annotated by a red star are selected at 95% confidence level (p-value lower than 0.05). Having a small group of participants, we tested the effect size with an Eta-squared test. As we can see in Appendix C.3, all of the significant variables from the ANOVA test indicate an ETA squared greater than 0.14. Therefore our sample size is large enough to consider those variables.

Task 1. For task 1, there is a significant difference in terms of mechanical effort, as indicated with the variables `NB_Selection` and `NB_BLANK`, validating the difference noted in our descriptive analysis. We attribute this difference mainly to the usability of the editors, as MPS requires more interaction to discover what is possible. To create an instance, the user has to right-click on the project folder and select it from the contextual menu, whereas they

Variable		Mean	Std. Deviation	F	P-value
`NB_Selection` *	Gentleman	27.73	5.15	(1, 42) = 8.08	**0.007**
	MPS	37.55	15.36		
`NB_BLANK` *	Gentleman	1.32	1.55	(1, 42) = 9.63	**0.003**
	MPS	4.59	4.70		
`NB_MISS`	Gentleman	0.27	0.70	(1, 42) = 3.57	0.066
	MPS	0.73	0.88		
`NB_CxtChange` *	Gentleman	0.00	0.00	(1, 42) = 130.79	**< 0.001**
	MPS	8.36	3.43		
`NB_IDLE`	Gentleman	1.59	1.68	(1, 42) = 1.41	0.241
	MPS	2.27	2.10		
`T_IDLE`	Gentleman	5.36	7.61	(1, 42) = 3.81	**0.058**
	MPS	12.32	14.88		
`NB_QA`	Gentleman	0.27	0.55	(1, 42) = 4.54	**0.039**
	MPS	1.32	2.23		
`T_QA`	Gentleman	1.50	3.47	(1, 42) = 3.50	0.068
	MPS	12.27	26.78		
`T_TOTAL` *	Gentleman	138.18	57.21	(1, 42) = 9.308	**0.004**
	MPS	227.05	124.07		
`NB_BLOCK` *	Gentleman	0.00	0.00	(1, 42) = 10.57	**0.002**
	MPS	0.41	0.59		

Table 6.4. Task 1 ANOVA results

are readily available in the header for Gentleman, as seen in Fig. 6.2. With MPS, the participant also has to query the editor to know what is expected at a certain position in the syntax, useful when making a reference, whereas the choice field list all the possible options that the user can choose from. Finally, as they were discovering the tools, they lacked any form of trained reflex to react automatically to the three dots left as a placeholder in MPS, which caused some participants to request some assistance to continue, indicated by `NB_BLOCK`. The highest difference is in the number of context changes (`NB_CtxChange`). Gentleman offering the grid layout, the participant never had to open another window (context) to access additional information as they had the whole model presented in one. In MPS, however, each root instance is opened in a separate window, thereby creating multiple contexts. Additionally, creating a reference between the instance usually required the participant to navigate through the windows to validate their choice. Lastly, we note that the total time (`T_TOTAL`) is significantly lower in Gentleman than in MPS, which follows the explanation given above. **Task 2**. For task 2, there is still a slight edge in favor of Gentleman, but for fewer variables. In terms of mechanical effort, the number of blanks `NB_BLANK` is the only remaining significant factor. This reduction is to be expected as the participant has learned the key interactions of the editor. We attribute the difference in terms of blank selection to the type of editor. As a textual editor, the participant is in a constant flow of interaction, which invites more

Variable		Mean	Std. Deviation	F	P-value
`NB_Selection`	Gentleman MPS	49.14 55.45	12.12 20.69	(1, 42) = 1.53	0.223
`NB_BLANK` *	Gentleman MPS	2.86 5.64	2.77 4.50	(1, 42) = 6.06	**0.018**
`NB_MISS`	Gentleman MPS	1.36 1.09	1.62 1.27	(1, 42) = 0.39	0.538
`NB_CxtChange` *	Gentleman MPS	0.32 9.95	1.13 4.37	(1, 42) = 100.30	**< 0.001**
`NB_IDLE`	Gentleman MPS	2.23 2.41	1.34 1.50	(1, 42) = 0.18	0.674
`T_IDLE`	Gentleman MPS	9.45 11.86	7.04 9.32	(1, 42) = 0.94	0.339
`NB_QA`	Gentleman MPS	1.50 1.95	1.14 1.99	(1, 42) = 0.87	0.358
`T_QA`	Gentleman MPS	11.45 15.36	9.69 19.46	(1, 42) = 3.50	0.068
`T_TOTAL`	Gentleman MPS	295.00 358.18	107.59 143.82	(1, 42) = 2.72	**0.106**
`NB_BLOCK`	Gentleman MPS	0.00 0.32	0.00 0.65	(1, 42) = 5.33	**0.026**

Table 6.5. Task 2 ANOVA results

interaction and, therefore higher chance of making an unnecessary interaction. The number of context changes (`NB_CtxChange`) remained as significant as with task 1. However, some participants mitigated this issue by repositioning the windows, as shown in Fig. 6.3.

Task 3. As seen in the descriptive analysis, there is no significant difference between the tool. We attribute this parity to the nature of the task and the greater familiarity with the tool. As this was an evaluation, the participant did not need to interact as much, which would invite more selection and possibly more blanks. Although they had to apply some changes, they did not have to create new structures but edit the existing ones.

Survey. For the survey, more than half the variables indicates a better experience with Gentleman than with MPS. As a rich UI composed of widgets and buttons, most participants navigated with the mouse and appreciated the experience as indicated by `Mouse Navigation`. As discussed in the task, Gentleman offers a simple and practical UI, which exposes the essential and does not clutter the view, resulting in a very favorable opinion for `User Interface` from the participants. On the same note, being explicit also translated in better discoverability, indicated by `Actions` coupled with `Executions`. At a lesser level of significance, we have also `Control` and `State`. As most actions were exposed, there was less to learn or remember for Gentleman, resulting in a better score. However, as noted in task 3, after getting familiar

Variable		Mean	Std. Deviation	F	P-value
User Interface *	Gentleman MPS	4.73 3.95	0.456 0.84	(1, 42) = 14.28	**< 0.001**
Control *	Gentleman MPS	4.77 4.14	0.43 0.99	(1, 42) = 7.65	**0.008**
State *	Gentleman MPS	4.73 4.27	0.46 0.77	(1, 42) = 5.71	**0.021**
Keyboard Navigation	Gentleman MPS	4.30 4.24	0.82 0.77	(1, 42) = 0.04	0.839
Mouse Navigation *	Gentleman MPS	4.86 4.05	0.35 0.95	(1, 42) = 14.35	**< 0.001**
Reuse value	Gentleman MPS	4.64 4.09	0.58 0.75	(1, 42) = 3.71	0.061
Focus	Gentleman MPS	4.59 4.09	0.80 0.92	(1, 42) = 7.27	**0.010**
Recover	Gentleman MPS	4.73 4.41	0.46 0.73	(1, 42) = 2.98	0.092
Actions *	Gentleman MPS	4.73 3.55	0.46 1.14	(1, 42) = 20.28	**< 0.001**
Executions *	Gentleman MPS	4.77 4.05	0.53 0.78	(1, 42) = 12.99	**< 0.001**

Table 6.6. Survey ANOVA results

with the syntax and the UI, evaluating the model takes the same amount of effort, which results in the lowest significant p-value for `State`.

6.5.3. Hypothesis validation

Hyp. 1: Gentleman's modeling approach improves the productivity by reducing the mechanical effort.

H_{0,Mec_Effort}: The tool does not impact the mechanical effort required to accomplish a task with a projectional editor. There is no difference between Gentleman and MPS when considering `NB_Selection`, `NB_Miss`. `NB_Blank`, and `NB_CtxChange` when completing a task.

H_{1,Mec_Effort}: The identified metrics produce lower results when using Gentleman, compared with MPS.

As identified in our metrics, in tasks 1 and 2, `NB_Selection`, CodeNB_Blank and `NB_CtxChange` are significantly lower in Gentleman, compared to MPS. Therefore, we can reject the null hypothesis and conclude that **the mechanical effort is reduced with Gentleman**.

Hyp. 2: Gentleman's modeling approach improves the productivity by reducing the cognitive effort.

H_{0,Cog_Effort}: The tool does not impact the cognitive effort required to accomplish a task with a projectional editor. There is no difference between Gentleman and MPS when considering `T_Idle`, `T_QA`, `NB_Idle`, `NB_QA` and `NB_BLOCK` when completing a task.

H_{1,Cog_Effort}: The identified metrics produce lower results when using Gentleman, compared with MPS.

Looking at our significant variables, `T_Idle`, `T_QA` are much lower for Gentleman in task 1, and `NB_QA` is significantly lower for Gentleman for task 1, but this is not observed in task 2. However, `NB_BLOCK` is significantly lower in both tasks. Therefore, we can reject the null hypothesis and conclude that that **The cognitive effort is reduced with Gentleman**.

Hyp. 3: Gentleman's modeling approach improves the productivity by reducing modeling time.

H_{0,Com_Time}: The tool does not impact the time taken to accomplish a task with a projectional editor. There is no difference between Gentleman and MPS when considering `T_Total` when completing a task.

H_{1,Com_Time}: The time taken to complete a task is reduced when using Gentleman.

As identified in our metrics, in tasks 1 and 2, `T_Total` is significantly lower in Gentleman, especially for task 1. Therefore, we can reject the null hypothesis and conclude that **the time taken to complete a task is reduced with Gentleman**.

Hyp. 4: Gentleman's modeling approach reduces the number of errors.

H_{0,Err_Number}: The tool does not impact the number of errors produced during a task with a projectional editor. There is no difference between Gentleman and MPS when considering `NB_Typos` and `NB_DesignErr` when completing a task.

H_{1,Err_Number}: The number of errors produced during a task is reduced when using Gentleman.

The participant did not produce significantly more or less errors using Gentleman. Therefore, we cannot reject the null hypothesis and conclude that **the number of errors is not impacted when using Gentleman**.

Hyp. 5: Gentleman's modeling approach improves the recovery rate of errors.

$H_{0,Err\_Recover}$: The tool does not impact the ability to recover from errors with a projectional editor. There is no difference between Gentleman and MPS when considering `RT_Detection` and `RT_Recover` when completing a task.

$H_{1,Err\_Recover}$: The recovery rate of errors during a task is higher when using Gentleman.

In accord with the previous hypothesis, the participant did not produce a significantly different recovery rate when using either editor. Therefore, we cannot reject the null hypothesis and conclude that **the recovery rate is not impacted when using Gentleman**.

Hyp. 6: Gentleman's modeling approach improves the user experience.

$H_{0,Usability}$: The tool does not impact the user experience.

$H_{1,Usability}$: The user experience is improved in Gentleman.

Considering the survey results, several variables identified in Section 6.5.2 indicates a significantly better experience with Gentleman. Therefore, we can reject the null hypothesis and conclude that **the user experience is improved with Gentleman**

6.6. Discussion

Having completed our analysis, it is also important to consider the context that brought such results. From our experience, Gentleman significantly outperforms MPS in mechanical effort, displaying a lower number of selections, which is made less significant with further practice. This is in line with the known usability concerns of MPS as discussed in Chapter 1. However, it should be noted that this experiment is based on one projection and could produce different results for MPS with another projection. This does not apply to all mechanical variables, though, such as `NB_CtxChange`, which, regardless of the given projection, would still be as significant as it is inherent to UI implementation. In terms of cognitive effort, Gentleman does perform better, but not overwhelmingly, according to the results. However, many participants indicated in a debriefing session following the tasks that for MPS, without the tutorial, it would have been a lot more difficult to use the editor. This is revealed here by `NB_BLOCK`, which is relevant in task 1 and even task 2. Considering the performance of the participants throughout the task, we notice a significant decrease in the disparity between the two editors between tasks 1 and 2. The only remaining significant variables are the total time and the number of context changes. Task 2 being more complex, participants went through it much slower in both editors but decelerated more in Gentleman's case. For MPS, after the first task and the tutorial, the participant mastered the key interactions. However, they still struggled with the copy-paste maneuver as the focus wasn't clear. On Gentleman's side, many trials were required to do the copy-pasting as well. It should be noted that none of the participants has ever encountered such an implementation of copy-paste in a previous GUI. Lastly, it should be remembered that MPS is a commercial product with a team of experienced professionals and over ten years of iteration, whereas Gentleman is currently being developed by a single developer and is still in a beta release version. As such, this experiment was also used as beta testing of the editor.

Having now put the task result in context, we can use the hypotheses evaluation to answer the research questions.

RQ1: *"Does the approach used by Gentleman improve the user's productivity compared to existing projectional editors?"* As seen in all our examples, Gentleman uses a very rich notation. Usually, the projection is a composition of text, images, widgets, and buttons, allowing the user to quickly and easily parse the content. Additionally, the simple user interface, indicated as very intuitive by most participants, makes the potential actions explicit to the user. On the other side, MPS has a crowded user interface, which intimidates newcomers. Moreover, the editor's presentation and interactions are programmer-oriented, making it less intuitive to practitioners of other domains. As a result, per our hypotheses results, we may conclude that **Gentleman improves the the user's productivity**.

RQ2: *"Does the approach used by Gentleman improve the ability to understand a model compared to existing projectional editors?"* A model is perceived and materialized through a concrete syntax. Therefore understanding a model is a factor of the degree of familiarity with the syntax. As highlighted in our analysis, after performing tasks 1 and 2, the participant had gained enough familiarity with the syntax to perform evenly with Gentleman and MPS when assessing the state of the model. However, most participants remarked that Gentleman's projections were more intuitive and thus required less practice to get used to. Therefore, when considering a projection with which the user is familiar, as shown in our hypotheses results, we may conclude that **Gentleman does not improve nor hamper the ability to understand a model**.

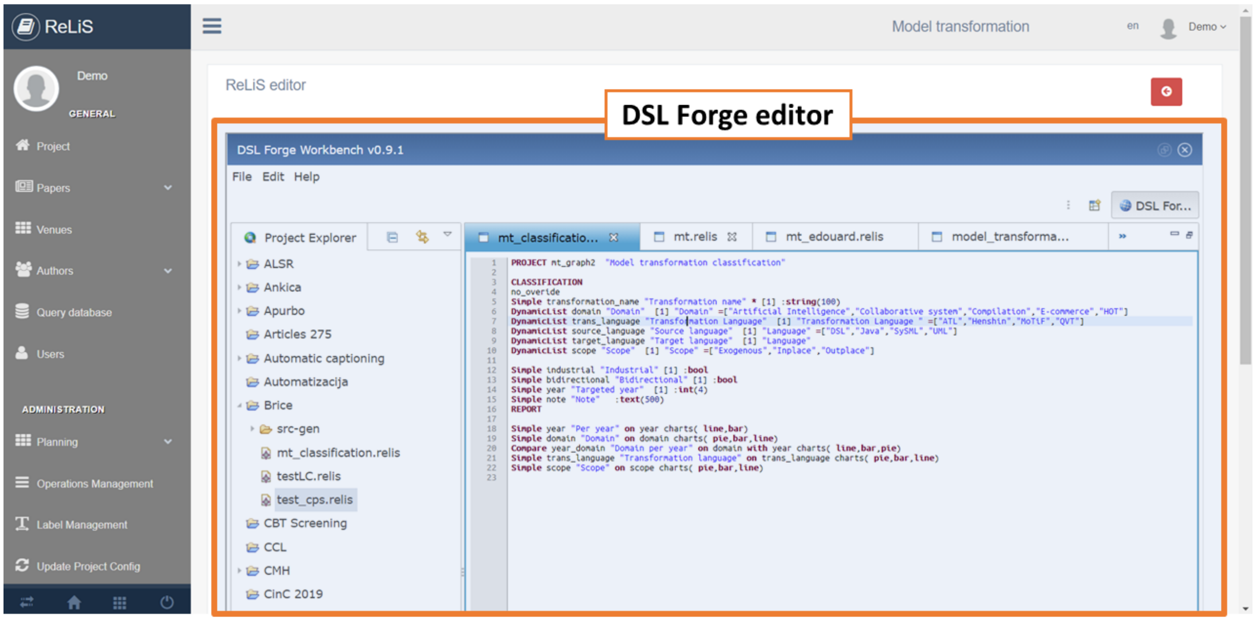
6.7. Threats to validity

Before we can accept the results and conclusions of the experiment, it is crucial to consider the actions that were taken in order to achieve adequate validity.

First, we consider the threat to construct validity, which refers to the extent to which the experimental setting reflects the construct under study. To minimize this threat, we did not communicate the goal of the experiment. As indicated in the discussion, our results are based on one projection, which might not adequately represent the tool's potential. Additionally, our choice of variables might not fully reflect the hypothesis, such as the cognitive effort.

Second, we consider the threat to internal validity related to the influences that can affect the factors with respect to causality. Our experience was conducted on two groups, differentiated by order of the tool with which they performed the tasks. We did a preliminary analysis to verify the influence of the order, and it was concluded that the order of the tool had no significant impact on the survey results nor the performance metrics. Additionally, to verify that the significance of the measures was not the result of our sampling, we performed an ETA squared test, which validated our sample size as large enough to consider the identified variables. Since we performed the experiment using a remote connection, there were few instances of lag, but we did not note any great disturbance with the participant.

Third, we consider the threat to external validity, which is concerned with generalization. Indeed, the subject population may not be representative of the entire population that we want to generalize. To deal with this threat, we used a confidence interval of 95%. This means that if conclusions followed a normal distribution, the results would be true 95% of the time every time evaluation is repeated.



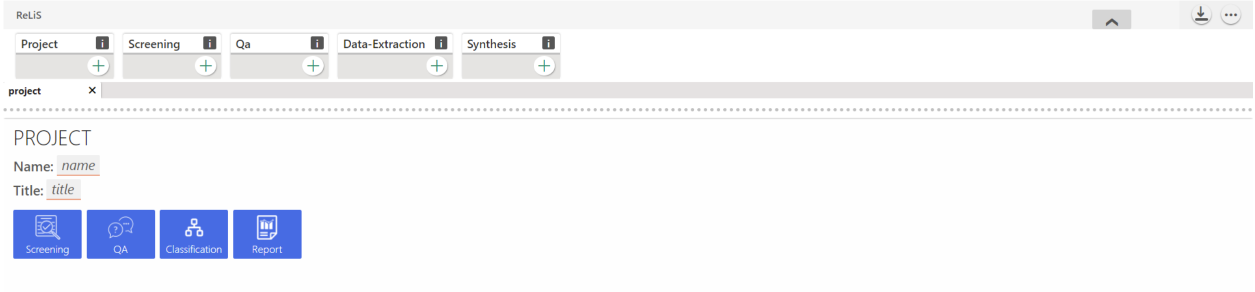
```
PROJECT mt_graph2  "Model transformation classification"

CLASSIFICATION
no_overide
Simple transformation_name "Transformation name" * [1] :string(100)
DynamicList domain "Domain"  [1] "Domain" =["Artificial Intelligence","Collaborative system","Compilation","E-commerce","HOT"]
DynamicList trans_language "Transformation Language"  [1] "Transformation Language " =["ATL","Henshin","MoTiF","QVT"]
DynamicList source_language "Source language"  [1] "Language" =["DSL","Java","SySML","UML"]
DynamicList target_language "Target language"  [1] "Language"
DynamicList scope "Scope"  [1] "Scope" =["Exogenous","Inplace","Outplace"]

Simple industrial "Industrial" [1] :bool
Simple bidirectional "Bidirectional" [1] :bool
Simple year "Targeted year"  [1] :int(4)
Simple note "Note"   :text(500)
REPORT

Simple year "Per year" on year charts( line,bar)
Simple domain "Domain" on domain charts( pie,bar,line)
Compare year_domain "Domain per year" on domain with year charts( line,bar,pie)
Simple trans_language "Transformation language" on trans_language charts( pie,bar,line)
Simple scope "Scope" on scope charts( pie,bar,line)
```

Fig. 6.5. ReLiS DSL Forge editor

Lastly is the threat to conclusion validity, concerned with the relationship between the treatment and the outcome. We have mitigated this threat by having a very varied sample of participants reflecting convenience sampling. This threat was also minimized by providing the participants with a hands-on tutorial for each editor. Also, our experiments may be threatened by the reliability of our measures. We used objective metrics such as time and click, which are more reliable than subjective measures.

6.8. Integration with ReLiS

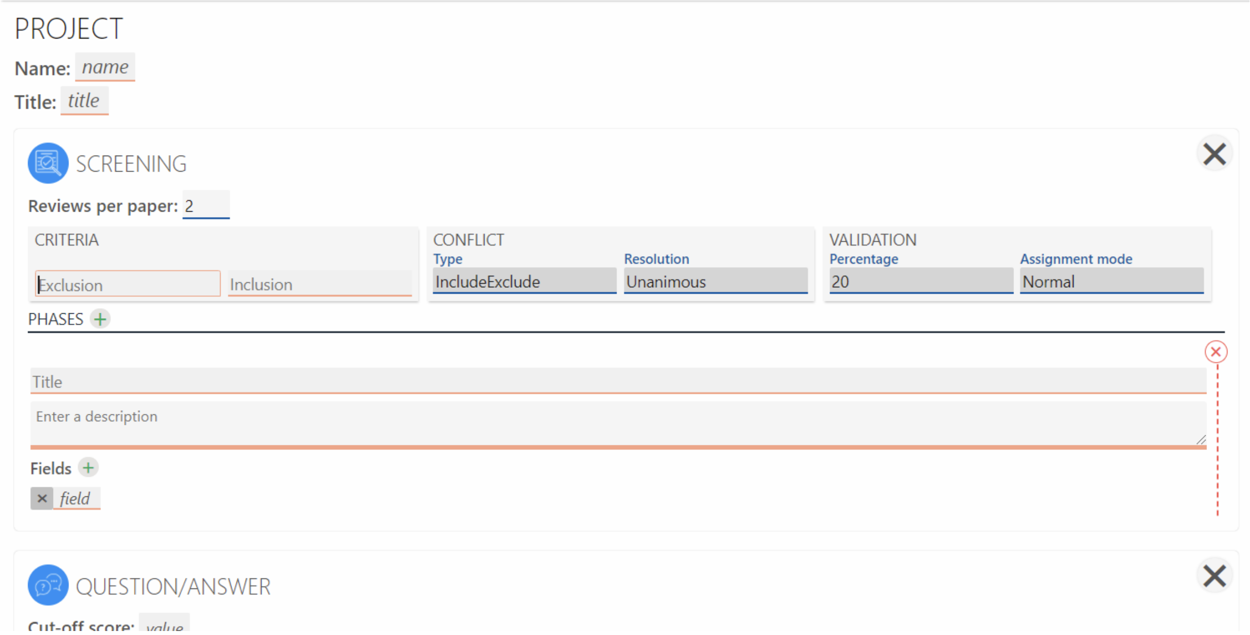
Gentleman is first and foremost an editor and thus not as complete as an IDE or a language workbench such as MPS. Therefore, most use-cases will have Gentleman embedded in a much larger system. As introduced in Section 3.4, one of Gentleman's strengths is its interoperability. To highlight this capability, Gentleman was successfully integrated into the web platform ReLis. ReLiS is a tool to automatically install and configure systematic reviews projects to conduct them collaboratively and iteratively on the cloud. [**8**]. It features a domain-specific modeling language using DSLFORGE [**56**] based on an Ecore model and an Xtext grammar. The editor is integrated into the platform and is used to create and edit configuration files, as displayed in Fig. 6.5. As the editor is targeting a wide audience of experts, this basic textual approach is not well suited to most. This makes ReLiS a very good candidate to transition to a Gentleman editor and provide its users with a user-friendly editor.



Name: name

Title: title

Fig. 6.6. ReLiS editor: initialization



Name: name

Title: title

Reviews per paper: 2

Exclusion | Inclusion

Type: IncludeExclude

Resolution: Unanimous

Percentage: 20

Assignment mode: Normal

Title

Enter a description

field

Cut-off score: value

Fig. 6.7. ReLiS editor: Screening

First, we created the concepts identified in the Ecore model, and with the input of some of ReLiS active users, we created a projection that would improve their editing experience. Here we present the integrated Gentleman editor to the ReLiS platform. Fig. 6.6 presents the Gentleman editor with a blank project. Here, the user is informed of the different configuration sections, which they can choose to open and configure. Fig. 6.7 shows the configuration of the *Screening* section. Here elements are grouped by categories such as *Criteria*, *Conflict*, and *Validation*. This makes it easier to scan the content. Finally, similar to the traffic light editor, this one also uses icons to help users recognize a section.

As a web application, the integration did not require much work. Following the procedure described in Section 3.4.1, we added the core script to the page and initialized the Gentleman editor with the previously created concept schema and projection schema. With those in place, a Gentleman editor can be used in place of the previous DSL Forge editor, thus completing our Integration. However, as the ReLiS environment is automatically generated, some additional work is required to include the gentleman artifacts in the process, thus avoiding the need to manually update the concepts and incidentally the projections.

Chapter 7

Conclusion

We conclude by summarizing the contributions of this thesis and outlining future work. The work presented in this thesis makes several contributions to MDE, more specifically modeling language and modeling tool.

7.1. Summary

The emphasis on modeling has opened software development to a larger public, composed of experts of various domains with very different requirements. Projectional editing supports various notations that may easily be interchanged and recomposed, making it a suitable approach to embrace this diversity and make modeling more accessible to domain experts. However, current solutions are heavyweight, closed, and suffer from poor usability. In this thesis, we introduced the projectional editor, Gentleman. It is lightweight to favor interoperability and accessibility and web-based to ease the process of integration and distribution. Gentleman generates an editing environment to build and explore models with the chosen concepts and projections, loaded in the editor.

In Gentleman, concepts are used to define the structure of the model. This concept-based approach moves away from the OO paradigm found in UML and other modeling languages by considering modeling in more direct terms, unrelated to the code executed afterward. This results in less cognitive effort when defining a model, as no prior knowledge of any programming paradigm is required. This is illustrated with the reference and set primitives, which are left implicit in the UML approach but can be manipulated directly in Gentleman, like a string or a number. The presented approach offers an extension mechanism with prototypes, similar to prototype-based programming, and a specialization mechanism with derivatives, making it simple to create enumerations. Lastly, concepts relations are considered following two dimensions to allow self-defined characteristics as properties and user-defined characteristics as attributes, which may define constraints on the target concept.

As a projectional editor, with Gentleman, projections are used to interact and manipulate the concepts. Our approach uses specialized elements to make it easy for the user to scan the content of a projection and interact with it, which was confirmed by most participants in the user study, who praised the intuitiveness of the projections. Gentleman leverages web technologies such as CSS to style the projections, HTML5 widgets for practical data entry, and HTML templates for reusability. This enables the creation of visually rich and intuitive projections composed of widgets, buttons, text, and images.

Gentleman offers some practical editor services to assist the user and ensure a good user experience. It gives the ability to create a link instance to visualize part of a concept, the garbage collecting service allowing the user to see the complete state of the model and easily undo a deletion, and other common services such as copy-pasting and importing or exporting an existing model. Built as a JS library, it is easy to integrate with any web application, which was showcased with the ReLiS integration.

Lastly, to evaluate the approach, we invited a group of participants to perform a series of modeling tasks using Gentleman and Jetbrains MPS. The tasks served to evaluate the efficiency of the participant when creating a model with an unknown syntax and their ability to understand a model in both tools. To do so, we measured the cognitive and mechanical effort to accomplish each task, the total time taken to complete the task, the number of errors produced, and the effort to correct errors when completing the activities. The experiment concluded with a very positive note for Gentleman, which displayed better performance than MPS for the modeling activities. The participants then also expressed their enjoyment with the tool, praising its simplicity and intuitiveness.

7.2. Outlook

Having laid the foundation of modeling using a concept-based approach, we can now dig deeper into the paradigm. The current structure of a concept is relatively static, both externally and internally, as every parameter of an attribute or property is defined in the metamodeling phase. Therefore, to embrace the dynamic nature of modeling, we plan to introduce late-binding attributes, where some parameters would be resolved during the modeling phase, where the user creates a model. The properties are, at the moment, only capable of elementary computation but could be developed much further to enable predicate-based property where the operation is applied under some condition, and context-based property, where the results depend on the presence of some concepts. Constraints are currently bound to primitives, which impacts their reusability. This could be solved by making them first-class citizens, which would allow constraints to be manipulated outside a specific concept and be reused in multiple contexts.

The concept-based approach uses projectional editing to enable interaction with the concepts. Even though projections in Gentleman bring lots of benefits in terms of usability, currently, most projection components are predefined. This impacts the flexibility of projections and the creativity of designers. Therefore, we will explore new primitives to give the designer the ability to create a new element or component. Current work is adding support for graphical projection and new graphic-specific layouts. This will enable the construction of floating elements, shapes, and diagrams. Projections are currently defined independently of the editor, resulting in arbitrary interaction. We will explore how to add editor-specific configurations into the projection definition.

Being a library, Gentleman offers no reliable data storage facility, nor can it be used to execute and validate models using domain-specific rules. As such, we will explore integrating Gentleman with a modeling back-end. Additionally, the concept-based approach is built to promote reusability, but few editor services have been developed to encourage such action. Therefore, we plan on creating a repository of concepts to which anyone could connect and share their concepts. Lastly, Gentleman is currently mostly capable of static modeling, where most pieces are predefined, and the user cannot change the behavior during the execution. Therefore, a topic to explore could be integrating more dynamic elements to Gentleman, such as model transformations.

References

[1] Silvia Abrahão, Francis Bourdeleau, Betty Cheng, Sahar Kokaly, Richard Paige, Harald Stöerrle et Jon Whittle : User experience for model-driven engineering: Challenges and future directions. *In 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 229–236. IEEE, 2017.

[2] L. Agner et T. Lethbridge : A survey of tool use in modeling education. *In Model Driven Engineering Languages and Systems*, pages 303–311. IEEE, 2017.

[3] Rachel Andrew : *The New CSS Layout*. A Book Apart, 2017.

[4] Jim Arlow et Ila Neustadt : *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.

[5] Peter J Ashenden : *The designer's guide to VHDL*. Morgan Kaufmann, 2010.

[6] Jon Bentley : Programming pearls: little languages. *Communications of the ACM*, 29(8):711–721, 1986.

[7] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert et J. Siegmund : Efficiency of projectional editing: A controlled experiment. *In International Symposium on Foundations of Software Engineering*, pages 763–774, 2016.

[8] Brice Michel Bigendako : Relis: un outil flexible pour réaliser des revues systématiques itératives et collaboratives. Mémoire de D.E.A., Université de Montréal, 2018.

[9] Meinte Boersma : Post-mortem for más. https://medium.com/@dslmeinte/post-mortem-for-m2018.

[10] Marco Brambilla, Jordi Cabot et Manuel Wimmer : Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[11] Karen Brennan et Mitchel Resnick : New frameworks for studying and assessing the development of computational thinking. *In Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25, 2012.

[12] Frank Budinsky, David Steinberg, Raymond Ellersick, Timothy J Grose et Ed Merks : *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional, 2004.

[13] Jordi Cabot et Martin Gogolla : Object constraint language (ocl): a definitive guide. *In International school on formal methods for the design of computer, communication and software systems*, pages 58–90. Springer, 2012.

[14] Raymond Camden : *Client-side data storage: keeping it local.* " O'Reilly Media, Inc.", 2015.

[15] F. Campagne : *The MPS Language Workbench Volume I: The Meta Programming System*. CreateSpace Independent Publishing Platform, 3rd édition, 2016.

[16] Y-S. Chang et N-W. Lin : A tool for constructing syntax-directed editors. *In Asia-Pacific Software Engineering Conference*. IEEE, 2005.

[17] DSL Consultancy : http://mas-wb.appspot.com/.

[18] Alessandro Del Sole : *Visual Studio Code Distilled: Evolved Code Editing for Windows, MacOS, and Linux*. Apress, 2018.

[19] Christophe Dony, Jacques Malenfant et Pierre Cointe : Prototype-based languages: from a new taxonomy to constructive proposals and their validation. *In Conference proceedings on Object-oriented programming systems, languages, and applications*, pages 201–217, 1992.

[20] Jon Duckett : *HTML & CSS: design and build websites*, volume 15. Wiley Indianapolis, IN, 2011.

[21] S. Erdweg, T. Van Der Storm, M. Völter *et al.* : The state of the art in language workbenches. *In International Conference on Software Language Engineering*, pages 197–217. Springer, 2013.

[22] Sebastian Erdweg, Tijs Van Der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh *et al.* : Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures*, 44:24–47, 2015.

[23] Moritz Eysholdt et Heiko Behrens : Xtext: implement your language faster than the quick and dirty way. *In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309, 2010.

[24] Andrew Forward et Timothy C Lethbridge : Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. *In Proceedings of the 2008 international workshop on Models in software engineering*, pages 27–32, 2008.

[25] M. Fowler : Language workbenches: The killer-app for domain specific languages. `https://martinfowler.com/articles/languageWorkbench.html`, 2005.

[26] Martin Fowler : *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[27] André WB Furtado et André LM Santos : Using domain-specific modeling towards computer games development industrialization. *In The 6th OOPSLA workshop on domain-specific modeling (DSM06)*. Citeseer, 2006.

[28] Miguel Gamboa et Eugene Syriani : Improving user productivity in modeling tools by explicitly modeling workflows. *Software & Systems Modeling*, 18(4):2441–2463, 2019.

[29] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides et Design Patterns : *Elements of reusable object-oriented software*, volume 99. Addison-Wesley Reading, Massachusetts, 1995.

[30] Anna Gerber et Kerry Raymond : Mof to emf: there and back again. *In Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 60–64, 2003.

[31] Richard C Gronback : *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

[32] V. Gupta : Ui programming: Handling events and using advanced widgets. *Accelerated GWT: Building Enterprise Google Web Toolkit Applications*, pages 105–134, 2008.

[33] Sam Hampton-Smith : Css flexible box layout. *In Pro CSS3 Layout Techniques*, pages 73–101. Springer, 2016.

[34] R. Harris et R. Warner : *The definitive guide to SWT and JFace*. Apress, 2004.

[35] Melanie Hartmann et Max Muhlhauser : Context-aware form filling for web applications. *In 2009 IEEE International Conference on Semantic Computing*, pages 221–228. IEEE, 2009.

[36] D. Hermes : *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin. Forms Fundamentals*. Apress, 2015.

[37] Paul Hudak : Modular domain specific languages and tools. *In Proceedings. Fifth international conference on software reuse (Cat. No. 98TB100203)*, pages 134–142. IEEE, 1998.

[38] J. Hutchinson, J. Whittle, M. Rouncefield et S. Kristoffersen : Empirical assessment of mde in industry. *In International Conference on Software Engineering*, pages 471–480. ACM, 2011.

[39] Lvar Jacobson et James Rumbaugh Grady Booch : *The unified modeling language reference manual.* 2021.

[40] Caroline Jarrett et Gerry Gaffney : *Forms that work: Designing Web forms for usability.* Morgan Kaufmann, 2009.

[41] Mehdi Jazayeri : Some trends in web application development. *In Future of Software Engineering (FOSE'07)*, pages 199–213. IEEE, 2007.

[42] Bruce Johnson : *Professional visual studio 2012.* John Wiley & Sons, 2012.

[43] J. Johnson : *Designing with the mind in mind: simple guide to understanding user interface design guidelines.* Elsevier, 2013.

[44] Frédéric Jouault et Jean Bézivin : Km3: a dsl for metamodel specification. *In International Conference on Formal Methods for Open Object-Based Distributed Systems*, pages 171–185. Springer, 2006.

[45] Frédéric Jouault et Ivan Kurtev : Transforming models with atl. *In International Conference on Model Driven Engineering Languages and Systems*, pages 128–138. Springer, 2005.

[46] L. C. Kats et E. Visser : The spoofax language workbench: rules for declarative specification of languages and ides. *In Object oriented programming systems languages and applications*, pages 444–463, 2010.

[47] Steven Kelly et Juha-Pekka Tolvanen : *Domain-specific modeling: enabling full code generation.* John Wiley & Sons, 2008.

[48] Stuart Kent : Model driven engineering. *In International conference on integrated formal methods*, pages 286–298. Springer, 2002.

[49] Zachary A King, Andreas Dräger, Ali Ebrahim, Nikolaus Sonnenschein, Nathan E Lewis et Bernhard O Palsson : Escher: a web application for building, sharing, and embedding data-rich visualizations of biological pathways. *PLoS computational biology*, 11(8):e1004321, 2015.

[50] Steve Kinney : *Electron in Action.* Simon and Schuster, 2018.

[51] Donald E Knuth : Semantics of context-free languages. *Mathematical systems theory*, 2(2):127–145, 1968.

[52] D. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. Polack et G. Botterweck : Taming emf and gmf using model transformation. *In Model Driven Engineering Languages and Systems*, pages 211–225. Springer, 2010.

[53] Dimitrios S Kolovos, Antonio García-Domínguez, Louis M Rose et Richard F Paige : Eugenia: towards disciplined and automated development of gmf-based graphical model editors. *Software & Systems Modeling*, 16(1):229–255, 2017.

[54] Thomas Kühne : Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):369–385, 2006.

[55] Louis-Edouard Lafontant et Eugene 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*, pages 1–5, 2020.

[56] Amine Lajmi, Jabier Martinez et Tewfik Ziadi : Dslforge: Textual modeling on the web. *Demos@ MoDELS*, 1255, 2014.

[57] Benoît Langlois, Consuela-Elena Jitia et Eric Jouenne : Dsl classification. *In OOPSLA 7th workshop on domain specific modeling*, 2007.

[58] Erhan Leblebici, Anthony Anjorin et Andy Schürr : Developing emoflon with emoflon. *In International Conference on Theory and Practice of Model Transformations*, pages 138–145. Springer, 2014.

[59] Avraham Leff et James T Rayfield : Web-application development using the model/view/controller design pattern. *In Proceedings fifth ieee international enterprise distributed object computing conference*, pages 118–127. IEEE, 2001.

[60] Jochen Ludewig : Models in software engineering–an introduction. *Software and Systems Modeling*, 2(1):5–14, 2003.

[61] Lechanceux Luhunu et Eugene Syriani : Comparison of the expressiveness and performance of template-based code generation tools. *In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, pages 206–216, 2017.

[62] I Scott MacKenzie et Kumiko Tanaka-Ishii : *Text entry systems: Mobility, accessibility, universality*. Elsevier, 2010.

[63] Luis Mandel et Maria Victoria Cengarle : On the expressive power of ocl. *In International Symposium on Formal Methods*, pages 854–874. Springer, 1999.

[64] Jatinder Manhas : A study of factors affecting websites page loading speed for efficient web performance. *International Journal of Computer Sciences and Engineering*, 1(3):32–35, 2013.

[65] Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky et Ákos Lédeczi : Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS*, 1237:41–60, 2014.

[66] Erica Mealy, David Carrington, Paul Strooper et Peta Wyeth : Improving usability of software refactoring tools. *In 2007 Australian Software Engineering Conference (ASWEC'07)*, pages 307–318. IEEE, 2007.

[67] Raul Medina-Mora et Peter H. Feiler : An incremental programming environment. *IEEE Transactions on Software Engineering*, (5):472–482, 1981.

[68] Jim Melton et Alan R Simon : *Understanding the new SQL: a complete guide*. Morgan Kaufmann, 1993.

[69] Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry et Benoit Combemale : Modeling modeling modeling. *Software & Systems Modeling*, 11(3):347–359, 2012.

[70] Gail C Murphy, Mik Kersten et Leah Findlater : How are java software developers using the elipse ide? *IEEE software*, 23(4):76–83, 2006.

[71] Jakob Nielsen : Ten usability heuristics, 2005.

[72] Erik G Nilsson : Design patterns for user interface for mobile applications. *Advances in engineering software*, 40(12):1318–1328, 2009.

[73] David Notkin : The gandalf project. *Journal of Systems and Software*, 5(2):91–105, 1985.

[74] Florian Noyrit, Sébastien Gérard et Bran Selic : Facademetamodel: masking uml. *In International Conference on Model Driven Engineering Languages and Systems*, pages 20–35. Springer, 2012.

[75] Edward E Ogheneovo *et al.* : On the relationship between software complexity and maintenance costs. *Journal of Computer and Communications*, 2(14):1, 2014.

[76] Terence J. Parr et Russell W. Quong : Antlr: A predicated-ll (k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

[77] Vaclav Pech, Alex Shatalin et 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*, pages 165–168, 2013.

[78] Marian Petre : Uml in practice. *In 2013 35th international conference on software engineering (icse)*, pages 722–731. IEEE, 2013.

[79] Dan Pilone et Neil Pitman : *UML 2.0 in a Nutshell.* " O'Reilly Media, Inc.", 2005.

[80] Anirudh Prabhu et Aravind Shenoy : Introducing materialize. *In Introducing Materialize*, pages 1–9. Springer, 2016.

[81] Andreas Prinz et Gergely Mezei : The art of bootstrapping. *In International Conference on Model-Driven Engineering and Software Development*, pages 182–200. Springer, 2019.

[82] Helen Purchase, Linda Colpoys, Matthew McGill, David Carrington et Carol Britton : Uml class diagram syntax: An empirical study of comprehension. pages 113–120, 12 2001.

[83] Dave Raggett, Arnaud Le Hors, Ian Jacobs *et al.* : Html 4.01 specification. *W3C recommendation*, 24, 1999.

[84] Ó. S. Ramón, J. S. Cuadrado et J. G. Molina : Model-driven reverse engineering of legacy graphical user interfaces. *Automated Software Engineering*, 21:147–186, 2014.

[85] Thomas Reps et Tim Teitelbaum : The synthesizer generator. *ACM Sigplan Notices*, 19(5):42–48, 1984.

[86] M. Resnick, J. Maloney et A. et al. Monroy-Hernández : Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[87] Dennis M Ritchie, Brian W Kernighan et Michael E Lesk : *The C programming language.* Prentice Hall Englewood Cliffs, 1988.

[88] Arnold Robbins, Elbert Hannah et Linda Lamb : *Learning the vi and Vim Editors.* " O'Reilly Media, Inc.", 2008.

[89] Louis M Rose, Richard F Paige, Dimitrios S Kolovos et Fiona AC Polack : The epsilon generation language. *In European Conference on Model Driven Architecture-Foundations and Applications*, pages 1–16. Springer, 2008.

[90] Rijul Saini, Shivani Bali et Gunter Mussbacher : Towards web collaborative modelling for the user requirements notation using eclipse che and theia ide. *In 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, pages 15–18. IEEE, 2019.

[91] Vinoth Pandian Sermuga Pandian, Sarah Suleri et Prof Dr Matthias Jarke : Uisketch: A large-scale dataset of ui element sketches. *In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021.

[92] Zohreh Sharafi, Alessandro Marchetto, Angelo Susi, Giuliano Antoniol et Yann-Gaël Guéhéneuc : An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. *In 2013 21st International Conference on Program Comprehension (ICPC)*, pages 33–42. IEEE, 2013.

[93] Maria Shitkova, Justus Holler, Tobias Heide, Nico Clever et Jörg Becker : Towards usability guidelines for mobile websites and applications. *In Wirtschaftsinformatik*, pages 1603–1617. Citeseer, 2015.

[94] Charles Simonyi : The death of computer languages, the birth of intentional programming. *In NATO Science Committee Conference*, pages 17–18. Citeseer, 1995.

[95] Kyle Simpson : *You Don't Know JS: ES6 & Beyond.* " O'Reilly Media, Inc.", 2015.

[96] K. Smolander, K. Lyytinen, V-P. Tahvanainen et P. Marttiin : Metaedit – a flexible graphical environment for methodology modeling. *In International Conference on Advanced Information Systems Engineering*, pages 168–193. Springer, 1991.

[97] R. Solmi : *Whole Platform.* Ph.d. thesis, Universitá di Bologna e Padova, mar 2005.

[98] Madeleine Sorapure : Text, image, data, interaction: Understanding information visualization. *Computers and Composition*, 54:102519, 2019.

[99] Jake Spurlock : *Bootstrap: responsive web development.* " O'Reilly Media, Inc.", 2013.

[100] Arjun Srinivasan, Hyunwoo Park, Alex Endert et Rahul C Basole : Graphiti: Interactive specification of attribute-based edges for network modeling and visualization. *IEEE transactions on visualization and computer graphics*, 24(1):226–235, 2017.

[101] Andreas Stefik et Susanna Siebert : An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)*, 13(4):1–40, 2013.

[102] Dave Steinberg, Frank Budinsky, Ed Merks et Marcelo Paternostro : *EMF: eclipse modeling framework*. Pearson Education, 2008.

[103] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf et Matthias Tichy : Henshin: A usability-focused framework for emf model transformation development. *In International Conference on Graph Transformation*, pages 196–208. Springer, 2017.

[104] Ana Maria Şutîi, Mark van den Brand et Tom Verhoeff : Exploration of modularity and reusability of domain-specific languages: an expression dsl in metamod. *Computer Languages, Systems & Structures*, 51:48–70, 2018.

[105] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo et H. Ergin : AToMPM: A web-based modeling environment. *In MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition*, volume 1115, pages 21–25. CEUR-WS.org, 2013.

[106] Eugene Syriani, Lechanceux Luhunu et Houari Sahraoui : Systematic mapping study of template-based code generation. *Computer Languages, Systems & Structures*, 52:43–62, 2018.

[107] Henry S Thompson, David Beech, Murray Maloney et Noah Mendelsohn : Xml schema part 1: structures second edition. *W3C recommendation*, 39, 2004.

[108] Juha-Pekka Tolvanen et Steven Kelly : Metaedit+ defining and using integrated domain-specific modeling languages. *In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 819–820, 2009.

[109] R. Van Der Straeten, T. Mens et S. Van Baelen : Challenges in model-driven software engineering. *In Model Driven Engineering Languages and Systems*, pages 35–47. Springer, 2008.

[110] A. Van Deursen, P. Klint et J. Visser : Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[111] Arie Van Deursen et Paul Klint : Domain-specific language design requires feature descriptions. *Journal of computing and information technology*, 10(1):1–17, 2002.

[112] M. Voelter et S. Lisson : Supporting diverse notations in mps' projectional editor. *In Workshop on The Globalization of Modeling Languages*, volume 1236, pages 7–16. CEUR-WS.org, 2014.

[113] M. Voelter, J. Siegmund, T. Berger et B. Kolb : Towards user-friendly projectional editors. *In International Conference on Software Language Engineering*, volume 8706 de *LNCS*, pages 41–61. Springer, 2014.

[114] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser et GH Wachsmuth : Dsl engineering-designing, implementing and using domain-specific languages. 2013.

[115] Markus Voelter, Bernd Kolb, Tamás Szabó, Daniel Ratiu et Arie van Deursen : Lessons learned from developing mbeddr: a case study in language engineering with mps. *Software & Systems Modeling*, 18(1):585–630, 2019.

[116] Markus Voelter, Daniel Ratiu, Bernd Kolb et Bernhard Schaetz : mbeddr: Instantiating a language workbench in the embedded software domain. *Automated Software Engineering*, 20(3):339–390, 2013.

[117] David Ward, Jim Hahn et Kirsten Feist : Autocomplete as research tool: A study on providing search suggestions. *Information Technology and Libraries*, 31(4):6–19, 2012.

[118] J. Whittle, J. Hutchinson et M. Rouncefield : The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2013.

[119] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden et R. Heldal : Industrial adoption of model-driven engineering: Are the tools really the problem? *In Model Driven Engineering Languages and Systems*, volume 8107 de *LNCS*, pages 1–17. Springer, 2013.

[120] Maarten Wijnants, Robin Marx, Peter Quax et Wim Lamotte : Http/2 prioritization and its impact on web performance. *In Proceedings of the 2018 World Wide Web Conference*, pages 1755–1764, 2018.

[121] Marco Winckler, Vicent Gaits, Dong-Bach Vo, Firmenich Sergio et Gustavo Rossi : An approach and tool support for assisting users to fill-in web forms with personal information. *In Proceedings of the 29th ACM international conference on Design of communication*, pages 195–202, 2011.

Appendix A

Ecore to Gentleman transformation

A.1. Ecore models

A.2. ATL transformation

```
-- @nsURI MM=http://www.eclipse.org/emf/2002/Ecore
-- @path MM1=/Ecore2Gentleman/MetaModels/Gentleman.ecore

module E2G;
create OUT: MM1 from IN: MM, IN_Library: MM1;

rule EPackage2Model {
  from
    pkg: MM!EPackage
  to
    m: MM1!Model (
      name <- pkg.name,
      concepts <- pkg.eClassifiers
    )
}

rule EClass2Concept {
  from
    class: MM!EClass (
      class.abstract = false and class.interface = false
    )
  to
    g: MM1!Concrete (
      name <- class.name,
      structures <- class.eStructuralFeatures,
```

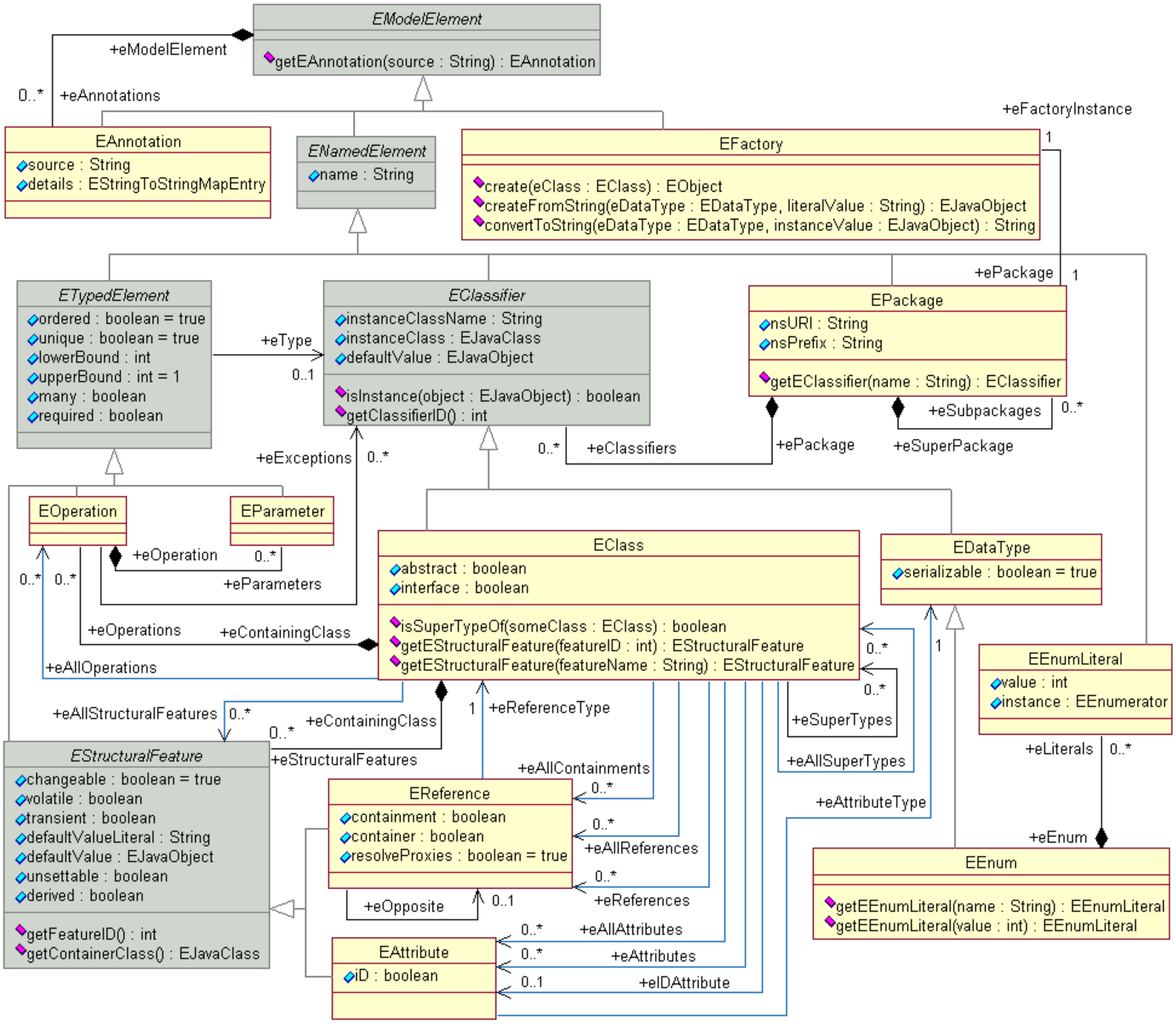


Fig. A.1. Ecore metamodel

```
      prototype <- class.eSuperTypes -> first()
    )
}

rule EAbstractClass2Prototype {
  from
    class: MM!EClass (
      class.abstract = true
    )
  to
    g: MM1!Prototype (
      name <- class.name,
      structures <- class.eStructuralFeatures
```

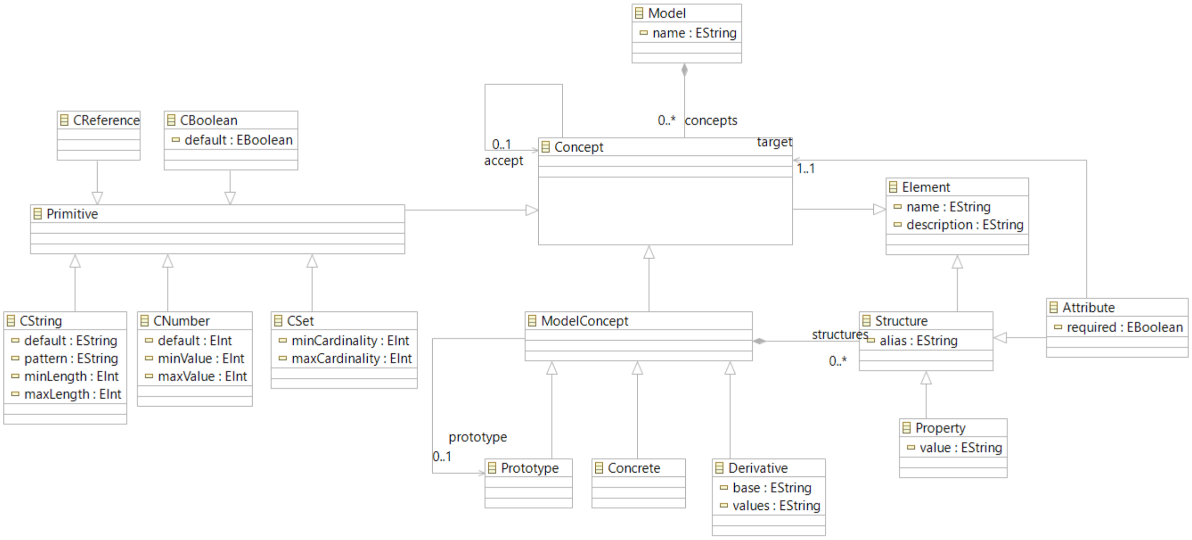


Fig. A.2. Gentleman Ecore model

```
  )
}

rule EEnum2Concept {
 from
  enum: MM!EEnum
 to
  g: MM1!Derivative (
   name <- enum.name,
   base <- ’string’,
   values <- enum.eLiterals->collect(e | e.name)
  )
}

rule EAttribute2Attribute {
 from
  attr: MM!EStructuralFeature (
   attr.many = false
  )
 to
  a: MM1!Attribute (
   name <- attr.name,
   target <- if (attr.eType.name = ’EString’) then
     thisModule.EType2String(attr)
    else
```

```
          if (attr.eType.name = 'EBoolean') then
            thisModule.EType2Boolean(attr)
          else
            if (attr.eType.name = 'EInt' or attr.eType.name = 'EByte')
   then
              thisModule.EType2Number(attr)
            else
              attr.eType
            endif
          endif
        endif
    )
}

rule EAttributeMany2Set {
  from
    attr: MM!EStructuralFeature (
      attr.many = true
    )
  to
    a: MM1!Attribute (
      name <- attr.name,
      target <- thisModule.EType2Set(attr)
    )
}

lazy rule EType2String {
  from
    st: MM!EAttribute
  to
    p1: MM1!CString (
      name <- 'string',
      default <- st.eType.defaultValue
    )
}

lazy rule EType2Boolean {
  from
    st: MM!ETypedElement (
      st.name = 'EBoolean'
    )
  to
    p1: MM1!CBoolean (
      name <- 'boolean'
    )
}

lazy rule EType2Number {
  from
    st: MM!EAttribute
  to
    p1: MM1!CNumber (
      name <- 'number',
      default <- st.eType.defaultValue
    )
}

lazy rule EType2Set {
  from
    st: MM!EReference
  to
    p1: MM1!CSet (
      name <- 'set',
      accept <- st.eType,
      minCardinality <- st.lowerBound,
      maxCardinality <- st.upperBound
    )
}
```

Listing A.1. ATL transformation of an Ecore model to a Gentleman model

A.3. EGL transformation

```
    {"concept": [
        [% for (concept in mm.concepts) { %]
        [% var attributes = concept.structures.select(struct|struct.
   isKindOf(Attribute)); %]
        [% var properties = concept.structures.select(struct|struct.
   isKindOf(Property)); %]
        {
            "id": "[%= concept.id %]",
            "name": "[%= concept.name %]",
            "nature": "[%= concept.type().name.toLowerCase() %]"
```

```
         [% if (concept.hasProperty("prototype") and concept.prototype
<> null) { %]
         ,"prototype": "[%= concept.prototype.name %]"
         [% } %]
         [% if (concept.hasProperty("base") and concept.base <> null) {
 %]
         ,"base": "[%= concept.base %]"
         [% } %]
         [% if (concept.hasProperty("values") and concept.values <>
null) { %]
         ,"values": [[% for (val in concept.values) { %]"[%= val %]"[%
if (concept.values.last() <> val) { %],[% } %][% } %]]
         [% } %]
         [% if (not attributes.isEmpty()) { %]
         ,"attributes": [
             [%for (attr in attributes) { %]
                 {
                     "name": "[%= attr.name %]",
                     "target": {
                         "name": "[%= attr.target.name %]"
                         [% if (attr.target.accept <> null) { %],"
accept": {
                             "name": "[%= attr.target.accept.name %]"
                             }
                         [% } %]
                     }
                 }[% if (attributes.last() <> attr) { %],[% } %]
             [% } %]
         ]
         [% } %]
     }[% if (mm.concepts.last() <> concept) { %],[% } %]
     [% } %]
 ]}
```

Listing A.2. EGL transformation of a Gentleman model to JSON

Appendix B

Gentleman TodoList artefacts

B.1. Concept schema

```
{
    "concept": [
        {
            "id": "11g5bwb5-451d-344f-80d0-1e2138a59ec9",
            "name": "todolist",
            "nature": "concrete",
            "attributes": [
                {
                    "name": "title",
                    "target": {
                        "name": "string",
                        "constraint": {
                            "length": {
                                "type": "range",
                                "range": {
                                    "min": { "value": 1 },
                                    "max": { "value": 50 }
                                }
                            }
                        }
                    },
                    "unique": true,
                    "required": true
                },
                {
                    "name": "tags",
                    "target": {
                            "name": "set",
                            "accept": {
                                "name": "tag"
                            }
                        },
                        "required": true
                    },
                    {
                        "name": "tasks",
                        "target": {
                            "name": "set",
                            "accept": {
                                "name": "task",
                                "default": "single-task"
                            },
                            "constraint": {
                                "cardinality": {
                                    "type": "range",
                                    "range": {
                                        "min": { "value": 1 }
                                    }
                                }
                            }
                        },
                        "required": true
                    }
                ]
            },
            {
                "id": "01f6fc23-pdab-4e74-b9cd-51afb30ecfc5",
                "name": "tag",
                "nature": "concrete",
                "attributes": [
                    {
                        "name": "name",
                        "target": {
                            "name": "string"
                        },
                        "required": true
                    },
                    {
                        "name": "priority",
                        "target": {
                            "name": "priority"
                        },
                        "required": true
                    }
                ]
            },
            {
                "id": "adefde42-abd9-o57f-8e84-ae1f056d26f7",
                "name": "task",
                "description": null,
                "nature": "prototype",
                "attributes": [
                    {
                        "name": "name",
                        "target": {
                            "name": "string",
                            "constraint": {
                                "length": {
                                    "type": "range",
                                    "range": {
                                        "min": { "value": 2 }
                                    }
                                }
                            }
                        },
                        "required": true
                    },
                    {
                        "name": "description",
                        "target": {
                            "name": "string",
                            "constraint": {
                                "length": {
                                    "type": "range",
                                    "range": {
                                        "min": { "value": 2 }
                                    }
                                }
                            }
                        },
                        "required": true
                    },
                    {
                        "name": "completed",
                        "target": {
                            "name": "boolean",
                            "default": false
                        },
                        "required": true
                    },
                    {
                        "name": "tags",
                        "target": {
                            "name": "set",
                            "accept": {
                                "name": "reference",
                                "accept": {
                                    "name": "tag"
                                }
                            }
                        },
                        "required": true
                    },
                    {
                        "name": "priority",
                        "target": {
                            "name": "priority"
                        },
                        "required": false
                    },
                    {
                        "name": "due_date",
                        "target": {
                            "name": "date"
                        },
                        "required": false
                    }
                ]
            },
            {
                "id": "g52dedae-ca1d-24f5-8e84-d2a1f0e566f7",
                "name": "single-task",
                "nature": "concrete",
                "prototype": "task",
                "attributes": []
            },
            {
                "id": "ddeqd2e4-4da1-7of4-82e9-a1hf06d2e6f7",
                "name": "recurring-task",
                "nature": "concrete",
                "prototype": "task",
                "attributes": [
                    {
                        "name": "start",
                        "target": {
                            "name": "date"
                        },
                        "required": true
                    },
                    {
                        "name": "end",
                        "target": {
                            "name": "date"
                        },
                        "required": true
                    },
                    {
                        "name": "recurrence",
                        "target": {
                            "name": "number",
                            "constraint": {
                                "value": {
                                    "type": "range",
                                    "range": {
                                        "min": { "value": 1 },
                                        "max": { "value": 7 }
                                    }
                                }
                            }
                        },
                        "required": true
                }
            ]
        },
        {
            "id": "p3a693c6-6e13-4e2f-b39f-26707210ab66",
            "name": "priority",
            "nature": "derivative",
            "base": "string",
            "constraint": {
                "values": ["P1", "P2", "P3", "P4"]
            }
        },
        {
            "id": "c6pa3963-713e-2e2f-9fb3-263707d2b6p6",
            "name": "date",
            "nature": "derivative",
            "base": "string",
            "constraint": {
                "value": {
                    "type": "pattern",
                    "pattern": {
                        "insensitive": true,
                        "global": true,
                        "value": "^([12]\\d{3}-(0[1-9]|1[0-2])
-(0[1-9]|[12]\\d|3[01]))$"
                    }
                }
            }
        }
    ]
}
```

Listing B.1. Gentleman Todo concepts

B.2. Projection schema

```
{
    "concept": [
        {
            "id": "11g5bwb5-451d-344f-80d0-1e2138a59ec9",
            "name": "todo",
            "nature": "concrete",
                "attributes": [
                    {
                        "name": "title",
                        "target": { "name": "string" },
                        "unique": true,
                        "required": true
                    },
                    {
                        "name": "tags",
                        "target": {
                            "name": "set",
                            "accept": { "name": "tag" }
                        },
                        "required": true
                    },
                    {
                        "name": "tasks",
                        "target": {
                            "name": "set",
                            "accept": { "name": "task" },
                            "constraint": {
                                "cardinality": {
                                    "type": "range",
                                    "range": {
                                        "min": { "value": 1 }
                                    }
                                }
                            }
                        },
                        "required": true
                    }
                ]
            }
        ]
    }
```

Listing B.2. Gentleman Todo projections

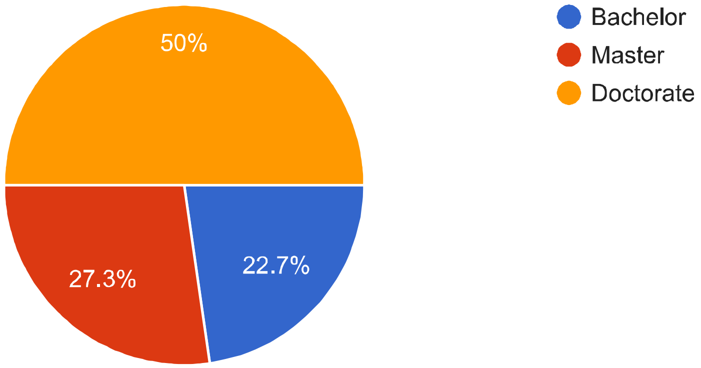
Appendix C

User study data

C.1. Participant profile

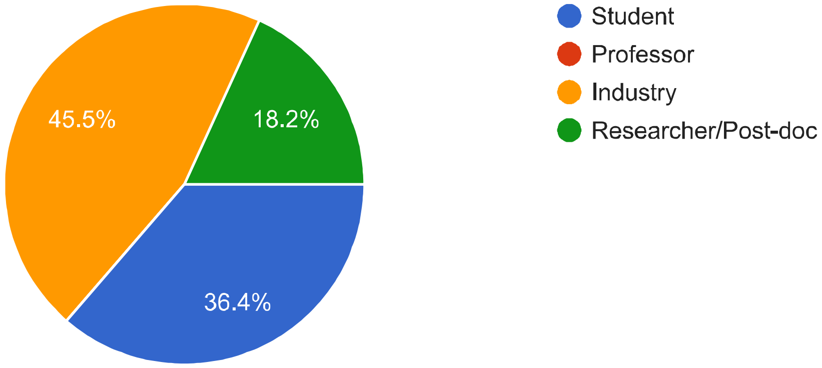
What is your academic level?

22 responses



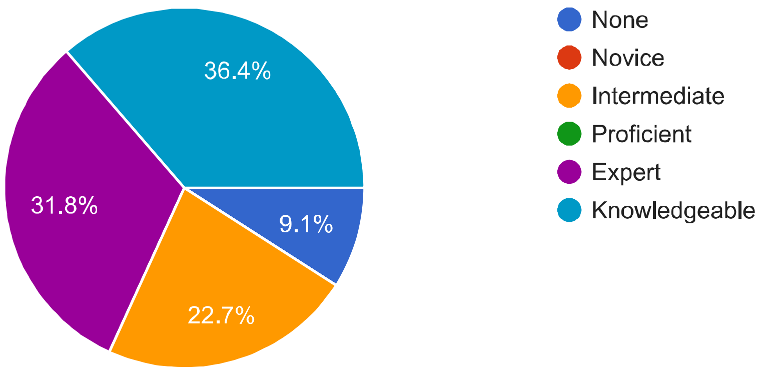
Which of the following describes you best for the purpose of this study?

22 responses



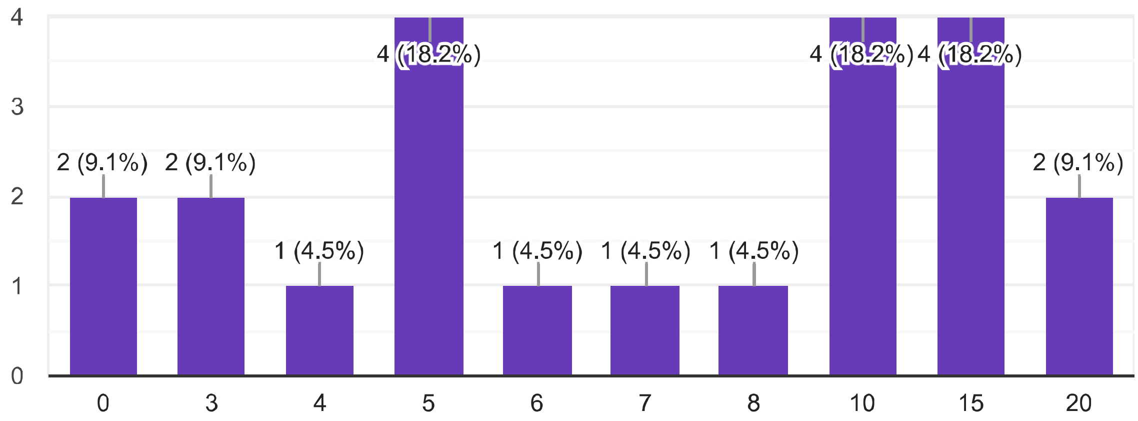
How would you rate your expertise in Computer Science/Software Engineering?

22 responses



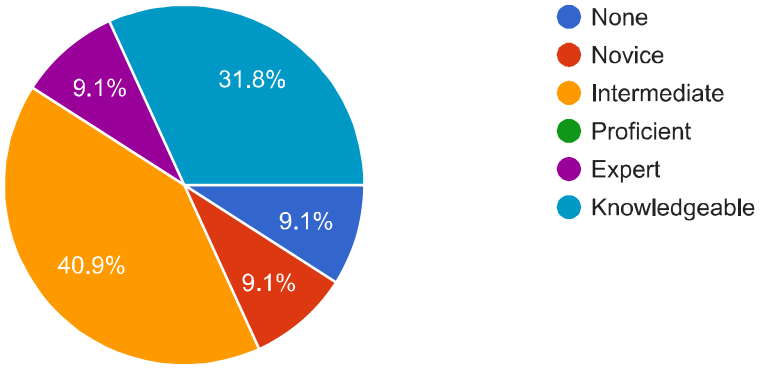
How many years of experience do you have in Computer Science/Software Engineering?

22 responses



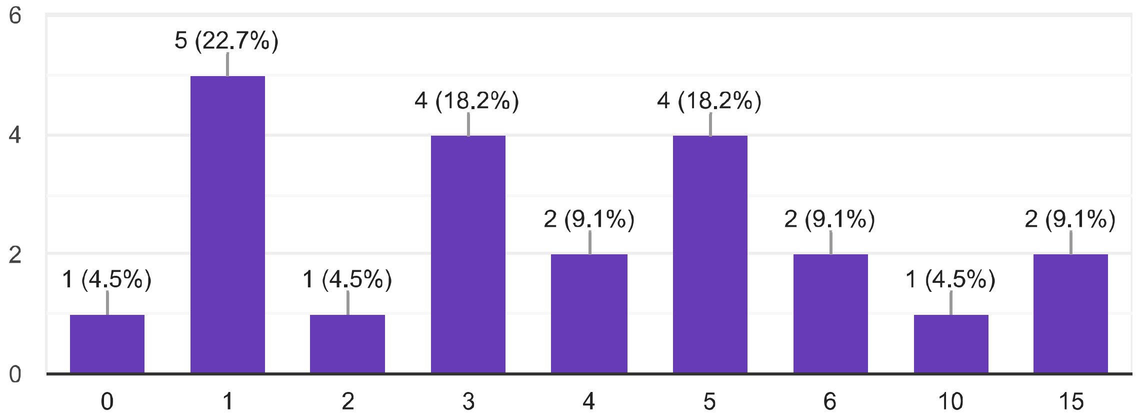
How would you rate your expertise in computer-based Design?

22 responses



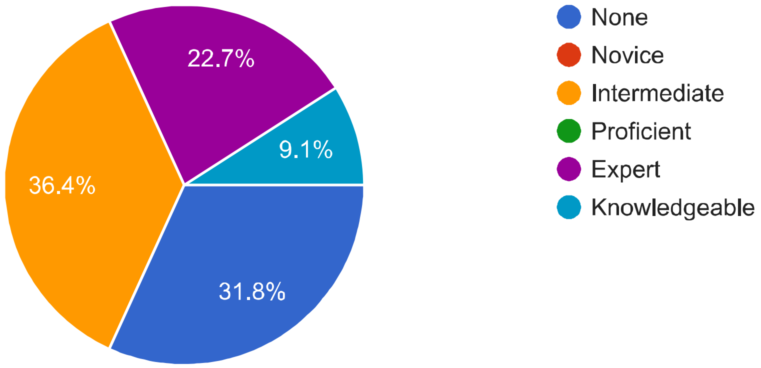
How many years of experience do you have in computer-based Design?

22 responses



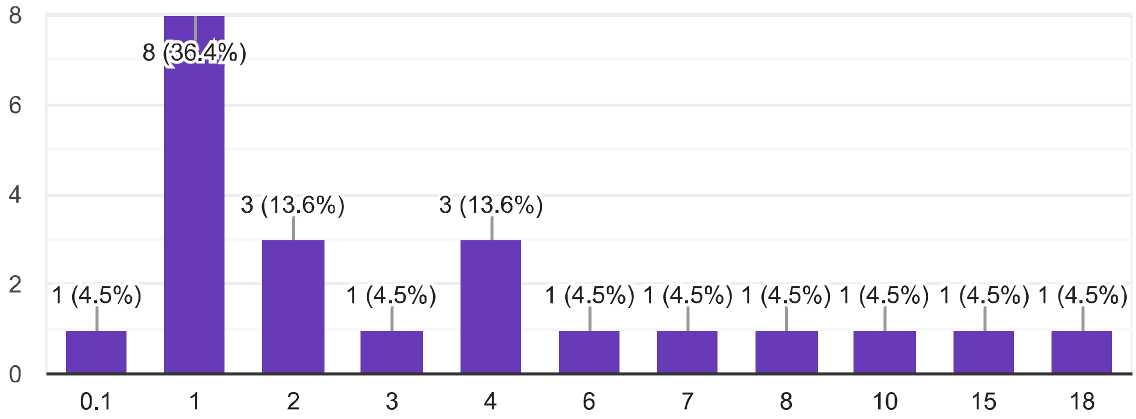
How would you rate your expertise in Model-Driven Engineering?

22 responses



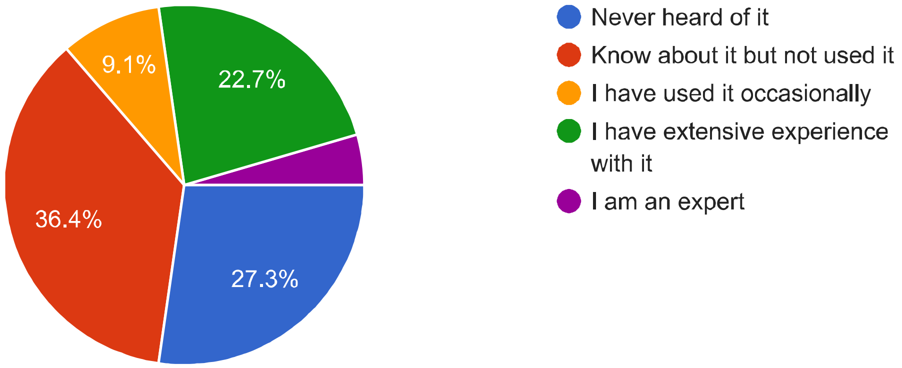
How many years of experience do you have in Model-Driven Engineering?

22 responses



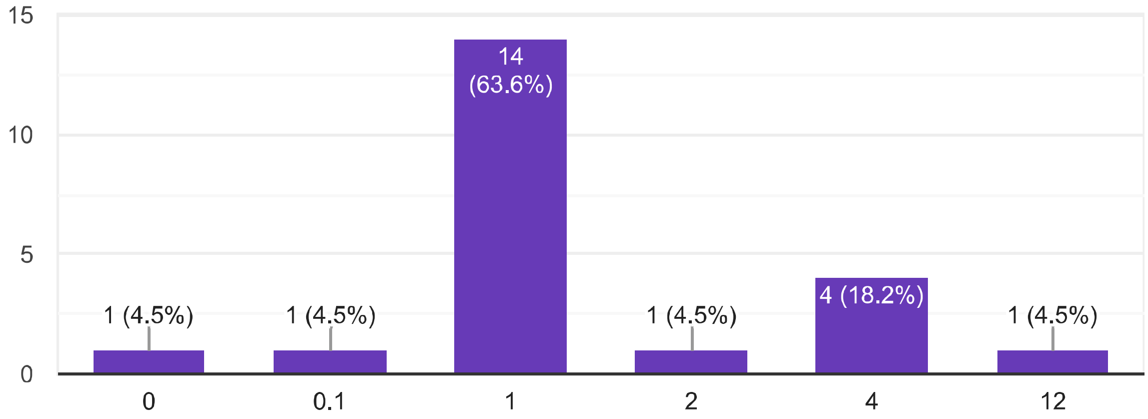
How familiar are you with Projectional editors?

22 responses



How many years of experience do you have with Projectional editing?

22 responses



C.2. Survey

Identification

Please complete this set of questions to better identify your background. note that all data will remain anonymous and individual data will not be shared.

* Required

1. What is your academic level? *

Mark only one oval.

○ Bachelor

○ Master

○ Doctorate

2. Which of the following describes you best for the purpose of this study? *

Mark only one oval.

○ Student

○ Professor

○ Industry

○ Researcher/Post-doc

3. How would you rate your expertise in Computer Science/Software Engineering? *

Mark only one oval.

○ None

○ Novice

○ Intermediate

○ Proficient

○ Expert

4. How many years of experience do you have in Computer Science/Software Engineering? *

____

5. How would you rate your expertise in computer-based Design? *

Mark only one oval.

○ None
○ Novice
○ Intermediate
○ Proficient
○ Expert

6. How many years of experience do you have in computer-based Design? *

____

7. How would you rate your expertise in Model-Driven Engineering? *

Mark only one oval.

○ None
○ Novice
○ Intermediate
○ Proficient
○ Expert

8. How many years of experience do you have in Model-Driven Engineering? *

____

9. How familiar are you with Projectional editors? *

Mark only one oval.

○ Never heard of it
○ Know about it but not used it
○ I have used it occasionally
○ I have extensive experience with it
○ I am an expert

10. How many years of experience do you have with Projectional editing? *

____

11. Which tool did you use first for the tasks? *

Mark only one oval.

○ Gentleman *Skip to question 12*
○ MPS

Modeling with Gentleman

Please answer the following questions regarding your experience using Gentleman.

12. It is easy to understand user interface of the editor *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

13. It is easy to remember the controls and commands of the editor *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

14. It is easy to know the state (values) of the model and whether it is valid *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

15. It is easy to navigate through elements of a model with the keyboard *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

16. It is easy to navigate through elements of a model with the mouse *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

17. It is easy to reuse values across different elements of the model *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

18. It is easy to know which element type (concept) is being focused on *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

19. It is easy to recover from a mistake *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

20. It is easy to know what actions are expected from me or available to me at any point in time *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

21. It is easy to execute the necessary steps to realize an action *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

22. Please share any other comments on your experience using Gentleman, if any.

Modeling with MPS

Please answer the following questions regarding your experience using MPS.

23. It is easy to understand user interface of the editor *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

24. It is easy to remember the controls and commands of the editor *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

25. It is easy to know the state (values) of the model and whether it is valid *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

26. It is easy to navigate through elements of a model with the keyboard *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

27. It is easy to navigate through elements of a model with the mouse *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

28. It is easy to reuse values across different elements of the model *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

29. It is easy to know which element type (concept) is being focused on *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

30. It is easy to recover from a mistake *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

31. It is easy to know what actions are expected from me or available to me at any point in time *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

32. It is easy to execute the necessary steps to realize an action *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

33. Please share any other comments on your experience using MPS, if any.

____

____

____

____

____

C.3. Results

C.3.1. Task 1 results

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Min
						Lower	Upper	
NB_Selection	Gentleman	22	27.73	5.147	1.097	25.45	30.01	20
	MPS	22	37.55	15.358	3.274	30.74	44.35	20
	Total	44	32.64	12.361	1.863	28.88	36.39	20
NB_Blank	Gentleman	22	1.32	1.555	0.332	0.63	2.01	0
	MPS	22	4.59	4.697	1.001	2.51	6.67	0
	Total	44	2.95	3.833	0.578	1.79	4.12	0
NB_Miss	Gentleman	22	0.27	0.703	0.150	-0.04	0.58	0
	MPS	22	0.73	0.883	0.188	0.34	1.12	0
	Total	44	0.50	0.821	0.124	0.25	0.75	0
NB_CxtChange	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0
	MPS	22	8.36	3.430	0.731	6.84	9.88	5
	Total	44	4.18	4.862	0.733	2.70	5.66	0
NB_Idle	Gentleman	22	1.59	1.681	0.358	0.85	2.34	0
	MPS	22	2.27	2.097	0.447	1.34	3.20	0
	Total	44	1.93	1.910	0.288	1.35	2.51	0
T_Idle	Gentleman	22	5.36	7.607	1.622	1.99	8.74	0
	MPS	22	12.32	14.875	3.171	5.72	18.91	0
	Total	44	8.84	12.194	1.838	5.13	12.55	0
NB_QA	Gentleman	22	0.27	0.550	0.117	0.03	0.52	0
	MPS	22	1.32	2.234	0.476	0.33	2.31	0
	Total	44	0.80	1.692	0.255	0.28	1.31	0
T_QA	Gentleman	22	1.50	3.474	0.741	-0.04	3.04	0
	MPS	22	12.27	26.767	5.707	0.40	24.14	0
	Total	44	6.89	19.634	2.960	0.92	12.86	0
T_Total	Gentleman	22	138.18	57.208	12.197	112.82	163.55	75
	MPS	22	227.05	124.065	26.451	172.04	282.05	80
	Total	44	182.61	105.525	15.908	150.53	214.70	75
RT_Success	Gentleman	22	1.00	0.000	0.000	1.00	1.00	1
	MPS	22	1.00	0.000	0.000	1.00	1.00	1
	Total	44	1.00	0.000	0.000	1.00	1.00	1
NB_Block	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0
	MPS	22	0.41	0.590	0.126	0.15	0.67	0
	Total	44	0.20	0.462	0.070	0.06	0.34	0
NB_Typos	Gentleman	22	0.59	0.959	0.204	0.17	1.02	0
	MPS	22	0.32	0.568	0.121	0.07	0.57	0
	Total	44	0.45	0.791	0.119	0.21	0.70	0

NB_DesignErr	Gentleman	22	0.05	0.213	0.045	-0.05	0.14	0
	MPS	22	0.09	0.426	0.091	-0.10	0.28	0
	Total	44	0.07	0.334	0.050	-0.03	0.17	0
NB_Bug	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0
	MPS	22	0.09	0.294	0.063	-0.04	0.22	0
	Total	44	0.05	0.211	0.032	-0.02	0.11	0
Detection	Gentleman	22	0.31818	0.476731	0.101639	0.10681	0.52955	0.000
	MPS	22	0.04545	0.213201	0.045455	-0.04907	0.13998	0.000
	Total	44	0.18182	0.390154	0.058818	0.06320	0.30044	0.000
RT_Recover	Gentleman	22	0.41	0.503	0.107	0.19	0.63	0
	MPS	22	0.23	0.429	0.091	0.04	0.42	0
	Total	44	0.32	0.471	0.071	0.17	0.46	0
T_Recover	Gentleman	22	3.64	6.673	1.423	0.68	6.59	0
	MPS	22	1.27	3.042	0.649	-0.08	2.62	0
	Total	44	2.45	5.263	0.793	0.85	4.05	0
task_velocity	Gentleman	22	0.2269	0.08541	0.01821	0.1891	0.2648	0.09
	MPS	22	0.1907	0.08216	0.01752	0.1543	0.2271	0.09
	Total	44	0.2088	0.08482	0.01279	0.1830	0.2346	0.09

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
NB_Selection	Between Groups	1060.364	1	1060.364	8.083	0.007
	Within Groups	5509.818	42	131.186		
	Total	6570.182	43			
NB_Blank	Between Groups	117.818	1	117.818	9.625	0.003
	Within Groups	514.091	42	12.240		
	Total	631.909	43			
NB_Miss	Between Groups	2.273	1	2.273	3.571	0.066
	Within Groups	26.727	42	0.636		
	Total	29.000	43			
NB_CxtChange	Between Groups	769.455	1	769.455	130.790	0.000
	Within Groups	247.091	42	5.883		
	Total	1016.545	43			
NB_Idle	Between Groups	5.114	1	5.114	1.416	0.241
	Within Groups	151.682	42	3.611		
	Total	156.795	43			
T_Idle	Between Groups	532.023	1	532.023	3.812	0.058
	Within Groups	5861.864	42	139.568		
	Total	6393.886	43			

NB_QA	Between Groups	12.023	1	12.023	4.544	0.039
	Within Groups	111.136	42	2.646		
	Total	123.159	43			
T_QA	Between Groups	1276.568	1	1276.568	3.504	0.068
	Within Groups	15299.864	42	364.282		
	Total	16576.432	43			
T_Total	Between Groups	86864.205	1	86864.205	9.308	0.004
	Within Groups	391960.227	42	9332.386		
	Total	478824.432	43			
RT_Success	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
NB_Block	Between Groups	1.841	1	1.841	10.565	0.002
	Within Groups	7.318	42	0.174		
	Total	9.159	43			
NB_Typos	Between Groups	0.818	1	0.818	1.317	0.258
	Within Groups	26.091	42	0.621		
	Total	26.909	43			
NB_DesignErr	Between Groups	0.023	1	0.023	0.200	0.657
	Within Groups	4.773	42	0.114		
	Total	4.795	43			
NB_Bug	Between Groups	0.091	1	0.091	2.100	0.155
	Within Groups	1.818	42	0.043		
	Total	1.909	43			
Detection	Between Groups	0.818	1	0.818	6.000	0.019
	Within Groups	5.727	42	0.136		
	Total	6.545	43			
RT_Recover	Between Groups	0.364	1	0.364	1.663	0.204
	Within Groups	9.182	42	0.219		
	Total	9.545	43			
T_Recover	Between Groups	61.455	1	61.455	2.285	0.138
	Within Groups	1129.455	42	26.892		
	Total	1190.909	43			
task_velocity	Between Groups	0.014	1	0.014	2.055	0.159
	Within Groups	0.295	42	0.007		
	Total	0.309	43			

Robust Tests of Equality of Means[b,c,d,e]

		Statistic[a]	df1	df2	Sig.

NB_Selection	Welch	8.083	1	25.659	0.009
	Brown-Forsythe	8.083	1	25.659	0.009
NB_Blank	Welch	9.625	1	25.548	0.005
	Brown-Forsythe	9.625	1	25.548	0.005
NB_Miss	Welch	3.571	1	39.985	0.066
	Brown-Forsythe	3.571	1	39.985	0.066
NB_CxtChange	Welch				
	Brown-Forsythe				
NB_Idle	Welch	1.416	1	40.097	0.241
	Brown-Forsythe	1.416	1	40.097	0.241
T_Idle	Welch	3.812	1	31.280	0.060
	Brown-Forsythe	3.812	1	31.280	0.060
NB_QA	Welch	4.544	1	23.542	0.044
	Brown-Forsythe	4.544	1	23.542	0.044
T_QA	Welch	3.504	1	21.707	0.075
	Brown-Forsythe	3.504	1	21.707	0.075
T_Total	Welch	9.308	1	29.544	0.005
	Brown-Forsythe	9.308	1	29.544	0.005
RT_Success	Welch				
	Brown-Forsythe				
NB_Block	Welch				
	Brown-Forsythe				
NB_Typos	Welch	1.317	1	34.113	0.259
	Brown-Forsythe	1.317	1	34.113	0.259
NB_DesignErr	Welch	0.200	1	30.882	0.658
	Brown-Forsythe	0.200	1	30.882	0.658
NB_Bug	Welch				
	Brown-Forsythe				
Detection	Welch	6.000	1	29.077	0.021
	Brown-Forsythe	6.000	1	29.077	0.021
RT_Recover	Welch	1.663	1	40.972	0.204
	Brown-Forsythe	1.663	1	40.972	0.204
T_Recover	Welch	2.285	1	29.368	0.141
	Brown-Forsythe	2.285	1	29.368	0.141
task_velocity	Welch	2.055	1	41.937	0.159
	Brown-Forsythe	2.055	1	41.937	0.159

a. Asymptotically F distributed.

b. Robust tests of equality of means cannot be performed for NB_CxtChange because at least one group has 0 variance.

c. Robust tests of equality of means cannot be performed for RT_Success because at least one group has 0 variance.

d. Robust tests of equality of means cannot be performed for NB_Block because at least one group has 0 variance.

e. Robust tests of equality of means cannot be performed for NB_Bug because at least one group has 0 variance.

Measures of Association

	Eta	Eta Squared
NB_Selection * tool	0.402	0.161
NB_Blank * tool	0.432	0.186
NB_Miss * tool	0.280	0.078
NB_CxtChange * tool	0.870	0.757
NB_Idle * tool	0.181	0.033
T_Idle * tool	0.288	0.083
NB_QA * tool	0.312	0.098
T_QA * tool	0.278	0.077
T_Total * tool	0.426	0.181
NB_Block * tool	0.448	0.201
NB_Typos * tool	0.174	0.030
NB_DesignErr * tool	0.069	0.005
NB_Bug * tool	0.218	0.048
Detection * tool	0.354	0.125
RT_Recover * tool	0.195	0.038
T_Recover * tool	0.227	0.052
task_velocity * tool	0.216	0.047

C.3.2. Task 2 results

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Min	Max
						Lower	Upper		
NB_Selection	Gentleman	22	49.14	12.124	2.585	43.76	54.51	28	76
	MPS	22	55.45	20.694	4.412	46.28	64.63	30	98
	Total	44	52.30	17.063	2.572	47.11	57.48	28	98
NB_Blank	Gentleman	22	2.86	2.765	0.590	1.64	4.09	0	9
	MPS	22	5.64	4.499	0.959	3.64	7.63	0	18
	Total	44	4.25	3.948	0.595	3.05	5.45	0	18
NB_Miss	Gentleman	22	1.36	1.620	0.345	0.65	2.08	0	7
	MPS	22	1.09	1.269	0.271	0.53	1.65	0	4
	Total	44	1.23	1.445	0.218	0.79	1.67	0	7
NB_CxtChange	Gentleman	22	0.32	1.129	0.241	-0.18	0.82	0	5
	MPS	22	9.95	4.370	0.932	8.02	11.89	3	20
	Total	44	5.14	5.805	0.875	3.37	6.90	0	20
NB_Idle	Gentleman	22	2.23	1.343	0.286	1.63	2.82	0	6
	MPS	22	2.41	1.501	0.320	1.74	3.07	0	6
	Total	44	2.32	1.410	0.213	1.89	2.75	0	6
T_Idle	Gentleman	22	9.45	7.042	1.501	6.33	12.58	0	26
	MPS	22	11.86	9.321	1.987	7.73	16.00	0	35
	Total	44	10.66	8.255	1.244	8.15	13.17	0	35
NB_QA	Gentleman	22	1.50	1.144	0.244	0.99	2.01	0	4
	MPS	22	1.95	1.988	0.424	1.07	2.84	0	6
	Total	44	1.73	1.619	0.244	1.24	2.22	0	6
T_QA	Gentleman	22	11.45	9.689	2.066	7.16	15.75	0	30
	MPS	22	15.36	19.456	4.148	6.74	23.99	0	60
	Total	44	13.41	15.317	2.309	8.75	18.07	0	60
T_Total	Gentleman	22	295.00	107.593	22.939	247.30	342.70	115	540
	MPS	22	358.18	143.822	30.663	294.41	421.95	160	750
	Total	44	326.59	129.524	19.526	287.21	365.97	115	750
RT_Success	Gentleman	22	1.00	0.000	0.000	1.00	1.00	1	1
	MPS	22	1.00	0.000	0.000	1.00	1.00	1	1
	Total	44	1.00	0.000	0.000	1.00	1.00	1	1
NB_Block	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0	0
	MPS	22	0.32	0.646	0.138	0.03	0.60	0	2
	Total	44	0.16	0.479	0.072	0.01	0.30	0	2
NB_Typos	Gentleman	22	0.18	0.395	0.084	0.01	0.36	0	1
	MPS	22	0.09	0.426	0.091	-0.10	0.28	0	2
	Total	44	0.14	0.409	0.062	0.01	0.26	0	2

NB_DesignErr	Gentleman	22	0.09	0.294	0.063	-0.04	0.22	0	1
	MPS	22	0.23	0.752	0.160	-0.11	0.56	0	3
	Total	44	0.16	0.568	0.086	-0.01	0.33	0	3
NB_Bug	Gentleman	22	0.05	0.213	0.045	-0.05	0.14	0	1
	MPS	22	0.00	0.000	0.000	0.00	0.00	0	0
	Total	44	0.02	0.151	0.023	-0.02	0.07	0	1
Detection	Gentleman	22	0.27273	0.455842	0.097186	0.07062	0.47484	0.000	1.000
	MPS	22	0.04545	0.213201	0.045455	-0.04907	0.13998	0.000	1.000
	Total	44	0.15909	0.369989	0.055778	0.04660	0.27158	0.000	1.000
RT_Recover	Gentleman	22	0.32	0.477	0.102	0.11	0.53	0	1
	MPS	22	0.14	0.351	0.075	-0.02	0.29	0	1
	Total	44	0.23	0.424	0.064	0.10	0.36	0	1
T_Recover	Gentleman	22	2.55	4.339	0.925	0.62	4.47	0	12
	MPS	22	1.14	3.299	0.703	-0.33	2.60	0	14
	Total	44	1.84	3.876	0.584	0.66	3.02	0	14
task_velocity	Gentleman	22	0.1813	0.06008	0.01281	0.1547	0.2080	0.10	0.30
	MPS	22	0.1658	0.06041	0.01288	0.1391	0.1926	0.09	0.32
	Total	44	0.1736	0.06005	0.00905	0.1553	0.1918	0.09	0.32

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
NB_Selection	Between Groups	439.114	1	439.114	1.527	0.223
	Within Groups	12080.045	42	287.620		
	Total	12519.159	43			
NB_Blank	Between Groups	84.568	1	84.568	6.064	0.018
	Within Groups	585.682	42	13.945		
	Total	670.250	43			
NB_Miss	Between Groups	0.818	1	0.818	0.387	0.538
	Within Groups	88.909	42	2.117		
	Total	89.727	43			
NB_CxtChange	Between Groups	1021.455	1	1021.455	100.300	0.000
	Within Groups	427.727	42	10.184		
	Total	1449.182	43			
NB_Idle	Between Groups	0.364	1	0.364	0.179	0.674
	Within Groups	85.182	42	2.028		
	Total	85.545	43			
T_Idle	Between Groups	63.841	1	63.841	0.936	0.339
	Within Groups	2866.045	42	68.239		
	Total	2929.886	43			

NB_QA	Between Groups	2.273	1	2.273	0.864	0.358
	Within Groups	110.455	42	2.630		
	Total	112.727	43			
T_QA	Between Groups	168.091	1	168.091	0.712	0.404
	Within Groups	9920.545	42	236.203		
	Total	10088.636	43			
T_Total	Between Groups	43911.364	1	43911.364	2.722	0.106
	Within Groups	677477.273	42	16130.411		
	Total	721388.636	43			
RT_Success	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
NB_Block	Between Groups	1.114	1	1.114	5.332	0.026
	Within Groups	8.773	42	0.209		
	Total	9.886	43			
NB_Typos	Between Groups	0.091	1	0.091	0.538	0.467
	Within Groups	7.091	42	0.169		
	Total	7.182	43			
NB_DesignErr	Between Groups	0.205	1	0.205	0.628	0.433
	Within Groups	13.682	42	0.326		
	Total	13.886	43			
NB_Bug	Between Groups	0.023	1	0.023	1.000	0.323
	Within Groups	0.955	42	0.023		
	Total	0.977	43			
Detection	Between Groups	0.568	1	0.568	4.487	0.040
	Within Groups	5.318	42	0.127		
	Total	5.886	43			
RT_Recover	Between Groups	0.364	1	0.364	2.074	0.157
	Within Groups	7.364	42	0.175		
	Total	7.727	43			
T_Recover	Between Groups	21.841	1	21.841	1.470	0.232
	Within Groups	624.045	42	14.858		
	Total	645.886	43			
task_velocity	Between Groups	0.003	1	0.003	0.727	0.399
	Within Groups	0.152	42	0.004		
	Total	0.155	43			

Robust Tests of Equality of Means[b,c,d]

	Statistic[a]	df1	df2	Sig.

NB_Selection	Welch	1.527	1	33.896	0.225
	Brown-Forsythe	1.527	1	33.896	0.225
NB_Blank	Welch	6.064	1	34.885	0.019
	Brown-Forsythe	6.064	1	34.885	0.019
NB_Miss	Welch	0.387	1	39.726	0.538
	Brown-Forsythe	0.387	1	39.726	0.538
NB_CxtChange	Welch	100.300	1	23.792	0.000
	Brown-Forsythe	100.300	1	23.792	0.000
NB_Idle	Welch	0.179	1	41.489	0.674
	Brown-Forsythe	0.179	1	41.489	0.674
T_Idle	Welch	0.936	1	39.082	0.339
	Brown-Forsythe	0.936	1	39.082	0.339
NB_QA	Welch	0.864	1	33.545	0.359
	Brown-Forsythe	0.864	1	33.545	0.359
T_QA	Welch	0.712	1	30.813	0.405
	Brown-Forsythe	0.712	1	30.813	0.405
T_Total	Welch	2.722	1	38.899	0.107
	Brown-Forsythe	2.722	1	38.899	0.107
RT_Success	Welch				
	Brown-Forsythe				
NB_Block	Welch				
	Brown-Forsythe				
NB_Typos	Welch	0.538	1	41.753	0.467
	Brown-Forsythe	0.538	1	41.753	0.467
NB_DesignErr	Welch	0.628	1	27.289	0.435
	Brown-Forsythe	0.628	1	27.289	0.435
NB_Bug	Welch				
	Brown-Forsythe				
Detection	Welch	4.487	1	29.768	0.043
	Brown-Forsythe	4.487	1	29.768	0.043
RT_Recover	Welch	2.074	1	38.610	0.158
	Brown-Forsythe	2.074	1	38.610	0.158
T_Recover	Welch	1.470	1	39.197	0.233
	Brown-Forsythe	1.470	1	39.197	0.233
task_velocity	Welch	0.727	1	41.999	0.399
	Brown-Forsythe	0.727	1	41.999	0.399

a. Asymptotically F distributed.

b. Robust tests of equality of means cannot be performed for RT_Success because at least one group has 0 variance.

c. Robust tests of equality of means cannot be performed for NB_Block because at least one group has 0 variance.

d. Robust tests of equality of means cannot be performed for NB_Bug because at least one group has 0 variance.

Measures of Association

	Eta	Eta Squared
NB_Selection * tool	0.187	0.035
NB_Blank * tool	0.355	0.126
NB_Miss * tool	0.095	0.009
NB_CxtChange * tool	0.840	0.705
NB_Idle * tool	0.065	0.004
T_Idle * tool	0.148	0.022
NB_QA * tool	0.142	0.020
T_QA * tool	0.129	0.017
T_Total * tool	0.247	0.061
NB_Block * tool	0.336	0.113
NB_Typos * tool	0.113	0.013
NB_DesignErr * tool	0.121	0.015
NB_Bug * tool	0.152	0.023
Detection * tool	0.311	0.097
RT_Recover * tool	0.217	0.047
T_Recover * tool	0.184	0.034
task_velocity * tool	0.130	0.017

C.3.3. Task 3 results

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Min
						Lower	Upper	
NB_Selection	Gentleman	22	16.55	4.512	0.962	14.55	18.55	11
	MPS	22	16.68	5.818	1.240	14.10	19.26	10
	Total	44	16.61	5.145	0.776	15.05	18.18	10
NB_Blank	Gentleman	22	0.77	1.066	0.227	0.30	1.25	0
	MPS	22	2.32	2.033	0.433	1.42	3.22	0
	Total	44	1.55	1.784	0.269	1.00	2.09	0
NB_Miss	Gentleman	22	0.18	0.501	0.107	-0.04	0.40	0
	MPS	22	0.18	0.501	0.107	-0.04	0.40	0
	Total	44	0.18	0.495	0.075	0.03	0.33	0
NB_CxtChange	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0
	MPS	22	0.00	0.000	0.000	0.00	0.00	0
	Total	44	0.00	0.000	0.000	0.00	0.00	0
NB_Idle	Gentleman	22	0.95	1.090	0.232	0.47	1.44	0
	MPS	22	1.05	1.174	0.250	0.52	1.57	0
	Total	44	1.00	1.121	0.169	0.66	1.34	0
T_Idle	Gentleman	22	3.82	5.422	1.156	1.41	6.22	0
	MPS	22	4.45	7.576	1.615	1.10	7.81	0
	Total	44	4.14	6.519	0.983	2.15	6.12	0
NB_QA	Gentleman	22	0.09	0.426	0.091	-0.10	0.28	0
	MPS	22	0.23	0.429	0.091	0.04	0.42	0
	Total	44	0.16	0.428	0.065	0.03	0.29	0
T_QA	Gentleman	22	1.82	8.528	1.818	-1.96	5.60	0
	MPS	22	1.36	2.904	0.619	0.08	2.65	0
	Total	44	1.59	6.300	0.950	-0.32	3.51	0
T_Total	Gentleman	22	187.50	78.327	16.699	152.77	222.23	75
	MPS	22	212.95	92.243	19.666	172.06	253.85	80
	Total	44	200.23	85.542	12.896	174.22	226.23	75
RT_Success	Gentleman	22	1.00	0.000	0.000	1.00	1.00	1
	MPS	22	1.00	0.000	0.000	1.00	1.00	1
	Total	44	1.00	0.000	0.000	1.00	1.00	1
NB_Block	Gentleman	22	0.00	0.000	0.000	0.00	0.00	0
	MPS	22	0.05	0.213	0.045	-0.05	0.14	0
	Total	44	0.02	0.151	0.023	-0.02	0.07	0
NB_Typos	Gentleman	22	4.36	0.848	0.181	3.99	4.74	3
	MPS	22	4.00	0.926	0.197	3.59	4.41	2
	Total	44	4.18	0.896	0.135	3.91	4.45	2

NB_DesignErr	Gentleman	22	3.36	0.848	0.181	2.99	3.74	2
	MPS	22	3.45	0.800	0.171	3.10	3.81	2
	Total	44	3.41	0.816	0.123	3.16	3.66	2
NB_Bug	Gentleman	22	0.09	0.294	0.063	-0.04	0.22	0
	MPS	22	0.00	0.000	0.000	0.00	0.00	0
	Total	44	0.05	0.211	0.032	-0.02	0.11	0
Detection	Gentleman	22	1.00000	0.000000	0.000000	1.00000	1.00000	1.000
	MPS	22	1.00000	0.000000	0.000000	1.00000	1.00000	1.000
	Total	44	1.00000	0.000000	0.000000	1.00000	1.00000	1.000
RT_Recover	Gentleman	22	1.00	0.000	0.000	1.00	1.00	1
	MPS	22	1.00	0.000	0.000	1.00	1.00	1
	Total	44	1.00	0.000	0.000	1.00	1.00	1
T_Recover	Gentleman	22	86.59	36.167	7.711	70.56	102.63	40
	MPS	22	93.41	48.926	10.431	71.72	115.10	35
	Total	44	90.00	42.659	6.431	77.03	102.97	35
task_velocity	Gentleman	22	0.1056	0.06238	0.01330	0.0779	0.1332	0.05
	MPS	22	0.0938	0.05700	0.01215	0.0686	0.1191	0.03
	Total	44	0.0997	0.05935	0.00895	0.0817	0.1178	0.03

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
NB_Selection	Between Groups	0.205	1	0.205	0.008	0.931
	Within Groups	1138.227	42	27.101		
	Total	1138.432	43			
NB_Blank	Between Groups	26.273	1	26.273	9.974	0.003
	Within Groups	110.636	42	2.634		
	Total	136.909	43			
NB_Miss	Between Groups	0.000	1	0.000	0.000	1.000
	Within Groups	10.545	42	0.251		
	Total	10.545	43			
NB_CxtChange	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
NB_Idle	Between Groups	0.091	1	0.091	0.071	0.791
	Within Groups	53.909	42	1.284		
	Total	54.000	43			
T_Idle	Between Groups	4.455	1	4.455	0.103	0.750
	Within Groups	1822.727	42	43.398		
	Total	1827.182	43			

NB_QA	Between Groups	0.205	1	0.205	1.118	0.296
	Within Groups	7.682	42	0.183		
	Total	7.886	43			
T_QA	Between Groups	2.273	1	2.273	0.056	0.814
	Within Groups	1704.364	42	40.580		
	Total	1706.636	43			
T_Total	Between Groups	7127.273	1	7127.273	0.973	0.329
	Within Groups	307520.455	42	7321.916		
	Total	314647.727	43			
RT_Success	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
NB_Block	Between Groups	0.023	1	0.023	1.000	0.323
	Within Groups	0.955	42	0.023		
	Total	0.977	43			
NB_Typos	Between Groups	1.455	1	1.455	1.846	0.181
	Within Groups	33.091	42	0.788		
	Total	34.545	43			
NB_DesignErr	Between Groups	0.091	1	0.091	0.134	0.716
	Within Groups	28.545	42	0.680		
	Total	28.636	43			
NB_Bug	Between Groups	0.091	1	0.091	2.100	0.155
	Within Groups	1.818	42	0.043		
	Total	1.909	43			
Detection	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
RT_Recover	Between Groups	0.000	1	0.000		
	Within Groups	0.000	42	0.000		
	Total	0.000	43			
T_Recover	Between Groups	511.364	1	511.364	0.276	0.602
	Within Groups	77738.636	42	1850.920		
	Total	78250.000	43			
task_velocity	Between Groups	0.002	1	0.002	0.425	0.518
	Within Groups	0.150	42	0.004		
	Total	0.151	43			

Robust Tests of Equality of Means[b,c,d,e,f,g]

		Statistic[a]	df1	df2	Sig.

NB_Selection	Welch	0.008	1	39.550	0.931
	Brown-Forsythe	0.008	1	39.550	0.931
NB_Blank	Welch	9.974	1	31.738	0.003
	Brown-Forsythe	9.974	1	31.738	0.003
NB_Miss	Welch	0.000	1	42.000	1.000
	Brown-Forsythe	0.000	1	42.000	1.000
NB_CxtChange	Welch				
	Brown-Forsythe				
NB_Idle	Welch	0.071	1	41.770	0.791
	Brown-Forsythe	0.071	1	41.770	0.791
T_Idle	Welch	0.103	1	38.039	0.750
	Brown-Forsythe	0.103	1	38.039	0.750
NB_QA	Welch	1.118	1	41.999	0.296
	Brown-Forsythe	1.118	1	41.999	0.296
T_QA	Welch	0.056	1	25.805	0.815
	Brown-Forsythe	0.056	1	25.805	0.815
T_Total	Welch	0.973	1	40.925	0.330
	Brown-Forsythe	0.973	1	40.925	0.330
RT_Success	Welch				
	Brown-Forsythe				
NB_Block	Welch				
	Brown-Forsythe				
NB_Typos	Welch	1.846	1	41.678	0.182
	Brown-Forsythe	1.846	1	41.678	0.182
NB_DesignErr	Welch	0.134	1	41.862	0.716
	Brown-Forsythe	0.134	1	41.862	0.716
NB_Bug	Welch				
	Brown-Forsythe				
Detection	Welch				
	Brown-Forsythe				
RT_Recover	Welch				
	Brown-Forsythe				
T_Recover	Welch	0.276	1	38.673	0.602
	Brown-Forsythe	0.276	1	38.673	0.602
task_velocity	Welch	0.425	1	41.664	0.518
	Brown-Forsythe	0.425	1	41.664	0.518

a. Asymptotically F distributed.

b. Robust tests of equality of means cannot be performed for NB_CxtChange because at least one group has 0 variance.

c. Robust tests of equality of means cannot be performed for RT_Success because at least one group has 0 variance.

d. Robust tests of equality of means cannot be performed for NB_Block because at least one group has 0 variance.

e. Robust tests of equality of means cannot be performed for NB_Bug because at least one group has 0 variance.

f. Robust tests of equality of means cannot be performed for Detection because at least one group has 0 variance.

g. Robust tests of equality of means cannot be performed for RT_Recover because at least one group has 0 variance.

Measures of Association

	Eta	Eta Squared
NB_Selection * tool	0.013	0.000
NB_Blank * tool	0.438	0.192
NB_Miss * tool	0.000	0.000
NB_Idle * tool	0.041	0.002
T_Idle * tool	0.049	0.002
NB_QA * tool	0.161	0.026
T_QA * tool	0.036	0.001
T_Total * tool	0.151	0.023
NB_Block * tool	0.152	0.023
NB_Typos * tool	0.205	0.042
NB_DesignErr * tool	0.056	0.003
NB_Bug * tool	0.218	0.048
T_Recover * tool	0.081	0.007
task_velocity * tool	0.100	0.010

C.3.4. Survey results

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Min	Max
						Lower	Upper		
ui	Gentleman	22	4.73	0.456	0.097	4.53	4.93	4	5
	MPS	22	3.95	0.844	0.180	3.58	4.33	2	5
	Total	44	4.34	0.776	0.117	4.11	4.58	2	5
control	Gentleman	22	4.77	0.429	0.091	4.58	4.96	4	5
	MPS	22	4.14	0.990	0.211	3.70	4.58	1	5
	Total	44	4.45	0.820	0.124	4.21	4.70	1	5
state	Gentleman	22	4.73	0.456	0.097	4.53	4.93	4	5
	MPS	22	4.27	0.767	0.164	3.93	4.61	2	5
	Total	44	4.50	0.665	0.100	4.30	4.70	2	5
navkey	Gentleman	10	4.30	0.823	0.260	3.71	4.89	3	5
	MPS	21	4.24	0.768	0.168	3.89	4.59	3	5
	Total	31	4.26	0.773	0.139	3.97	4.54	3	5
navmouse	Gentleman	22	4.86	0.351	0.075	4.71	5.02	4	5
	MPS	22	4.05	0.950	0.203	3.62	4.47	2	5
	Total	44	4.45	0.820	0.124	4.21	4.70	2	5
reuseval	Gentleman	22	4.64	0.581	0.124	4.38	4.89	3	5
	MPS	22	4.09	0.750	0.160	3.76	4.42	3	5
	Total	44	4.36	0.718	0.108	4.15	4.58	3	5
focus	Gentleman	22	4.59	0.796	0.170	4.24	4.94	3	5
	MPS	22	4.09	0.921	0.196	3.68	4.50	2	5
	Total	44	4.34	0.888	0.134	4.07	4.61	2	5
recover	Gentleman	22	4.73	0.456	0.097	4.53	4.93	4	5
	MPS	22	4.41	0.734	0.157	4.08	4.73	3	5
	Total	44	4.57	0.625	0.094	4.38	4.76	3	5
actions	Gentleman	22	4.73	0.456	0.097	4.53	4.93	4	5
	MPS	22	3.55	1.143	0.244	3.04	4.05	1	5
	Total	44	4.14	1.047	0.158	3.82	4.45	1	5
execution	Gentleman	22	4.77	0.528	0.113	4.54	5.01	3	5
	MPS	22	4.05	0.785	0.167	3.70	4.39	3	5
	Total	44	4.41	0.757	0.114	4.18	4.64	3	5

Test of Homogeneity of Variances

		Levene Statistic	df1	df2	Sig.
ui	Based on Mean	2.797	1	42	0.102
	Based on Median	4.004	1	42	0.052

	Based on Median and with adjusted df	4.004	1	39.475	0.052
	Based on trimmed mean	2.318	1	42	0.135
control	Based on Mean	2.711	1	42	0.107
	Based on Median	3.556	1	42	0.066
	Based on Median and with adjusted df	3.556	1	32.238	0.068
	Based on trimmed mean	3.872	1	42	0.056
state	Based on Mean	3.319	1	42	0.076
	Based on Median	2.908	1	42	0.096
	Based on Median and with adjusted df	2.908	1	39.311	0.096
	Based on trimmed mean	4.280	1	42	0.045
navkey	Based on Mean	0.107	1	29	0.746
	Based on Median	0.196	1	29	0.661
	Based on Median and with adjusted df	0.196	1	28.419	0.661
	Based on trimmed mean	0.092	1	29	0.764
navmouse	Based on Mean	19.903	1	42	0.000
	Based on Median	22.129	1	42	0.000
	Based on Median and with adjusted df	22.129	1	36.527	0.000
	Based on trimmed mean	22.758	1	42	0.000
reuseval	Based on Mean	0.514	1	42	0.477
	Based on Median	1.217	1	42	0.276
	Based on Median and with adjusted df	1.217	1	41.297	0.276
	Based on trimmed mean	0.844	1	42	0.364
focus	Based on Mean	0.030	1	42	0.862
	Based on Median	1.065	1	42	0.308
	Based on Median and with adjusted df	1.065	1	40.557	0.308
	Based on trimmed mean	0.236	1	42	0.629
recover	Based on Mean	9.227	1	42	0.004
	Based on Median	2.983	1	42	0.092
	Based on Median and with adjusted df	2.983	1	35.097	0.093
	Based on trimmed mean	8.723	1	42	0.005
actions	Based on Mean	16.697	1	42	0.000
	Based on Median	10.290	1	42	0.003
	Based on Median and with adjusted df	10.290	1	33.060	0.003

	Based on trimmed mean	16.153	1	42	0.000
execution	Based on Mean	3.347	1	42	0.074
	Based on Median	5.463	1	42	0.024
	Based on Median and with adjusted df	5.463	1	41.900	0.024
	Based on trimmed mean	4.514	1	42	0.040

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
ui	Between Groups	6.568	1	6.568	14.280	0.000
	Within Groups	19.318	42	0.460		
	Total	25.886	43			
control	Between Groups	4.455	1	4.455	7.651	0.008
	Within Groups	24.455	42	0.582		
	Total	28.909	43			
state	Between Groups	2.273	1	2.273	5.707	0.021
	Within Groups	16.727	42	0.398		
	Total	19.000	43			
navkey	Between Groups	0.026	1	0.026	0.042	0.839
	Within Groups	17.910	29	0.618		
	Total	17.935	30			
navmouse	Between Groups	7.364	1	7.364	14.354	0.000
	Within Groups	21.545	42	0.513		
	Total	28.909	43			
reuseval	Between Groups	3.273	1	3.273	7.269	0.010
	Within Groups	18.909	42	0.450		
	Total	22.182	43			
focus	Between Groups	2.750	1	2.750	3.709	0.061
	Within Groups	31.136	42	0.741		
	Total	33.886	43			
recover	Between Groups	1.114	1	1.114	2.983	0.092
	Within Groups	15.682	42	0.373		
	Total	16.795	43			
actions	Between Groups	15.364	1	15.364	20.280	0.000
	Within Groups	31.818	42	0.758		
	Total	47.182	43			
execution	Between Groups	5.818	1	5.818	12.986	0.001
	Within Groups	18.818	42	0.448		
	Total	24.636	43			

Robust Tests of Equality of Means

		Statistic[a]	df1	df2	Sig.
ui	Welch	14.280	1	32.294	0.001
	Brown-Forsythe	14.280	1	32.294	0.001
control	Welch	7.651	1	28.613	0.010
	Brown-Forsythe	7.651	1	28.613	0.010
state	Welch	5.707	1	34.182	0.023
	Brown-Forsythe	5.707	1	34.182	0.023
navkey	Welch	0.040	1	16.721	0.844
	Brown-Forsythe	0.040	1	16.721	0.844
navmouse	Welch	14.354	1	26.636	0.001
	Brown-Forsythe	14.354	1	26.636	0.001
reuseval	Welch	7.269	1	39.529	0.010
	Brown-Forsythe	7.269	1	39.529	0.010
focus	Welch	3.709	1	41.141	0.061
	Brown-Forsythe	3.709	1	41.141	0.061
recover	Welch	2.983	1	35.097	0.093
	Brown-Forsythe	2.983	1	35.097	0.093
actions	Welch	20.280	1	27.511	0.000
	Brown-Forsythe	20.280	1	27.511	0.000
execution	Welch	12.986	1	36.778	0.001
	Brown-Forsythe	12.986	1	36.778	0.001

a. Asymptotically F distributed.

Measures of Association

	Eta	Eta Squared
ui * tool	0.504	0.254
control * tool	0.393	0.154
state * tool	0.346	0.120
navkey * tool	0.038	0.001
navmouse * tool	0.505	0.255
reuseval * tool	0.384	0.148
focus * tool	0.285	0.081
recover * tool	0.257	0.066
actions * tool	0.571	0.326
execution * tool	0.486	0.236