Sara Russo Rosa

Licenciada em Ciência e Engenharia Informática

A Rule-based Engine to support a Framework for the Experimental Validation of Domain Specific Languages

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador:	Prof. Dr. Vasco Miguel Moreira Amaral, Prof. Auxiliar, Universidade Nova de Lisboa
Co-orientador:	Prof. Dr. Miguel Carlos Pacheco Afonso Goulão, Prof. Auxiliar, Universidade Nova de Lisboa

Júri

Presidente:	Prof. Dr. Carlos Augusto Isaac Piló Viegas Damásio
Arguente:	Prof. Dr. João Carlos Pascoal de Faria
Vogal:	Prof. Dr. Vasco Miguel Moreira Amaral

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

A Rule-based Engine to support a Framework for the Experimental Validation of Domain Specific Languages

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template “unlthesis” [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Em memória dos meus avôs,
Fernando e João.

Acknowledgements

Começo por agradecer aos meus orientadores, Professores Miguel Goulão e Vasco Amaral, e à investigadora Ankica Barišić por todo o apoio, sugestões e críticas, e pela disponibilidade que demonstraram ao longo da realização desta dissertação. Sem dúvida que com a sua ajuda o trabalho tornou-se mais fácil.

Aos meus colegas que participaram nos estudos desta dissertação.

À Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, pela formação de excelência oferecida aos alunos. E em especial ao Departamento de Informática, que nestes anos me acolheu tão bem e se tornou na minha segunda casa.

Aos meus companheiros nesta aventura, em especial à Katia Duarte, ao Pedro Simão, ao Tomás Rogeiro, ao Rui Teodósio e ao André Correia.

À Jessica Alegria, pelos anos de amizade. À Mariana Lourenço e ao José Sintra pelos momentos de diversão.

Um agradecimento muito especial ao Pedro Dias, o meu principal companheiro nesta aventura, por todo o carinho e paciência ao longo destes anos.

Por último, quero agradecer aos meus pais e a minha avó, por me terem encaminhado ao longo destes anos, pela sua constante preocupação, e principalmente, por sempre acreditarem no meu sucesso.

Abstract

Software systems are widely used in people daily routines and responsibilities, therefore, systems need to be developed rapidly and efficiently. Domain specific languages (DSLs) are languages that are applied to a specific application domain. Since DSLs provide notations and constructs adapted to a particular domain, they offer gains in expressiveness and ease of use when compared with general-purpose languages (GPLs). Therefore, one of the most important steps in the *Software Language Engineering* is the evaluation of the languages produced, with the end-users, since the risk of building inappropriate languages, that often do not fit the end users, may decrease productivity. Although DSLs evaluation is one of the most important steps in development process, *Software Language Engineers* tend to relax the experimental validation of their products due to several reasons like costs (time, means, money, the number of people required, etc.) and required know-how associated with it. The lack of systematic approaches and guidelines to evaluate DSLs, and a comprehensive set of tools may explain this shortcoming in the current state of practice. The Usability Driven DSL development with USE-ME (USE-ME) approach, developed in NOVA-LINCS, *"promotes the quality in use of DSLs by building a framework that leverages usability as a main concern"*. The feedback of the pilot studies was that despite the approach was "more or less easy" to understand it was not easy to model, since "there were too many steps to follow" and the framework did not provide a "guided cycle". So, in order to improve the system usability and the quality of the models produced with USE-ME, we developed a new version of the framework with *validation rules* implemented with *Eclipse Validation Language* (EVL) that guide, suggest and validate the *Software Language Engineer* actions throughout the development process. The *validation rules* were designed in such a way that the tool educates the user about the process, so that the user makes the best decision regarding his DSL evaluation.

We performed two experiments, with different goals. The main goals of the first one was to analyse the effect of *validation rules* on the USE-ME framework, with respect to their impact on the *System Usability Scale*, and on the *Model Correctness* of USE-ME models. We analysed the results and we found evidences of improvements on the *System Usability Scale*, and on the *Model Correctness* of models, brought by the addition of the rules. The second experiment was conducted with a research team from Ege University, in Turkey. The main goal of this experiment was to perform a guided evaluation on a DSL related with Multi-Agent Systems, SEA-ML. Since the number of participants was low we cannot draw conclusions regarding this experiment.

Despite the significant results from the first experiment further evaluation on the *new* version of the framework is necessary, this time, with more experienced users and with more complex exercises. With this new experiment, we can compare the results and improve the USE-ME framework.

Keywords: Domain Specific Languages, Languages Evaluation, Software Language Engineering, Tool Support, Usability, Validation Rules, Workflow-oriented tool, Rule-based System

Resumo

Os sistemas de software são amplamente utilizados nas rotinas e nas responsabilidades diárias das pessoas, desta forma, os sistemas têm de ser desenvolvidos de forma rápida e eficiente. As Linguagens de Dominio Especifico (LDEs) são linguagens que são aplicadas a um domínio específico. Uma vez que as LDEs fornecem notações e construções adaptadas a um domínio específico, oferecem ganhos na expressividade e facilidade de uso quando comparados com as linguagens de uso geral. Uma das etapas mais importantes na *Engenharia de Software e Linguagens* é a avaliação das linguagens produzidas com os utilizadores finais, uma vez que o risco de desenvolver linguagens inadequadas, que muitas vezes não se adaptam aos utilizadores finais, pode diminuir produtividade. Embora a avaliação das LDEs seja uma das etapas mais importantes no processo de desenvolvimento, os *Engenheiros de Linguagens de Software* tendem a relaxar a validação experimental dos seus produtos devido a vários motivos (tempo, meios, dinheiro, número de pessoas necessárias, etc.) e à falta de conhecimento necessário associado a estas avaliações. A falta de abordagens e diretrizes sistemáticas para avaliar as LDEs, e a falta de ferramentas que suportem a avaliação podem explicar o atual estado de prática.

O Usability Driven DSL development with USE-ME (USE-ME), desenvolvido na NOVA-LINCS, *"promove a qualidade no uso de LDEs, através de uma framework que eleva a usabilidade como principal artefacto"* [Bar+17]. O feedback dos estudos-piloto, realizado ao USE-ME, foi que apesar da abordagem ser *"mais ou menos fácil"* de entender não era fácil de modelar, já que *"haviam muitos passos a seguir"* e a *framework* não fornecia um *"ciclo de desenvolvimento guiado"*. De forma a melhorar a usabilidade do sistema e a qualidade dos modelos produzidos com o USE-ME, desenvolvemos uma nova versão da *framework*, com *regras de validação* expressas no *Eclipse Validation Language* (EVL), que permitem guiar, sugerir e validar as ações dos *Engenheiros de Linguagens de Software* ao longo do processo de desenvolvimento. As *regras de validação* foram desenhadas de forma a que a ferramenta eduque o utilizador sobre o processo, para que quando seja necessário tomar decisões o utilizador escolha a mais adequada em relação à sua avaliação LDE.

Realizamos duas experiências, com diferentes objetivos. Os principais objetivos da primeira experiência foram analisar o efeito das regras de validação no USE-ME, em relação ao seu impacto na *Escala de Usabilidade do Sistema* e na *Correção dos Modelos*, produzidos com o USE-ME. Analisamos os resultados e encontramos evidências de melhorias tanto na *Escala de Usabilidade do Sistema* como na *Correção dos Modelos*, influencidos pela adição das regras. A segunda experiência foi conduzida com uma equipa de investigação da Ege University, na Turquia. O principal objetivo desta experiência foi a realização de uma avaliação guiada a uma LDE relacionada com Sistemas Multi-Agentes, SEA-ML. Como o número de participantes foi baixo, não podemos tirar conclusões desta experiência.

Apesar de os resultados da primeira experiência serem significativos, é necessária uma avaliação mais aprofundada da nova versão da *framework*, com utilizadores mais experientes e com exercícios mais complexos. Para que, possamos comparar os resultados e melhorar a estrutura do USE-ME.

Palavras-chave: Linguagem de Domínio Específico, Avaliação de Linguagens, Engenharia de Software e Linguagens, Suporte à ferramenta, Usabilidade, Regras de validação, Ferramenta orientada ao workflow, Sistema baseado em regras

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this chapter, we introduce the work carried out in this dissertation by making an overview description of the context (section 1.1), highlighting and motivating the problem to be solved (section 1.2). We also explain our main goal (section 1.3) and the contributions of this dissertation (section 1.4). The chapter ends with a presentation on the structure of this document (section 1.5).

1.1 Context and Description

Software systems are widely used in people daily routines and responsibilities, therefore, systems need to be developed rapidly and efficiently, in order to match the users mental model of the problems. Domain specific languages (DSLs) are languages that are applied to a specific application domain. These languages are designed to bridge the gap between the Problem Domain (essential concepts, domain knowledge, techniques, and paradigms) and the Solution Domain (technical space, middleware, platforms and programming languages) [Vö+13]. Since DSLs provide notations and constructs adapted to a particular domain, they offer gains in expressiveness and ease of use when compared with general-purpose languages (GPLs) [Mer+05].

Software Language Engineering (SLE) *"is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages"* [Rad+90]. One of the most important steps in the SLE is the evaluation of the languages produced with the end users, since the risk of building inappropriate languages, that often do not fit the end users, may decrease productivity or increase the language maintenance costs [Bar+12a]. Although DSLs evaluation is one of the most important steps in development process, Software Language Engineers tend to relax the experimental validation of their products due to several reasons like costs (time, means, money, the number of people required, etc.) and required know-how associated with it. The lack of systematic approaches and guidelines to evaluate DSLs, and a comprehensive set of tools may explain this shortcoming in the current state of practice. These concerns need to be addressed from early stages of the DSL development cycle in order to enable practitioners to perform timely evaluations, rather than designing the complete DSL before the implementation.

1.2 Motivation

A well designed DSL is based on a thorough understanding of the application domain, since these languages are expected to be used by groups that are more familiar with domain concepts. So, there is a need to involve language stakeholders (*i.e.* Software Language Engineers, Domain Experts and End Users) in the process, as it allows to increase end users productivity and product quality. As we mentioned in the previous section 1.1, *Software Language Engineers* lack the validation of their languages with the end users. Based on the lack of systematic approaches and guidelines to evaluate DSLs, Ankica et al. developed Usability Driven DSL development with USE-ME (USE-ME) in NOVA-LINCS. The goal of this approach is to "promote quality in use of DSLs by building a framework that leverages usability as a main concern" [Bar+17]. The DSL development with USE-ME is composed by 6 phases performed by the SLE. During the iterate and incremental development process the SLE defines: the DSL context of use (Context Modeling phase), sets the DSL goals (Goal Modeling phase), organizes usability experiments (Evaluation Modeling phase), defines interaction tasks (Interaction Modeling phase), develops survey questionnaires (Survey Modeling phase), and collects the data from the interaction and survey experiments (Report Modeling phase). USE-ME was tested with students, during a DSL course, in which each student played the role of a *Software Language Engineer* and developed USE-ME models in order to assess their own DSL. The feedback of the pilot study was that despite USE-ME was "more or less easy" to understand the approach was not easy to model since "there were too many steps to follow" and the framework did not provide a "guided cycle". So, in order to improve the user experience with the USE-ME framework we developed a new version of the USE-ME tool and in the next section 1.3 we present its main objectives.

1.3 Objectives

As mentioned in the previous section 1.2, the participants of the pilot studies found that USE-ME workflow was complex, so they suggested that the tool should offer "some kind of guidance". For that purpose, we developed a new version of the USE-ME framework with *validation rules* that guides, suggests and validates the *Software Language Engineer* actions throughout the development process. Some of the USE-ME activities are mandatory (*e.g.* the creation of the context modeling phase is mandatory) and some are suggestions (*e.g.*

the creation of the interaction and survey modeling phase is a suggestion, however, one of these modeling activites must be created) so we need to provide to the *SLE* the right information regarding the activity so that the *SLE* could choose what is the best option regarding his DSL. Because of that, we decided to add to the *validation rules* information about the process so that the user could learn while using the tool. Other concern with the *original* USE-ME version was that not all the language syntactic rules could be expressed in the language meta-model (*i.e.*), so users could create USE-ME models that were not correct. With the addition of *validation rules* we can verify the models produced with USE-ME and alert the user for errors/warnings.

In this dissertation, we evaluate the *System Usability Scale* and the *Model Correctness* of both USE-ME versions, original (*i.e.* without validation rules) and new (*i.e.* with validation rules). The main objective is to understand which version suits end users better.

1.4 Key Contributions

The main contribution of this dissertation is an updated version of the USE-ME framework that allows to support the *SLE* in one of the most important steps of the *Software Language Engineering* process, which is the language (*i.e.* DSL) evaluation. Supporting the *SLE* during the evaluation process is crucial to mitigate the risk of building languages that often do not fit the end users. With the *new* version of the USE-ME framework we are able to guide, suggest and validate the *SLE* actions through the development process. We also present other important contributions, such as:

- a study and an analysis on experimental evaluations of Domain Specific Languages, and a comparison between these approaches;
- a study and an analysis on wizards/tools that are concerned with user guidance, the quality of the models produced, increasing productivity of the end user, and to decrease the maintenance costs;
- a discussion on the implementation alternatives, a solution and an architecture for the prototype;
- an experiment planning, conduction, and analysis regarding the usability (measured with *SUS*) and the *Model Correctness* of the USE-ME and the *Model Correctness* of the USE-ME models.

1.5 Structure

This document is organised, excluding the current chapter, in the following way:

- **Chapter 2** - Background: in this chapter, we introduce the basic notions and concepts that will be used throughout this dissertation. First, it is crucial to understand the notion of Domain Specific Languages (DSLs) (section 2.1), the main difference between DSLs and General-Purpose Languages (GPLs) (section 2.1.1), the DSLs stakeholders (section 2.1.2) and its development cycle (section 2.1.3). In this chapter, we also talk about Model-Driven Development (section 2.2), Human-computer interaction (section 2.3), Usability (section 2.3.1), and Eclipse Modelling Tools (section 2.4);

- **Chapter 3** - Usability Driven DSL development with USE-ME: in this chapter, we introduce the USE-ME framework. In section **??** we elaborate a state-of-the-art regarding DSLs evaluation. In section 3.2, we present the USE-ME framework, its domain concepts, meta-models and activity specifications. Also, in this section we present the results regarding the USE-ME pilot studies;

- **Chapter 4** - Related Work: in this chapter, we present some tools related with this dissertation: Tool Support for Agent Development using the Prometheus Methodology (section 4.1), J-PRiM: A Java Tool for a Process Reengineering i* Methodology (section 4.2), PETIC Wizard Proposal: a Software Tool for Support PETIC Methodology (section 4.3), A Qualitative Study on User Guidance Capabilities in Product Configuration Tools (section 4.4), Business process modeling with continuous validation (section 4.5), Rule-based detection of inconsistency in UML models (section 4.6), and Cognitive support, UML adherence, and XMI interchange in Argo/UML (section 4.7). All these approaches are concerned with user guidance, models quality, increase productivity and decrease maintenance costs. For each approach we provide a description, present the stakeholders involved, present its implementation, and explain how it could be applied to the USE-ME framework;

- **Chapter 5** - Validation Rules Implementation: in this chapter, we discuss the implementation alternatives and the problems that were found during the USE-ME pilot studies (section 5.1). We also describe the solution that we found to be the most appropriate to deal with the problems described (section 5.2). Then, we explain in more detail how we implemented the validation rules and how we integrated them on the USE-ME framework (section 5.3). We finish this chapter by providing an use case scenario (section 5.4);

- **Chapter 6** - Evaluation: in this chapter, we report the experiment conducted (section 6.1), including its goals (section 6.1.1), the tasks proposed (section 6.1.2), the experiment materials (section 6.1.3), the participants (section 6.1.4), the hypotheses (section 6.1.5), the design (section 6.1.6), the procedure (section 6.1.7), and the analysis (section 6.1.8). The results from the *SUS* and the *Model Correctness* assessment are then analysed, in section 6.2, and discussed in section 6.3. In section 6.3.2 we examined the validity of the process;

- **Chapter 7** - Conclusions: in this chapter, we sum up the main contributions of this dissertation 7.1, the limitations of our solution 7.2, and we propose future work in order to improve the USE-ME framework 7.3.

Chapter 2

Background

In this chapter, we introduce the basic notions and concepts that will be used throughout this dissertation. First, it is crucial to understand the notion of Domain Specific Languages (DSLs) (section 2.1), the main difference between DSLs and General-Purpose Languages (GPLs) (section 2.1.1), the DSLs stakeholders (section 2.1.2) and its development cycle (section 2.1.3). In this chapter, we also talk about Model-Driven Development (section 2.2), Human-computer interaction (section 2.3), Usability (section 2.3.1), and Eclipse Modelling Tools (section 2.4).

2.1 Domain Specific Languages

Domain-specific languages (DSLs) are *"programming languages or executable specification languages that offer, through appropriate notation and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"* [VD+00]. These programming languages offer gains in expressiveness and ease of use when compared with General-purpose languages (GPLs) since they apply to a specific domain [Bar+12b].

DSLs are designed to narrow the gap between the Problem Domain (essential concepts, domain knowledge, techniques, and paradigms) and the Solution Domain (technical space, middleware, platforms and programming languages) (Figure 2.1) [Vö+13]. These types of languages are usually expressed as text or graphic diagrams, but they can also be represented as matrices, tables, forms, or trees [KP09].

By offering domain abstractions and semantics in a more readily apparent form, DSLs allow experts in the domain to work directly with domain concepts [KP09].

A well designed DSL is based on a thorough understanding of the application domain, and this allows to increase end users productivity [Bar+12b].

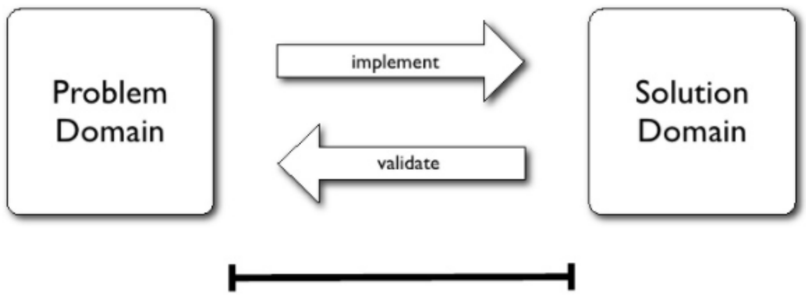


Figure 2.1: Gap between the Problem Domain and the Solution Domain, taken from [Viste].

2.1.1 DSLs versus GPLs

General-purpose languages (GPLs) are programming languages designed to be used in a wide variety of application domains. Unlike DSLs, these languages are not specialized for a particular domain. GPLs are used by people that have high knowledge of technical and computational concepts, while DSLs are expected to be used by groups that are more familiar with the domain concepts (*e.g.* experts from physics, chemistry, finance, management, etc.) [Fow10] [Bar+17].

DSLs offer several advantages when compared to GPLs [Mer+05]:

- allow to hide complexity;
- improve end users productivity;
- promote better product quality;
- are more amenable to verifications;
- increase data longevity (as independent abstractions, models are migratable);
- act as communication tools (*i.e. Domain Experts* themselves may understand, communicate, validate, modify, and often even develop DSL programs);
- enhance quality, productivity, reliability, maintainability, portability and reusability;
- allow validation at the domain level.

However, DSLs also have some disadvantages when compared to a GPLs [Mer+05]:

- require that users learn a new language (that has limited applicability) *versus* using a general language;
- have higher design, implementation, and maintenance costs;

- scope is more difficult to maintain;
- are more likely to loose processor efficiency;
- are harder to modify;
- are more difficult to integrate with other components of the IT system, as compared to integrating with GPLs;
- code examples are harder to find.

When developing a new language the DSLs advantages and disadvantages should be compared to an existing baseline solution (often, implemented with a GPL), in order to make the best decision regarding the stakeholder requirements.

2.1.2 DSLs Stakeholders

There are three main stakeholders, each one with different background and knowledge, that are involved in the DSL development process:

- **Language Engineer:** is a professional that is an expert in the creation of software languages. The language engineer is responsible for managing implementation priorities, design the software language and making the language functional. To sum up, language engineers are involved in specification, implementation, and evaluation of the language [Kle08]. Language Engineers work with Domain Experts to determine abstractions, notations and constraints in order to capture domain knowledge;
- **DSL End-User or Domain User:** is the person that is going to use the language developed [Kle08]. During the DSL development domain users should be involved in the process and can propose changes in the application specifications. Not involving these users in the process may result in failure to adopt the DSL, if it does not fit the target audience [Vö+13];
- **Domain Expert:** is a person that has a deep knowledge of the domain. In spite of commonly not having a strong background in software development, domain experts work closely with language engineers in the language definition. Domain experts are responsible for managing the system goals and iterations [Vö+13].

2.1.3 DSLs Development Cycle

A well designed DSL is hard to build since it requires both domain knowledge and language development expertise, and a few people have both [Mer+05]. Software Language Engineering (SLE) *"is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages"* [Rad+90].

According to Mernik et al. the DSL development cycle consists of five development phases: decision, domain analysis, design, implementation and deployment [Mer+05]. Barišić et al. added one more phase to DSL development cycle: evaluation, before the deployment phase (Figure 2.2) [Bar+12b].

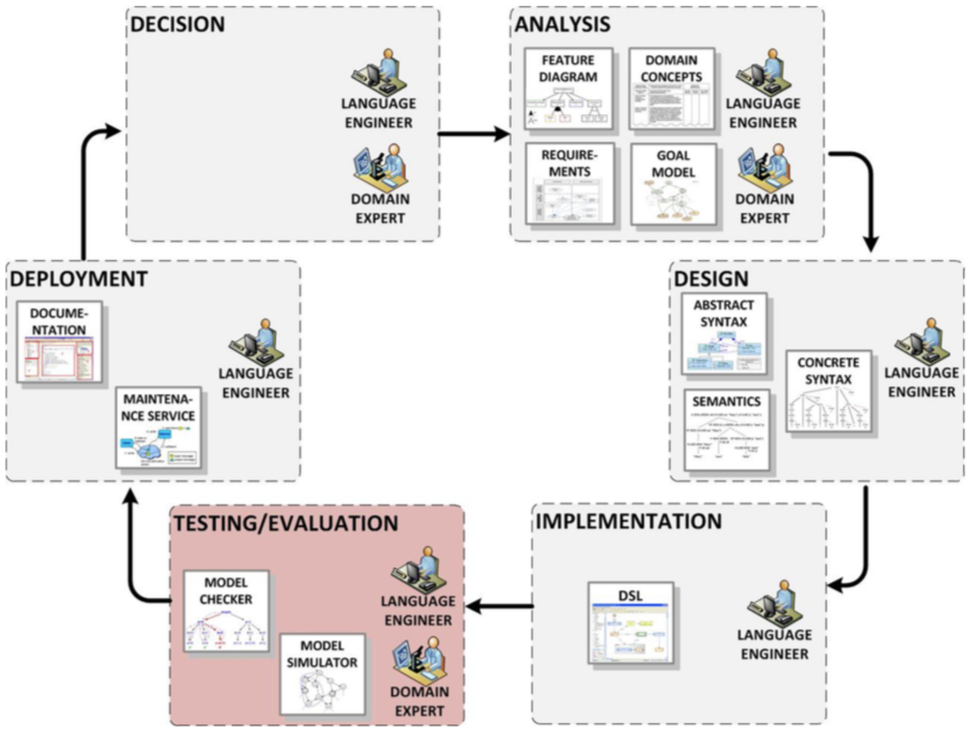


Figure 2.2: DSL life cycle, taken from [Bar+17].

A typical SLE process starts with the **decision** phase. It corresponds to the "When?", *When should a DSL be developed?*, while the remaining phases are the "How?" parts, *How to implement a DSL?*. In the beginning of the DSL development the decision to develop a new one or to reuse an existing one should be considered, by the stakeholders (*i.e.* Domain Experts and Language Engineers), since developing a DSL may imply more expenses and/or maintenance in the future [Mer+05].

After the **decision** phase comes the **analysis** phase. In this phase, the problem is identified and the knowledge on the domain is collected. During this phase Domain Experts help Language Engineers to define the domain concepts, the feature models, the functional and technical requirements, and the goal model for the language.. The output of this phase is the domain model. To sum up, the domain model represents [Mer+05]:

- the domain definition where the scope of the problem is identified;
- domain terminology (vocabulary, ontology);
- description of domain concepts;

- feature models describing the commonalities and variabilities of domain concepts as well as their interdependencies.

Usually the variabilities indicate what information is required to specify a system, while commonalities define, with a set of common operations and primitives, the execution model.

After the **analysis** phase comes the **design** phase. In this phase the Language Engineers define the language abstract syntax (*i.e.* the meta-model), the model representations, and the production/composition rules [Bar+17]. Therefore, the language semantics is defined. Also in this phase, the DSL relationship with other languages, and the formal nature of the design description are identified [Mer+05]. As already mentioned in the **decision** phase, a DSL can be designed from scratch or based on a existing one.

In the **implementation** phase the most suitable implementation approach (*e.g.* interpreter, compiler, preprocessing, embedding, extensible compiler/interpreter, COTS, hybrid) should be chosen [Mer+05] [Bar+17]. Also in this phase the developers produce model checkers and simulators that will help modelers to validate the models [Bar+17].

Barišić et al. proposed a new phase for the DSL development, which is the **evaluation** phase. In this DSL development phase the verifications (*i.e.* testing if the right functionality is provided by the DSL) and the validations (*i.e.* testing if the DSL is right for its users) are executed [Bar+17]. This phase helps to mitigate one of the biggest problems in software engineering, which is the software comprehension [Bar+17].

After the **evaluation** phase comes the **deployment** phase, where the DSL and the applications built with it are used [Mer+05]. Also in this phase the DSL documentation is delivered [Bar+17]. In this phase the developers and the domain experts use the DSL developed to specify models, which are implemented with one of the implementation patterns.

Visser recommends that DSLs should be developed incrementally, with an inductive approach, in contrast to designing the complete DSL before implementation, because the DSL introduces abstractions that allow to capture a set of common programming patterns in software development for a particular domain [Vis08]. Developing DSLs iteratively mitigates the risk of failure, since an iterative process produces useful DSLs for sub-domains early on [Vis08].

2.2 Model-Driven Development

Model-Driven Development (MDD) is a style of software development that is an alternative to the traditional style of programming (Figure 2.3). Stahl et al. described modelling as a important tool in engineering, since it allows engineers to create abstractions when analysing and/or designing systems [SV06]. A model does not have a specific meaning, and can only be understood if it is combined with an interpretation [SV06].

In software projects usually the models and the code are not directly connected, so models often become obsolete due to software evolution actions where only the code is updated, due to time constraints. The MDD proposes to solve this problem with techniques that include model-driven requirements engineering, design, code generation from models, testing, software evolution, among others [KT08] [WC99].

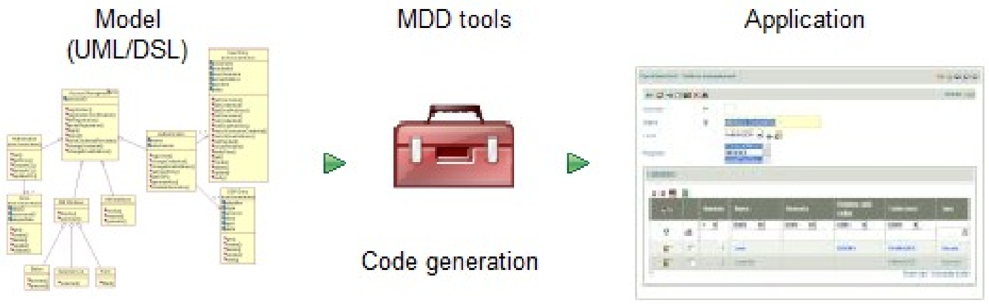


Figure 2.3: MDD software development, taken from [Voxte].

DSLs can be (and often are) built with the MDD approach, since MDD allows to create modelling abstractions close to the problem domain. MDD techniques and tools are seen as a viable approach for dealing with accidental complexity of the solution [Bar+17]. MDD also supports the code generation required to implement the specifications created with the DSL. In the DSL development the modeler has to be aware of the meaning of the language elements in order to create and transform models. Therefore, it is important that a DSL is well-documented and adopts concepts of the problem domain, in order to be clear to the modeler.

2.3 Human–computer interaction

Human Computer Interaction (HCI) *"is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them"* [Sin+10]. From a Human-Computer Interaction perspective, when developing a system, there are two main terms that should be considered [Sin+10]:

- **Functionality**: is defined by the set of actions or services that a system provides to its users;
- **Usability**: is the degree by which the system can be used efficiently and effectively to accomplish the user goals.

Nowadays, humans are surrounded by computers, so the way they interact plays a very important role. HCI major concern is improving the interactions between users and computers, by minimizing the barrier between the humans cognitive model of what they want to achieve, and computers understanding of the users task [Sin+10].

2.3.1 Usability

In the field of Human Computer Interaction, there are two widely recognized definitions of usability: Jacob Nielsen usability attributes and the International Organization for Standardization (ISO) standard concerning usability. Nielsen divides usability in five attributes (Figure 2.4) [Nie94]:

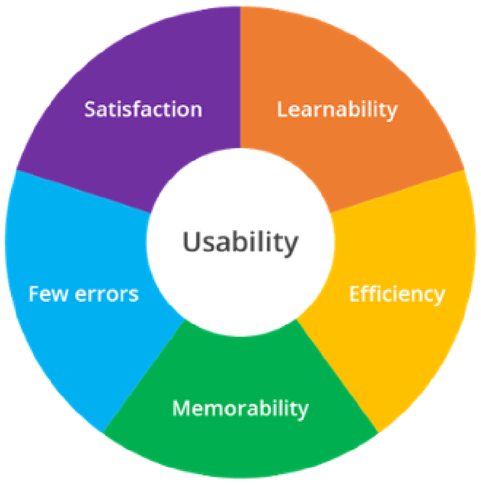


Figure 2.4: Jacob Nielsen usability attributes, taken from [Gyrte].

- **Learnability**: the system should be easy to learn. It means that a user must be able to learn how to use a system as quickly and as easily as possible. However, different learning times are acceptable, depending on the type of system. If a system is intended to be used by advanced users, the learning curve can be longer;
- **Efficiency**: the system should be efficient to use, so that once the user has learned how to use the system, a high level of productivity should be achieved. There are some users who do not need to learn to use a system fully, however: they feel satisfied when they have learned the basic functionalities;
- **Memorability**: when casual users return to the system, after some time without using it, they should easily remember the system functions without having to learn everything all over again;
- **Errors**: the system should be clear so that the users make as few errors as possible, during the use of the system, when they make errors, the system should support easy recovery from them, and as result the system will have a low error rate. Catastrophic errors must not occur.
- **Satisfaction**: the system should be pleasant to use, so that users are satisfied when using it, as it affects the users motivation and thus the effectiveness of use. The

system satisfaction can be evaluated, for example, through user questionnaires as we are going to see on section 2.3.1.1.

A user-friendly interface (*e.g.* a website or software application) design that is easy-to-learn, supports users tasks and goals efficiently and effectively, is satisfying and engaging to use. Usability also depends on a number of factors including how well the functionality adapts to user needs, how well the application flow adapts to user tasks, and how well the application response adapts to user expectations [Firte]. From both final user and developers point of view, usability is crucial since it can cost time and effort, and also can determine the success or failure of a software system. Software with poor usability can reduce the productivity to a level of performance worse than without the system. Also when applied to DSLs, usability has an impact on the achieved productivity of the DSL end users [Bar+12a].

The ISO 9241-11 standard defines usability as *"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"* [924te]. To sum up, usability is about:

- **Effectiveness**: the accuracy and completeness with which specified users can achieve specified goals in particular environments;
- **Efficiency**: the resources expended in relation to the accuracy and completeness of goals achieved;
- **Satisfaction**: the comfort and acceptability of the work system to its users and other people affected by its use.

The difference between these approaches is their scope. Nielsen refers to the usability of the product, in a particular context of use, while the ISO definition is in terms of the results of using the product.

2.3.1.1 System Usability Scale

The System Usability Scale (SUS) is a simple, ten-item scale, that allows to *"quick and dirty"* assess the usability of a given product or service [Bro+96] [Ban+08]. SUS is a *Likert scale* that can be applied to a wide range of interface technologies, since it is technology agnostic [Ban+08].

Each item is scored on a 5-point scale of strength of agreement (Figure 2.5), where 1-point means *strongly disagree* and 5-point means *strongly agree*. If a participant feels that (s)he cannot respond to a particular item, (s)he should mark the centre point of the scale. The SUS questionnaire contains the following items:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.

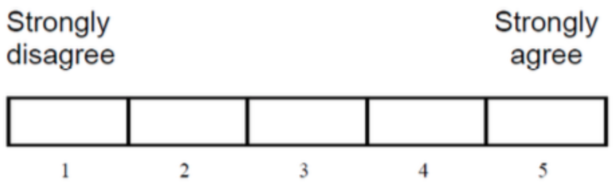


Figure 2.5: SUS *Likert scale*, adapted from [Asste].

3. I thought the system was easy to use.
4. I think I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The SUS scale is used after the participant has had an opportunity to use the system being evaluated, but before any discussion [Bro+96]. Participants should be asked to record their response to each item, rather than thinking about items for a long time [Bro+96].

2.3.1.2 Scoring SUS

Each item score contribution will range from 0 to 4. For items 1,3,5,7 and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10 the score contribution is 5 minus the scale position. Then the sum of scores is multiplied by 2.5, in order to obtain the overall value of the system usability [Bro+96]. The SUS final score can range from 0 to 100, where higher scores indicate better usability [Ban+08]. Though the scores are from 0 to 100, these values are not percentages and should be considered only in terms of their percentile ranking. A SUS score above 68 would be considered above average and anything below 68 is below average. However the best way to interpret the results involves *"normalizing"* the score to produce a percentile ranking [Ban+09].

2.4 Eclipse Modelling Tools

Eclipse Modeling Tools provides tools for building model-based applications [Eclteb], with Eclipse Modeling Framework (EMF) [Ecltea] technologies. The EMF is a *plugin* that allows to generate code based on a structured data model, the Ecore. The Ecore is the language meta-model. The modelled data can be validated, persisted, and edited [Ste+08].

It is not possible to express all the language syntactic rules through the language meta-model, so if we want to add more validations to a language we should use Epsilon Validation Language (EVL) [Eclted]. EVL is a validation language, from the Epsilon [Ecltec] family, that allows to verify the model correctness. EVL constraints are similar to OCL [1] constraints. However, EVL supports dependencies between constraints (*e.g.* if constraint A fails, do not evaluate constraint B). Also, with EVL we can evaluate inter-model constraints (unlike OCL).

2.5 Summary

In this chapter we presented some concepts related with Domain Specific Languages (DSLs) (section 2.1). First, we introduced the basic differences between DSLs and General-Purpose Languages (GPLs) (section 2.1.1), we presented the DSLs main stakeholders (section 2.1.2), and we talked about the DSLs development cycle (section 2.1.3). Also, in this chapter we presented Model-Driven Development (section 2.2), Human-computer interaction (HCI) (section 2.3) concerns and focused on Usability (section 2.3.1). We explained how to evaluate a system Usability with System Usability Scale (section 2.3.1.1). Finally, we presented Eclipse Modelling Tools technologies (section 2.4).

[1]Object Constraint Language. Latest access: August 2017. URL: `http://www.omg.org/spec/OCL/`

Chapter 3

Usability Driven DSL development with USE-ME

In this chapter we introduce the USE-ME framework. In section 3.2 we present the USE-ME framework, its domain concepts, meta-models and activity specifications. Also, in this section we present the results regarding the USE-ME pilot studies.

3.1 Introduction

The Usability Driven DSL development with USE-ME (USE-ME) approach was developed, in NOVA-LINCS, with the goal of *promoting quality in use of DSLs by building up a conceptual framework that supports the development process by leveraging usability as a first-class concern* [Bar+17]. As we saw in the previous chapter, in section 2.1.2, DSLs are intended to be used by the end users. However, the software industry does not seen to report investment on assessing DSLs with the end users. Possible explanations for this include the perceived high costs of such evaluations, as well as the lack of systematic approaches, guidelines and tools. Most of the evaluations are only performed at the final stages of DSL development, when changes have a significant impact on the budget.

Barišić et al. [Bar+17] state the DSL usability concerns must be addressed from the early stages of development so that DSL engineers can perform timely evaluations. As discussed in section 2.1.3, Visser recommends that DSLs should be developed incrementally in order to introduces abstractions that help to mitigate the risk of developing inappropriate solutions that often do not fit the end users and cannot be reused.

In section 3.2 we present the USE-ME framework, its domain concepts, meta-models and activity specifications. Also in this section we present the results regarding the USE-ME pilot studies.

3.2 The USE-ME framework

As mentioned in section 3.1, the USE-ME approach supports Software Language Engineers during the DSL development process taking into consideration the DSL final usability. The USE-ME activities can be seen as normal Expert Evaluator (*i.e.* the person that designs, gathers, interprets and synthesise DSL evaluations) activities, in the SLE (section 2.1.3) [Bar+17].

In section 3.2.1 we present the technologies used to develop USE-ME, and in section 3.2.2 we introduce the USE-ME development workflow, and describe each phase (*i.e.* the stakeholders involved, the phase models, and the artefacts produced).

3.2.1 Architecture and Technologies

The USE-ME conceptual framework (Figure 3.1) was specified on Cameo Systems Modeler [1], a Model-Based Systems Engineering (MBSE) environment, that is used to define, trace and visualize systems models and diagrams. The USE-ME main concepts were represented as UML [2] class diagrams, and the workflow as UML activity diagrams [Bar+17].

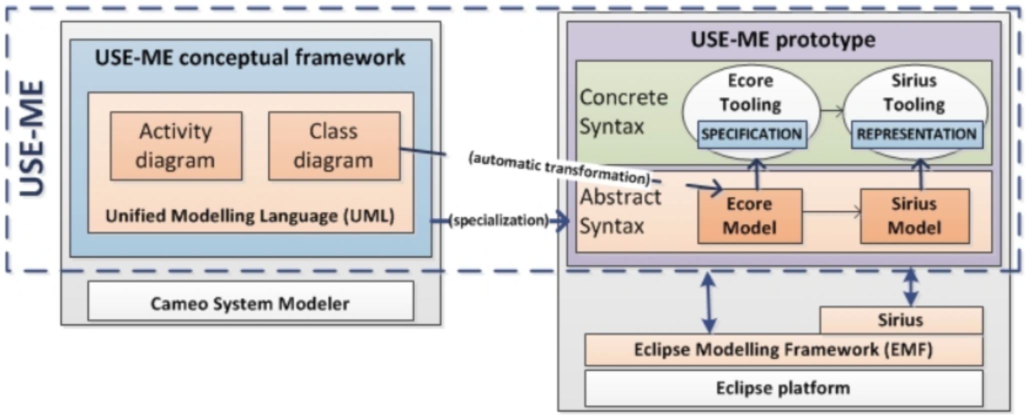


Figure 3.1: USE-ME architecture, taken from [Bar+17].

The USE-ME framework was developed in Eclipse Modeling Tools (mentioned in section 2.4) with Eclipse Modeling Framework (EMF) technologies [Ecltea] [Bar+17]. The EMF was used to develop the language meta-model, the Ecore (section 2.4), based on the class diagrams specified in Cameo System Modeler [Bar+17]. The USE-ME Ecore is composed by 7 small Ecores (*i.e.* context, goal, evaluation, interaction, survey, report, and utility), one for each development phase, we will explain each one of them in the next section 3.2.2. Sirius [3], allows to create custom graphical modeling workbenches by leveraging Eclipse Modeling Tools technologies, such as EMF. The workbench created is

[1]Cameo Systems Modeler. Latest access: August 2017. URL: `https://www.nomagic.com/products/cameo-systems-modeler`

[2]Unified Modeling Language. Latest access: August 2017. URL: `http://www.uml.org/`

[3]Sirius. Latest access: August 2017. URL: `https://eclipse.org/sirius/`

composed on a set of Eclipse editors (*e.g.* diagrams, tables and trees) which allow users to create, edit, and visualize EMF models. Sirius was used to create visual representations of USE-ME models [Bar+17]. To sum up, the USE-ME allows Software Language Engineers to design USE-ME instances in an EMF generated tree editor, and further, to preview the implemented representations with Sirius [Bar+17].

3.2.2 Workflow

The USE-ME framework life-cycle is composed by 6 phases (Figure 3.2), performed by the Expert Evaluator, that support the DSL development cycle (mentioned in section 2.1.3). Each modeling activity goal is described briefly below [Bar+17]:

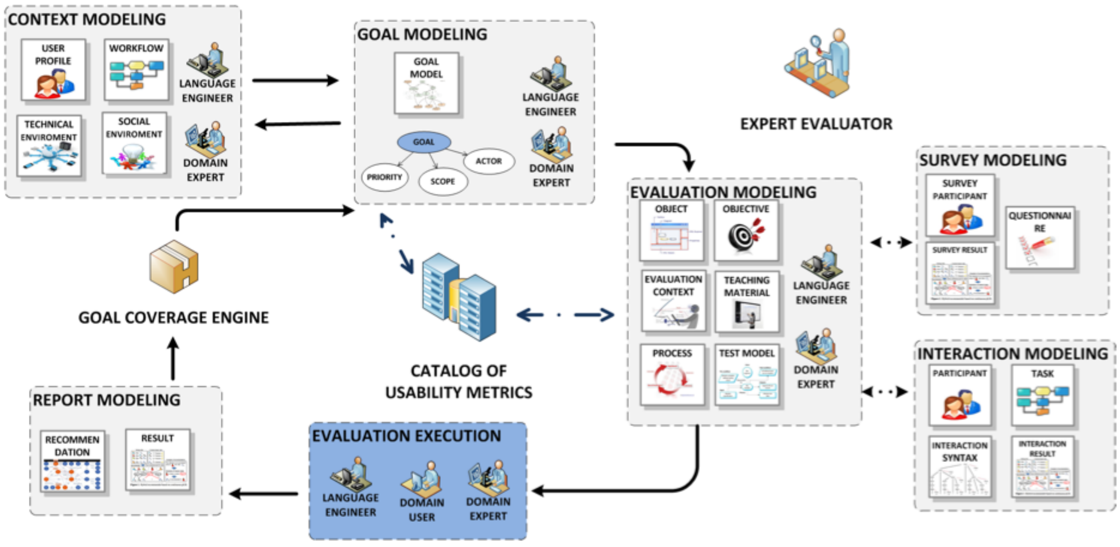


Figure 3.2: USE-ME life cycle, taken from [Bar+17].

1. the **Context Modeling**, allows to define the DSL context of use,
2. the **Goal Modeling**, sets the DSL goals,
3. the **Evaluation Modeling**, organizes the usability experiments,
4. the **Interaction Modeling**, defines the interaction tasks,
5. the **Survey Modeling**, supports the survey questionnaires,
6. the **Report Modeling**, collects the data from survey and/or interaction experiments.

The usability evaluation with USE-ME framework is an iterative and incremental development process (Figure 3.3). It starts with the **Context Modeling**, in order to define the language context of use. Next, in the **Goal Modeling** phase, the language usability goals and their correspondent scope is defined. After **Goal Modeling** comes the **Evaluation**

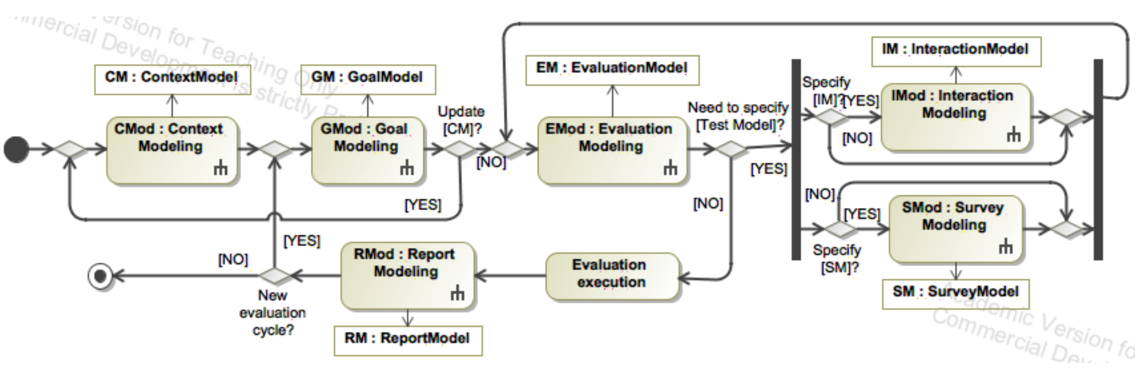


Figure 3.3: USE-ME activity diagram, taken from [Bar+17].

Modeling, in which the expected evaluation goals and their corresponding evaluation steps are set, and so the evaluation is prepared. The next phase, is the **Interaction Modeling** and/or **Survey Modeling**, since it is not mandatory to perform both models. The **Interaction Modeling** should be created if some interaction tasks are performed (*i.e.* DSL usability tests). The **Survey Modeling** should be used if the evaluation is through the means of a questionnaire. The last phase is the **Report Modeling**, in which the data from the evaluations is collected and then stored in a report.

In each one of the next sections, from section 3.2.2.2 to section 3.2.2.7, we will describe each development phase in more detail. We will present each phase meta-model, activity diagram, stakeholders involved, and artefacts produced.

3.2.2.1 Utility

The main goal of the Utility Package is to reuse the artefacts developed in previous evaluations [Bar+17]. The main elements from this phase are (Figure 3.4):

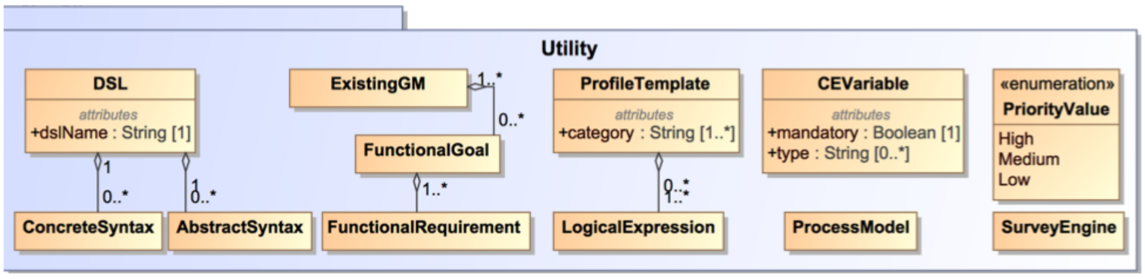


Figure 3.4: Utility class diagram, taken from [Bar+17].

- **DSL:** the language under evaluation. Each DSL should have an *Abstract Syntax* and/or *Concrete Syntax* associated;

- **Existing GM:** an existing goal model. If the DSL already was evaluated normally there is an existing goal model that represents the language goals in previous iterations;

- **Profile Template:** is a template for characterizing a specific user profile based on a *Logical Expression* (*e.g.* age > 18). Profile Templates can describe background information (*e.g.* demographic data, education, special needs/disabilities) and relevant experience with domain activities (*e.g.* expected knowledge sets, ontology);

- **CE Variable:** is a variable that describes the language environment. It can be associated to a Technical, Physical and/or Social environment;

- **Process Model:** refers to business process models that were developed during the DSL development;

- **Survey Engine:** refers to survey engines that were used to collect/store the experiments results (*e.g.* *Google Forms*);

- **Priority Value:** is a value that represents an object priority (*e.g.* a stakeholder priority). It can have one of 3 values: *High*, *Medium*, or *Low*.

3.2.2.2 Context Modeling

In the **Context Modeling** phase the Software Language Engineer and the Domain Expert have to answer to the following questions:

1. Who will use the DSL?

2. Where will the DSL be used?

3. How is the DSL expected to be used?

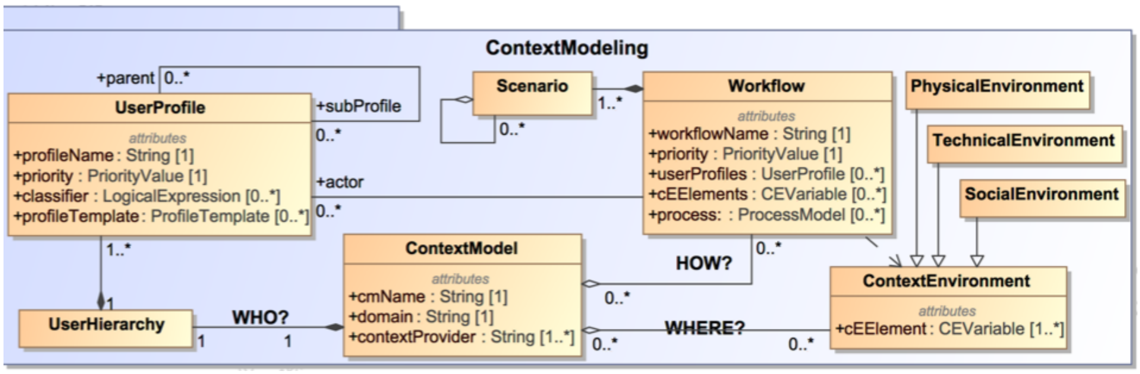

ContextModeling
+parent 0..*
UserProfile
attributes
+profileName : String [1]
+priority : PriorityValue [1]
+classifier : LogicalExpression [0..*]
+profileTemplate : ProfileTemplate [0..*]
+subProfile
0..*
+actor
0..*
1..*
1
UserHierarchy
WHO?
1
1
Scenario
0..*
1..*
Workflow
attributes
+workflowName : String [1]
+priority : PriorityValue [1]
+userProfiles : UserProfile [0..*]
+cEElements : CEVariable [0..*]
+process: : ProcessModel [0..*]
PhysicalEnvironment
TechnicalEnvironment
SocialEnvironment
ContextModel
attributes
+cmName : String [1]
+domain : String [1]
+contextProvider : String [1..*]
HOW?
0..*
0..*
WHERE?
0..*
0..*
ContextEnvironment
attributes
+cEElement : CEVariable [1..*]

Figure 3.5: Context Model class diagram, taken from [Bar+17].

The main elements from this phase are (Figure 3.5):

- **User Profile:** allows to answer to the question **who** will use the language. The USE-ME approach suggests that regarding the DSL in evaluation there are 5 User Profiles (*i.e.* stakeholders) that should be created, they are specifications of *DSL Stakeholder*:

Domain Expert, *Expert Evaluator*, *Language Engineer*, and *End User*. All these *stakeholders* have different *priorities* (mentioned in section 3.2.2.1): *DSL Stakeholder* and *End User* have the highest priority *High* since the DSL is supposed to be used by the end users; *Domain Expert* priority is *Medium* since it is a *stakeholder* that has a high knowledge on the domain, however it is not the final user; *Expert Evaluator* and *Language Engineer* have the lowest priority (*Low*) since they evaluate and build the language, respectively, but they do not use it. Each new *End User* sub-profile should be justified by the creation of a *Logical Expression* (mentioned in section 3.2.2.1);

- **User Hierarchy:** is composed by all the *User Profiles* created. The USE-ME approach suggests that the root node is the *DSL Stakeholder*. The *Domain Expert*, *Expert Evaluator*, *Language Engineer*, and *End User* are *DSL Stakeholder* sub-profiles. The new *End User* sub-profiles created, the DSL final users which inherit the parent properties, should be connected to the *End User*;

- **Context Environment:** allows to answer to the question **where** the DSL is going to be used. There are 3 types of environments: *Technical*, *Physical*, and *Social*. Each environment has a *CE Variable* (mentioned in section 3.2.2.1) that is a specification of the environment associated (*e.g. CE Variable* Computer is an specification of the *Physical* Environment). Not all the environments need to be created but at least one must be;

- **Workflow:** allows to answer to the question **how** the DSL is expected to be used. Each *Workflow* should be associated to an actor (*i.e. User Profile*) that is going to perform the workflow, to a *CE Variable* that represents the specification of the environment that is going to be used during the workflow execution, and to a priority;

- **Scenario:** represents a task, related with a *Workflow*, that has to be performed by a user. It should be created when a *Workflow* priority is high, in order to decompose the *Workflow* into *Scenarios*.

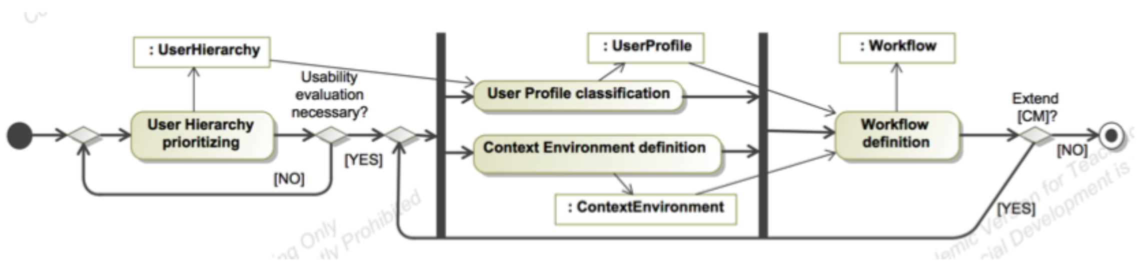

: UserHierarchy
User Hierarchy prioritizing
Usability evaluation necessary?
[YES]
[NO]
: UserProfile
User Profile classification
Context Environment definition
: ContextEnvironment
: Workflow
Workflow definition
Extend [CM]?
[NO]
[YES]

Figure 3.6: Context Modeling activity diagram, taken from [Bar+17].

The first artefact to be produced in the **Context Modeling** (Figure 3.6) is the User Hierarchy, by prioritizing the *User Profiles*. However, in order to do that, first the DSL *User Profiles* have to be specified. At the same time, two artefacts are produced: the User

Profile and the Context Environment. During the User Profile classification each *User Profile* is classified, this means that to each *User Profile* a *Logical Expression* is associated. During the Context Environment definition each environment (*i.e.*Technical, Physical and Social) is associated to a *CE Variable*. The final artefact to be produced, by the Expert Evaluator, is the Workflow. During the Workflow definition the *Workflows* are defined and, if they are important, they are decomposed into *Scenarios*. If the Expert Evaluator needs to make changes he can continue by extending the Context Modeling. If not the next phase is the Goal Modeling.

3.2.2.3 Goal Modeling

In the **Goal Modeling** phase the Software Language Engineer and the Domain Expert determine the language goals, the *Why is the language being developed?*. The main elements from this phase are (Figure 3.7):

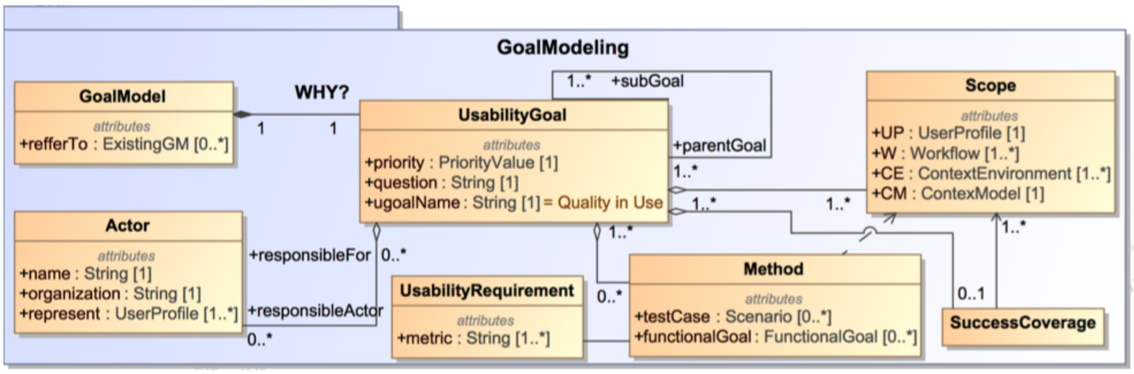


Figure 3.7: Goal Model class diagram, taken from [Bar+17].

- **Usability Goal:** represents the language usability goals. The USE-ME approach considers the Usability Goal *Quality in Use* as the highest goal of a DSL. The Usability Goal *Quality in Use* is decomposed into more refined *Sub Goals*. One of the *Sub Goals* specified must have the same priority as the *Parent* goal;
- **Scope:** should be related to one *Usability Goal*, since it specifies the scope to which an *Usability Goal* applies to;
- **Actor:** represents a *User Profile* and is responsible for achieving a specific *Usability Goal*. The USE-ME approach suggests that an *Actor* representing the User Profile *Expert Evaluator* should be created and should be responsible for achieving the Usability Goal *Evaluating the DSL*;
- **Method:** allows to define the measurable requirements (*i.e. Usability Requirements*) that contribute to achieve a *Usability Goal*. Each Method should have a *Test Case* associated in order to evaluate the *i.e. Usability Requirements*;
- **Usability Requirement:** contributes to achieve a *Usability Goal*;

- **Success Coverage:** represents the evaluated context coverage.

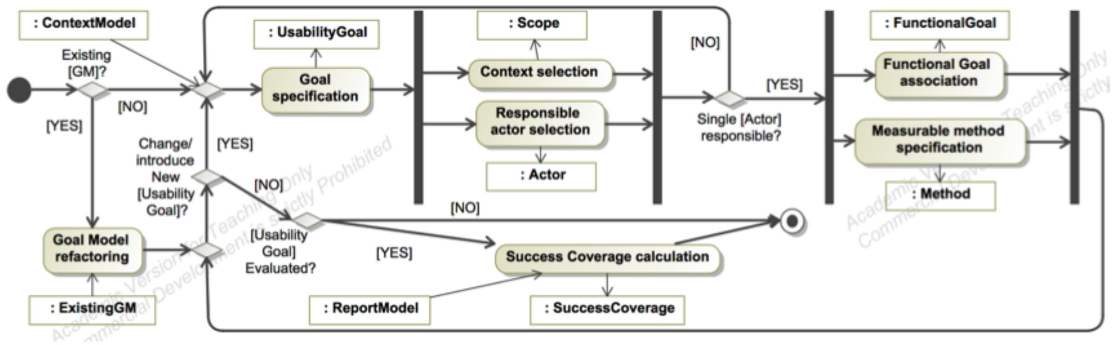


Figure 3.8: Goal Modeling activity diagram, taken from [Bar+17].

The first artefacts to be produced in the **Goal Modeling** (Figure 3.8) are the *Usability Goals*. During the Goal specification the *Usability Goals* are defined. *Usability Goals* represent the goals, highest objectives, that the language should achieve. At the same time two artefacts are produced, the *Scope* and the *Actor*. During the Context selection the *Usability Goals* are associated to a *Scope*. During the Responsible actor selection the *Usability Goals* are associated to an *Actor* responsible for achieving the *Usability Goal*. Next, if the *Usability Goals* are decomposed into *Sub Goals* that only have one *Actor* responsible, two artefacts are produced at the same time. During the Functional Goal association the *Functional Goals*, provided by the Language Engineer, are associated with the *Method*. During the Measurable method application all the measurable requirements *i.e. Usability Requirements* are specified. During Success Coverage calculation, after the last development phase, the *Success Coverage* is produced. The *Success Coverage* indicates the evaluation results. The next phase is the Evaluation Modeling.

3.2.2.4 Evaluation Modeling

In the **Evaluation Modeling** phase the Software Language Engineer and the Domain Expert organize the usability experiments. The main elements of this phase are (Figure 3.9):

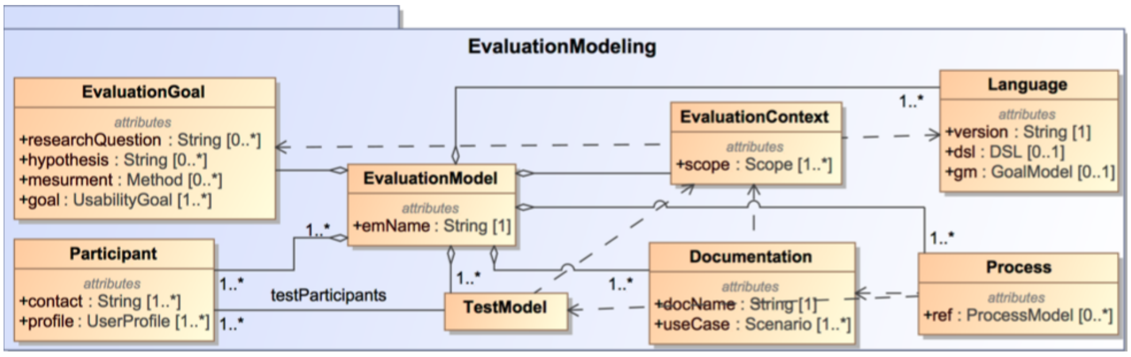


Figure 3.9: Evaluation Model class diagram, taken from [Bar+17].

- **Evaluation Goal:** allows to define the experiment problem, the research questions, and the hypothesis;
- **Language:** is the language under evaluation. If the is a comparative evaluation then a second language should be created;
- **Participant:** refers to the actual participants, that represent a specific *User Profile*, in the experiment;
- **Evaluation Context:** in which context *i.e. User Profiles, Workflows, and Context Environments* are the experiments executed;
- **Documentation:** refers to the teaching materials (*e.g.* presentations, videos, guided examples, etc.) that were produced for the experiment;
- **Process:** defines the experiment process, by modeling the activities that should be performed, and the *User Profiles* that are going to execute the activities.
- **Test Model:** refers to usability activities that do not require learning (*i.e.* questionnaires, interviews, and observations). The *Test Model* is supported by the Interaction Modeling and Survey Modeling.

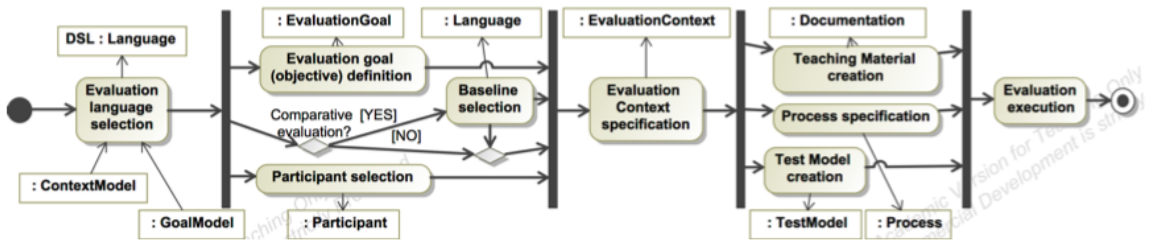


Figure 3.10: Evaluation Modeling activity diagram, taken from [Bar+17].

The first artefact to be produced in the **Evaluation Modeling** (Figure 3.10) phase is the language under evaluation. Next, if the evaluation is comparative then a new language should be created, producing an artefact *Language*. Then, the evaluation goals are defined and so the artefact *Evaluation Goal* that describes all the Evaluation Goal is created. After that, the Evaluation Expert defines the actual participants in the evaluation that represent *User Profiles*. Next, during the Evaluation Context specification the context of the evaluation is defined, and so the *Evaluation Context* is produced. At the same time three artefacts are produced: the *Documentation*, *Process*, and the *Test Model*. During the Teaching Material creation (*e.g.* presentations, videos, guided examples, etc.) the *Documentation* is produced. During the Process specification the experiment *Process* is created by defining the activities that should be performed. In parallel, the *Test Model* is produced by defining activites that do not require learning such as questionnaires, interviews, or observations. The next phase is the Interaction and/or Survey Modeling.

3.2.2.5 Interaction Modeling

The main goal of **Interaction Modeling** is to measure DSL usability through user interaction tasks. The main elements of this phase are (Figure 3.11):

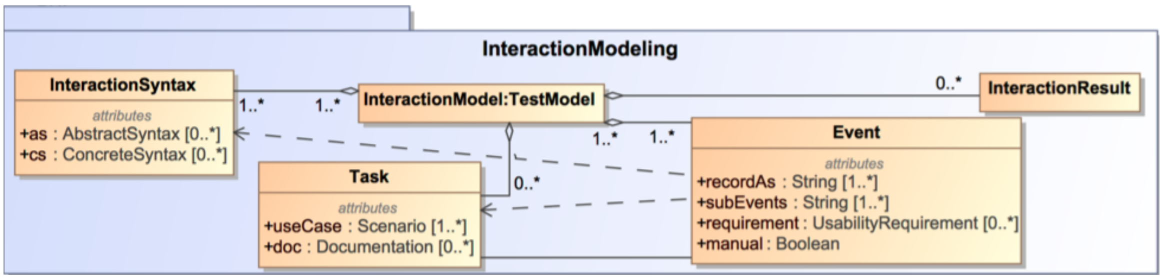

InteractionModeling
InteractionSyntax
attributes
+as : AbstractSyntax [0..*]
+cs : ConcreteSyntax [0..*]
1..*
1..*
InteractionModel:TestModel
0..*
InteractionResult
1..*
1..*
Event
attributes
+recordAs : String [1..*]
+subEvents : String [1..*]
+requirement : UsabilityRequirement [0..*]
+manual : Boolean
0..*
Task
attributes
+useCase : Scenario [1..*]
+doc : Documentation [0..*]

Figure 3.11: Interaction Model class diagram, taken from [Bar+17].

- **Task:** is an activity that should be performed by the participant, and that further is going to be analysed;
- **Interaction Syntax:** represents the language syntax (*i.e.* abstract and concrete syntax). A DSL does not need to have associated both but must have at least one;
- **Event:** describes how one event is going to be captured (*i.e.* which sub-events are going to be performed by the participant), analysed (*e.g.* observation) and recorded (*e.g.* screen record, live observation and think aloud);
- **Interaction Result:** stores the statistical analysis and the results from the experiment.

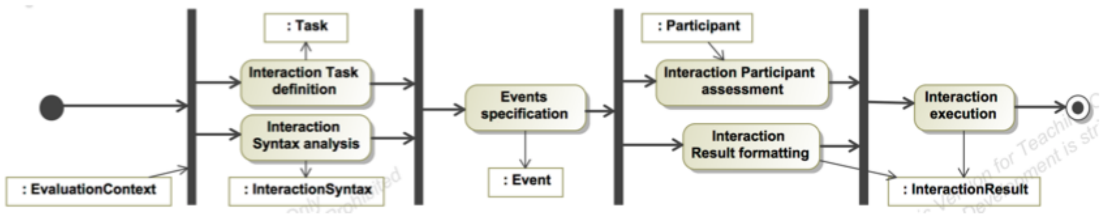


Figure 3.12: Interaction Modeling activity diagram, taken from [Bar+17].

The first artefact to be developed in the **Interaction Modeling** (Figure 3.12) activity is the *Task*, during the Interaction Task definition. In parallel, during the Interaction Syntax analysis, the language *Interaction Syntax* is described by defining the *Abstract Syntax* and/or *Concrete Syntax*. After that, during the Events specification, the *Event* is detailed by defining how the event is going to be captured (*i.e.* which sub-events are going to be performed by the participant), analysed (*e.g.* observation) and recorded (*e.g.* screen record, live observation and think aloud). The last artefact to be produced is the *Interaction Result*, during the Interaction execution, which is going to store the experiment results.

3.2.2.6 Survey Modeling

The main goal of **Survey Modeling** is to gather information in order to describe, compare and explain certain behaviours. The main elements of this phase are (Figure 3.13):

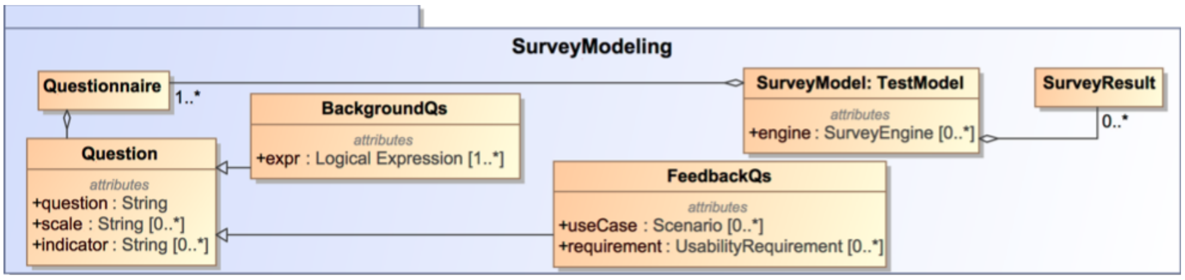


Figure 3.13: Survey Model class diagram, taken from [Bar+17].

- **Questionnaire:** defines a set of questions, *background* and *feedback*. These questionnaires can be done in a *Survey Engine* that automatic organizes answers (*e.g.* Google Forms, Survey Monkey, etc.);
- **Background Qs:** is a question related with the participant *background* (*e.g.* demographic data, education, knowledge, special needs/disabilities);
- **Feedback Qs:** is a question that collects the user satisfaction *feedback* regarding one specific subject;
- **Survey Results:** stores the statistical analysis and the results from the questionnaires in a *Survey Engine* (*e.g.* Google Forms, Survey Monkey, etc.).

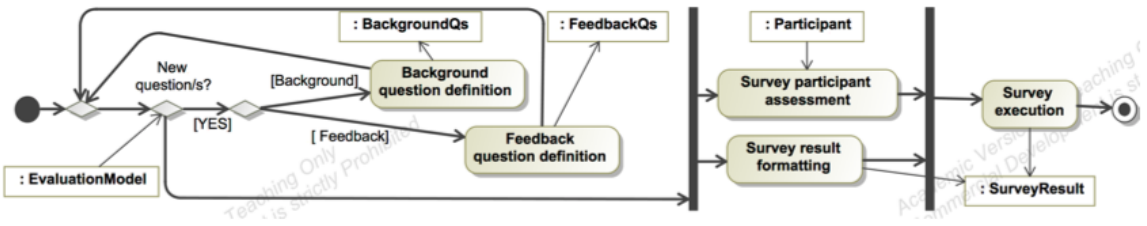


Figure 3.14: Survey Modeling activity diagram, taken from [Bar+17].

The **Survey Modeling** (Figure 3.14) starts with a decision. If the Evaluation Expert decides to add more *Background* questions then the next step is Background question definition and the artefact produced is *BackgroundQs*; if the Evaluation Expert decides to add more *Feedback* questions then the next step is Feedback question definition and the artefact produced is *FeedbackQs*. Next, during the Survey participant assessment the experiment *Participants* are assigned, and during the Survey result formatting the *Survey* is performed. Then, during the Survey execution the final artefact to be produced is the *Survey Result*, that stores the results from the experiments.

3.2.2.7 Report Modeling

The main goal of *Report Modeling* is to construct a final report on the experiment, by collecting all the results and to suggest changes for the next iterations. The main elements of this phase are (Figure 3.15):

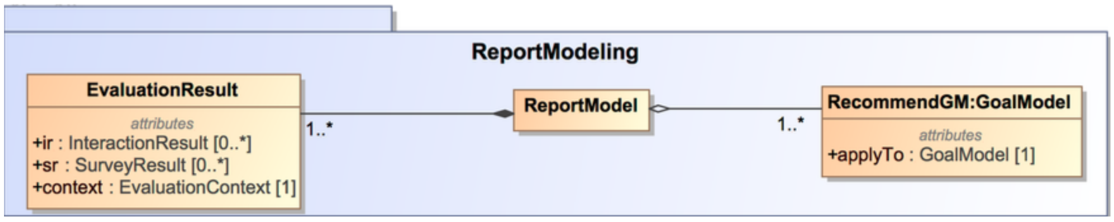


Figure 3.15: Report Model class diagram, taken from [Bar+17].

- **Evaluation Result:** gathers the results values from both *interaction* and *survey* experiments, and stores these results in a report;
- **RecommendGM:** refers to the *Usability Goals* that were tested and suggests requirements to improve in next iterations.

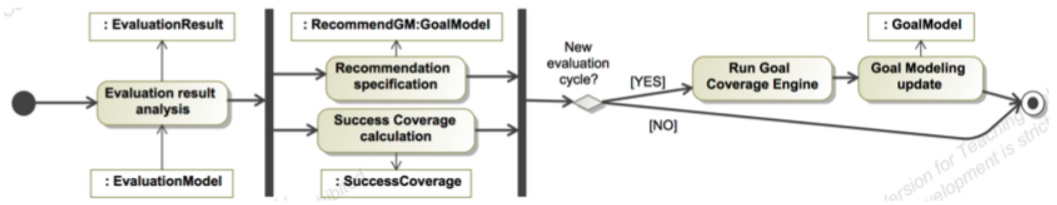


Figure 3.16: Report Modeling activity diagram, taken from [Bar+17].

The first artefact to be produced in the **Report Modeling** activity (Figure 3.16) is the *Evaluation Result*, during the Evaluation result analysis. Based on the results from the *Evaluation Result*, the Expert Evaluator performs the Recommendation specification, that produces the *RecommendGM*. At the same time, during the Success Coverage calculation, the *Success Coverage* (mentioned in section 3.2.2.3) is produced which helps the Expert Evaluator to decide if the goals should be redesigned or not.

3.2.3 Pilot Studies

The pilot studies were conducted with Computer Science Master students (*i.e.* novice users), that did not know anything about the USE-ME approach, during a DSL course. Each group was developing their own DSL. In total 4 different DSLs were being developed. After an introduction to usability evaluation the students received a guided tutorial on the USE-ME tool, and were also guided during the usability evaluation of their own DSLs. At the end of the session, the students filled a *background* and *feedback questionnaire*. Students found that although it was *"more or less easy"* to understand the approach it was not so easy to model the DSL evaluation with the framework. Some pointed out that *"there were too many steps to follow"* and that the tool should provide a *"guided cycle"* [Bar+17]. To sum up, they did not feel very confident while using the USE-ME tool [Bar+17].

3.3 Summary

In this chapter we introduced the USE-ME framework. In section 3.2 we present the USE-ME framework, technologies used, domain concepts, meta-models and activity specifications. Also, in this section 3.2.3 we present the results regarding the USE-ME pilot studies.

Chapter 4

Related Work

In this chapter we present some tools related with this dissertation: ***Tool Support for Agent Development using the Prometheus Methodology*** *(section 4.1),* ***J-PRiM: A Java Tool for a Process Reengineering i* Methodology*** *(section 4.2),* ***PETIC Wizard Proposal: a Software Tool for Support PETIC Methodology*** *(section 4.3),* ***A Qualitative Study on User Guidance Capabilities in Product Configuration Tools*** *(section 4.4),* ***Business process modeling with continuous validation*** *(section 4.5),* ***Rule-based detection of inconsistency in UML models*** *(section 4.6), and* ***Cognitive support, UML adherence, and XMI interchange in Argo/UML*** *(section 4.7). All these approaches are concerned with user guidance, models quality, increasing productivity and decreasing maintenance costs. For each approach we provide a description, present the stakeholders involved, present its implementation, and explain how we applied these approaches to the USE-ME framework.*

4.1 Tool Support for Agent Development using the Prometheus Methodology

In [PW02] Padgham et al. describe the Prometheus Design Tool (PDT) and how it supports the Prometheus Methodology. Prometheus is an intelligent agent methodology that covers all the development stages (*i.e.* specification, design, implementation and testing/debugging). PDT is a graphical user interface that covers 3 design phases of the Prometheus methodology: *System Specification*, *Architectural Design*, and *Detailed Design*. Since the Prometheus methodology is iterative, when changes are done in the design they affect other parts as well. They are linked together, so there is a need to update the other parts as well. However, it is really hard for the *System Developer* to manually check if the design remains consistent. So, the main goal of the PDT is to support the *System Developer*, during the Prometheus methodology, by helping the *System Developer* to maintain a consistent design. *Consistency Checking* in PDT has 2 features: **continuously active**, that prevents some errors from occurring (*e.g.* not possible to have references to non-existent entities, two entities with the same name, only valid links between entities are allowed, etc.), and **consistency check** that generates a list of errors and warnings, performed on demand, that can be checked by the *System Developer* (*e.g.* writing internal data that is never read, etc.).

Despite our approach (section 3.1) is different from the Prometheus Methodology, *consistency check* is performed based on the relationships between design artefacts and the USE-ME approach validation is based on a meta-model, we can retain some ideas from the PDT tool. In USE-ME we guide, validate and suggest the *Software Language Engineer* actions during the DSL development process but we only do that on demand, like the *consistency check* feature on PDT, since it is intended to help novice users more than experts.

4.2 J-PRiM: A Java Tool for a Process Reengineering i* Methodology

In [Gra+06] Grau et al. state that i* is widely used in fields such as requirements engineering, organizational analysis, business process reengineering, etc. However, it requires the adoption of a methodology for defining the models and tool support in order to manipulate these models. The i* models are represented graphically, which is one of the strongest points, but when users work on a specific problem or domain some difficulties often appear, such as: overload of variants of the i* language, a lack of guidelines for constructing the models, and lack of tools to support the users. In order to deal with these problems Grau et al. developed J-PRiM, a Java tool, that guides the users through the construction of i* models. The J-PRiM is composed by 5 phases: *Analysis of the Currrent Process*, *Construction of the i* Model*, *Generation of Alternatives*, *Evaluation of Alternatives*, and *Specification of the New System*. Each phase is decomposed into steps. There are 3 types of steps: *forms* (*i.e.* steps that require the user to enter data), *guided* (*i.e.* steps that complete existing data but still require user expertise), and *automatic* (*i.e.* steps that generate new data without user interaction). During the development the tool guides the user for obtaining the i* model (*e.g.* by providing the actors created in the beginning when they need to be associated, by helping the user to distribute responsibilities between actors, by defining the metrics for evaluating the model, etc.). Usually the i* models are represented graphically. However, the J-PRiM shows the i* elements in a tree-form hierarchy like the USE-ME framework. Also in the J-PRiM all the phases and steps are ordered in specific tabs, which allow to guide the user through the tool. Since the PRiM is a iterative process when the user changes the data in the early steps the J-PRiM allows to apply those changes in the next phases. This feature is also used in the USE-ME framework. For example, when a stakeholder is deleted USE-ME alerts the user to associate another stakeholder where the deleted one was associated.

4.3 PETIC Wizard Proposal: a Software Tool for Support PETIC Methodology

In [Pal+12] Palmeira et al. describe an Information and Communication Technology (ICT) strategic planning methodology called PETIC. The goal of PETIC is to use information and knowledge, for a competitive advantage, to manage businesses by guiding the manager through the strategic planning. PETIC is composed by 5 components: *PETIC artefact*, *TIC Process Catalogue*, *TIC Stock Repository*, *Costs Graphs versus Importance*, and *Gantt Maps*. In order to support the manager through the strategic planning, to decrease costs, and to decrease development time, Palmeira et al. developed PETIC Wizard. When using the PETIC Wizard the manager will be able to easily check the achieved goals and targets, and the associated costs.

The main goal of this tool is to provide guidance to the manager to develop PETIC artefacts, just like our goal with the USE-ME tool, since developing artefacts without a tool/wizard support takes more time and the costs are higher. In [Pal+12] Palmeira et al. did not describe which technologies they used to developed the PETIC Wizard.

4.4 A Qualitative Study on User Guidance Capabilities in Product Configuration Tools

In [Rab+12] Rabiser et al. performed a qualitative study on tools capabilities, using cognitive dimensions of notations, for user guidance in product configuration. Rabiser et al. developed a *wizard* called DOPLER ConfigurationWizard (CW). The main goal of this configuration tool is to guide users during the product configuration with a focus on business-oriented end users. There are 3 types of configuration tools: *Feature-oriented tools* (*i.e.* if the configuration options are presented in a tree; the user guidance is not the primary focus in these tools, since users can selects any node of the tree), *Knowledge-based tools* (*i.e.* if the configuration options are described using constraints; the user guidance is important in these tools), and *Workflow-oriented tools* (*i.e.* if the configuration options are described to users by presenting choices, like questions, in a certain order). Rabiser et al. used cognitive dimensions to evaluate and characterize user guidance capabilities. Next, we describe each capability that is applied to USE-ME approach and explain how:

- **Hiding and showing:** configuration options depending on stage of the configuration. In the USE-ME approach we use this by providing to the user only the necessary activities that need to be performed at a certain workflow phase, distinguishing between mandatory and suggestion activities;

- **Views and filters:** views allow to structure the decision space, while filters can be used to only show certain options. In the USE-ME approach we use this by only showing *errors* and *warnings* from a particular development phase;
- **Branching and navigation support:** guide the user through the configuration activities. In the USE-ME approach we use this by guiding the user through the workflow, so the tool only shows the next workflow step when the user completed the previous one;
- **Freedom in navigation:** the tool should give freedom to the user, it should not force the user to follow a strict guidance. In the USE-ME approach we guide novice users more than experts, so for that purpose our tool only guides the user on demand and when the user needs help from the tool;
- **On-the-fly validity checks:** of user choices. In the USE-ME approach we validate *Software Language Engineer* choices in a more preventive way, so we provide explanation and help in order to guide the user;
- **Immediate feedback:** to understand the effects of their choices. In the USE-ME approach we provide to the users immediate feedback on their actions through validation, on demand. For that we suggest that the user validates the model every time the make important changes;
- **Annotations and comments:** on options. In the USE-ME approach we explain to the user his/her options, we do not want a strict guidance so we educate the user so that he makes the best option regarding the DSL under evaluation.

The DOPLER CW guides the users through questionnaires, so in a workflow-oriented tool. When a decision depends on other decisions, Rabiser et al. constrained the validity options.

Despite the work done by Rabiser et al. was applied to configuration tools we retained and applied the cognitive dimensions study to the USE-ME approach since this tool had guidance problems (section 3.2.3). DOPLER CW was developed using the cognitive dimensions concepts. However, Rabiser et al. did not mention how they implemented it.

4.5 Business process modeling with continuous validation

In [Kü+10] Kühne et al. describe a modeling tool that identifies problems in business process models through the application of rules. Like in the DSL development most validation methods are only applied when the model is already complete. The validation approach proposed by Kühne et al. provides immediate feedback about the modeling errors to the modeler during the model construction. It identifies technical (*e.g.* deadlocks in the control flow) and *'bad style'* errors. The tool provides feedback about the problem, identifies the reasons and gives suggestions on how to fix it. In order to provide to users this feedback Kühne et al. developed a feature called *continuous validation* that allows to detect and fix errors at a early stage in the process modeling based on rules. The *continuous validation* was used to validate Event-Driven Process Chains (EPC) syntactical, semantic and pragmatic issues. For EPC models, the syntax is decribed by the meta-model. However, not all syntactical restrictions can be captured by the meta-model. So in order to check the syntactic correctness of a model, OCL constraints were added to the meta-model. Kühne et al. did not want to restrict the modeler, but to give him/her feedback about the possible errors and improvements. For that purpose Kühne et al. defined the following principles:

- the validation rules should be expressed in a human-readable manner;
- the validation rules should refer to parts of the model that may cause errors;
- the validation rules should be specific enough so that the user can easily understand the feedback;
- the validation rules should be seamlessly integrated into the modeling tool with the capability to provide error and suggestions.

The application of these principles results in a process-specific validation rules and helper functions.

For the implementation of the approach Kühne et al. chose oAW because of its build-in GMF integration. The validation rules therefore were expressed in the oAW Check language, where the keyword **error** represents an error and specifies a corresponding advice, and the keyword **warning** provides a suggestion.

We applied the *validation* feature (not continuous) and principles proposed by Kühne et al. to the USE-ME framework, to provide feedback about errors and suggestions to the *Software Language Engineer* since it is not possible to express all the validation rules through the language meta-model. The mandatory activities are represented as errors, and the suggested activities are represented as suggestions/improvements. However, since the USE-ME framework was developed using the Eclipse Modeling Tools (section 3.2.1) with Eclipse Modeling Framework (EMF) technologies, we choose to use EVL (section 2.4) since it is a validation language from the Epsilon [Ecltec] family, that allows to verify the model correctness.

4.6 Rule-based detection of inconsistency in UML models

In [Liu+02] Liu et al. describe an approach that detects inconsistencies, notifies users, and recommends solutions through specific rules during the design process. Every time the designers change the language model there is a chance of inadvertently introducing inconsistencies that are really hard to identify and resolve manually. In order to solve that problem Liu et al. developed a software, RIDE (Rule-based Inconsistency Detection Engine), that automates the detection and resolution of design inconsistencies (*i.e.* information redundancy, nonconformance to standards and requirements, and the propagation of change through a model as it evolves) in UML models, by using OCL constraints that maintain the well-formedness of the semantics. In order to detect these inconsistencies RIDE, runs a production system in the background of an editor, so when the rules detect inconsistencies they add an entry to the *working memory* of the production system. Liu et al. describe 3 types of inconsistencies:

- **Redundancy:** when an artefact is represented multiples times;
- **Conformance to Constraints and Standards:** *constraints* can be from internal or external conflicts. The model has to include the *standards* (*i.e.* best practices, industry standards, and corporate standards);
- **Change:** due to change requests in the design, inconsistencies can easily be introduced.

The production system works in a cyclic way by:

1. **Recognizing:** the rules that can be applied;
2. **Resolving conflicts:** choosing one of the rules to be executed;
3. **Acting:** by applying the rule.

Most of the actions require user feedback, therefore the approach notifies the user with the inconsistency notice, and then the user can solve the problem. There are 4 types of rules:

- **Inconsistency rules:** that represent inconsistencies in the design;
- **Resolution rules:** that correspond to user fixing;
- **Cleanup rules:** that remove expired inconsistency from the *working memory* elements;
- **Orphan control rules:** that remove the *working memory* elements whose parent is invalid or has been deleted.

RIDE was implemented in Java, with Jess - Java Rule Engine, and can be integrated into an existing UML Design Environment.

We applied Liu et al.'s work to the USE-ME approach since we have 3 types of rules: error, guidance and suggestions. However, in our case the user solves the problems since there are some suggestions, for example, that the tool should help the user to decide but the final decisions is from the user. Also, the user is the one that has the knowledge on the DSL under evaluation since the tool behaves equally for all the DSLs under evaluation.

4.7 Cognitive support, UML adherence, and XMI interchange in Argo/UML

In [RR00] Robbins et al. describe ARGO/UML, an object-oriented design tool that uses UML design notation, that provides support for cognitive tasks (*i.e.* decision-making, decision ordering, and task-specific design understanding) as a way to increase productivity and quality of the final designs, and to decrease maintenance costs. *Design critics* are active agents that continuously check the design for potential errors, stylistic violations, and incomplete sections. *Design critics* provide knowledge to the designers when they are missing information about the problem or about the solution domain. The *design critic* feature is continuously active and designers cannot control when the validation is applied, and can only see the feedback produced. The goal of the *critics* is to warn the designer about potential problems or suggest improvements in the design. Also, each *critic* is independent from the other and delivers is own feedback to the designer. All the feedbacks are stored in a *'to do'* list, where items are grouped by priority, category, etc. When the designer choose a problem to solve, from the *'to do'* list, all the design elements in all diagrams related to that problem are highlighted. To sum up, the designer is responsible for solving a *critic*. However, some *critics* are harder to solve and for that purpose Robbins et al. developed a ARGO/UML *wizard*. The *wizard* guides the designer through the necessary steps and decisions to solve the *critic*. However, the designer can leave the *wizard* at any time and manipulate the models on his own.

ARGO/UML was developed using JavaBeans with a UML meta-model. ARGO/UML uses XMI files to store design representations, and for validating the representations Robbins et al. used OCL.

In ARGO/UML the *design critic* is continuously active but in the USE-ME framework the validation is performed by the *Software Language Engineer* on demand. In the ARGO/UML tool designers do not see the *critics*, only the feedback. In the USE-ME tool the users do not see the validation rules only the results from applying the rules to the model, so they only see the *errors* and the *suggestions*. In the ARGO/UML the *critics* are independently of others, but in the USE-ME tool they are not since we guide the users throughout the workflow so rules depend on each other.

4.8 Summary

In this chapter we presented some approaches that are concerned with user guidance, models quality, increased productivity and decreased maintenance costs. For each approach we presented a description, the stakeholders involved, its implementation, and explain how we applied each one to the USE-ME framework.

Chapter 5

Validation Rules Implementation

In this chapter we discuss the implementation alternatives and the problems that were found during the USE-ME pilot studies (section 5.1). We also describe the solution that we found to be the most appropriate to deal with the problems described (section 5.2). Then, we explain in more detail how we implemented the validation rules and how we integrated them on the USE-ME framework (section 5.3). We finish this chapter by providing an use case scenario (section 5.4).

5.1 Implementation Alternatives

For the prototype implementation, we considered some alternatives. First, we thought about building the prototype from scratch but this would take more time, and since we already had a framework built at NOVA-LINCS, the USE-ME framework [Bar+17] (mentioned in section 3), that already supports the iterative and incremental DSL development process taking in consideration the language usability, we decided to use this framework as the starting point. The USE-ME framework was developed using Eclipse Modeling Tools with Eclipse Modeling Framework (EMF) technologies (mentioned in section 3.2.1) [Bar+17]. The EMF was used to develop the language meta-model, the Ecore, that allows to define the language syntactic rules (section 2.4). However, it is not possible to express all the language syntactic rules through the language meta-model. So, by not adding more validation to the meta-model, with a validation language, the *Software Language Engineers* could create USE-ME models that were not correct.

During the USE-ME framework pilot evaluations the participants, that played the role of *Software Language Engineers*, found that although the approach was *'more or less easy'* to understand it was not so easy to model the DSL evaluation with the tool (mentioned in section 3.2.3). Most of the students pointed out that the USE-ME framework has *'too* *many steps to follow'* and that the tool should provide *'a form of guidance'*. To sum up, the users did not feel very confident while using the tool [Bar+17]. Therefore, our solution should validate the model correctness while also guiding the *Software Language Engineer* throughout the process.

5.2 Solution

In order to ensure that the USE-ME model is correct, and also to provide guidance to the *Software Language Engineer* throughout the process we analyse some alternatives. The tools that we presented (mentioned in section 4) support the user (*i.e.* system developer, manager, designer, etc.) throughout all the approach development stages by providing features that verified the correctness of the models and then provided feedback about the errors and suggestions to the user (*e.g.* consistency checking, etc.), or by ordering phases and steps in specific tabs which guides the user, or by providing a *wizard* that helps the user to develop language artefacts [RR00]. In the field of Human-Computer Interaction (HCI) one of the most popular guidance forms is to create a wizard when the tool workflow is complex and error-prone, and if there are many steps in one task and they must be completed in a specific order [Dow+05]. However, most *wizards* that we presented (mentioned in section 4) are used to automate the work of the user which is not our goal since we are going to guide the *Software Language Engineer* through the workflow but he is the one that has the knowledge on the DSL so he should make the best decisions regarding their DSL evaluation with the tool support.

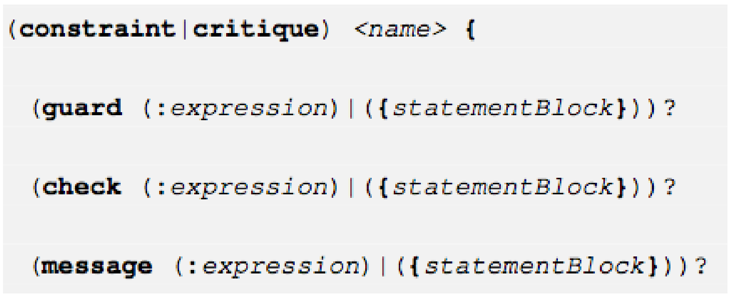
Some of the validations performed in some tools (mentioned in section 4) were based on relationships between design artefacts [PW02], and some were performed based on the language meta-model [Kü+10] [RR00]. In the USE-ME framework we perform the validation based on the language meta-model, in order to provide immediate feedback about the modeling errors and suggestions during the model construction. These type of tools are called *Rule-based Systems*. Also, some approaches performed the validations continuously [Kü+10], while others performed the validations *on demand* [PW02], so the user decides when he would like to validate the model. In the USE-ME framework we are more interested in performing validation *on demand* since we believe that the tool should give freedom and not force the user to follow a strict validation/guidance. We also validate *Software Language Engineer* actions in a more preventive way, so we provide explanation and help in order to guide the user not to commit errors.

Our solution for the USE-ME framework is to transform it into a *Workflow-oriented tool* since the activities are described to the users in a certain order providing guidance, with validation rules (see Appendix A) that also check the USE-ME model correctness, so it is also a *Rule-based System*. Some activities, as we already mentioned, are mandatory and some are suggestions so we need to provide to the user the right information regarding the activity so that he decides to take the suggestion or not. Also, our goal is educate novice users on our DSL approach and framework, since most of the users do not read the documentation [NW06] we provide with each rule the basic information on why the user should create an artefact, or associate a name, etc. As we discussed before we do not want to restrict the user with the validation rules (see Appendix A), however, we think that in the first time using the tool it is crucial to follow the validation/guidance rules in order to learn how efficiently use the framework.

5.3 Validation Rules

As mentioned in the previous section 5.2 our solution verifies the model correctness by identifying errors and suggestions in USE-ME models, through the application of validation rules (see Appendix A). The USE-ME framework was developed in Eclipse Modeling Tools (mentioned in section 3.2.1) with Eclipse Modeling Framework (EMF) technologies, so in order to add more validation to the meta-model (*i.e.* Ecore) we used a validation language.

The tools presented in section 4 that performed model validations used OCL to express the rules, however, since we used EMF to implement the USE-ME framework and the EMF has its own validation language, Epsilon Validation Language (EVL), we analyse both pros and cons of each language in order to decide which one would suit USE-ME interests better. As we mentioned in section 5.2 our goal with the validation rules (see Appendix A) is to educate novice users on the USE-ME framework and OCL has weak support for specific feedback messages, meaning that the users should be familiar with OCL in order to comprehend the failed rules [Kol+10]. Normally, development environments produce 2 types of feedback: errors and warnings (*i.e.* in our case they act like suggestions). While errors indicate critical deficiencies, warnings indicate non-critical issues. In OCL there is no such distinction between these two types of feedback, so it is harder for the user to prioritize which one he should fix first [Kol+10]. In OCL each rule is independent, meaning that *'if constraint A fails, don't evaluate constraint B'* it is not possible to express in OCL. In USE-ME almost each activity has a dependency on other activity, so it would be meaningless to evaluate a rule were its pre-condition already failed [Kol+10]. After analysing both languages we have decided to use EVL since we think its the most appropriate language to express our requirements.



```
(constraint|critique) <name> {

  (guard (:expression)|({statementBlock}))?

  (check (:expression)|({statementBlock}))?

  (message (:expression)|({statementBlock}))?
```

Figure 5.1: Concrete Syntax of EVL, adapted from [Kol+10].

In Figure 5.1 we describe EVL Concrete Syntax. Each EVL rule in USE-ME is composed by:

- **Constraint or Critique:** a *Constraint* captures critical errors that invalidate the model, while the *Critique* captures non-critical situations that do not invalidate the model, although they should be addressed by the user to increase the model quality. In the USE-ME implementation we used *Constraint* to represent **mandatory** activities or properties that need to be filled by the user, and *Critique* to represent **suggestion** activities or properties;
- **Guard:** limit the applicability of invariants. For example, in USE-ME we have a first rule that checks if the activity *Create User Hierarchy* was created, and have a second rule that checks if *User Hierarchy has name*. In the second rule we have a *Guard* that checks if the first rule is *true*, so if the *User Hierarchy* already exists, because it only makes sense to check if the *User Hierarchy has name* if the *User Hierarchy* exists;
- **Check:** is an expression to be checked, it can return true or false. If it returns false then the rule *message* is shown to the user;
- **Message:** allows to provide detailed feedback to the user on why the rule failed and what needs to be done in order to fix it.

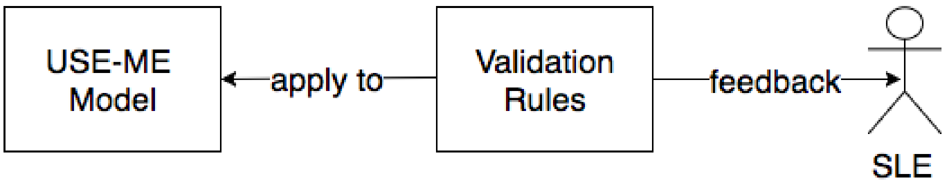


Figure 5.2: USE-ME framework with validation rules.

In Figure 5.2 we illustrate how we integrated the validation rules (see Appendix A) with the USE-ME framework. After creating the USE-ME model (see Appendix I), the *Software Language Engineer* needs to validate the model through the Validation Rules in order to get feedback (see Appendix I.6). The rules will guide, suggest and validate the *Software Language Engineer* actions during the DSL evaluation development. As we mentioned, the validation does not work continuously. It is performed *on demand*, so the user *validates* the model every time he needs feedback from the tool, either for validation or guidance.

The validation rules were integrated in Eclipse *Problems* tab, such as *Ecore* default messages and as messages generated when user compiles code, in order to avoid increasing the complexity of the solution, so every user should be comfortable with it. Every time the user *validates* the model, the list of feedback messages is updated.

5.3.1 Rules Design

Since most users do not read manuals and when it comes to use a new tool they prefer to experiment on their own, when we designed the validation rules (see Appendix A) we did it in a way that they transmit the right information to the user in a compact way [NW06]. In order to do that we applied the following principles [Kü+10]:

- the validation rules are expressed in a human-readable manner;
- the validation rules refer to parts of the model that cause errors;
- the validation rules are specific enough so that the user can easily understand the feedback;
- the validation rules are integrated into the modeling tool with the capability to provide error and suggestions.

The application of these principles results in a process-specific validation rules and helper functions. For both type of rules, *error* and *suggestion*, we present the following message format:

USE-ME *development phase* **Error or Suggestion**: *informative message*

In the beginning of the rule we inform the users that the feedback belongs to the USE-ME model, in case they have more messages from other projects.

In EVL we can specify the context of the rules, a context specifies the kind of instances on which the contained rules will be evaluated. In the USE-ME meta-model we decided to divide it in packages, so there is a different package for every one of the phases (*i.e.* Context, Goal, Evaluation, Interaction, Survey, Report and Utility). However, when we tried to reach the packages in EVL context we discovered a bug[1] that does not allow us to do that. For that reason all rules are specified in the same context, the USE-ME context, so we cannot show only the specific rules from one phase. In order to mitigate this problem we decided to add the development phase on the rule itself. For example, when an error occurs in *Context Specification* the rule is: *'USE-ME Context Error...'*.

On the informative message we explain what is the error/suggestion and what the user should do to fix it. Sometimes we give examples on how to fill a property, the values that are most used, etc.

5.4 Use Case Scenario

In order to clarify how the USE-ME framework works with the validation rules (see Appendix A) we are going to provide an example.

[1]Bug. Latest access: August 2017. URL: https://bugs.eclipse.org/bugs/show_bug.cgi?id=515262

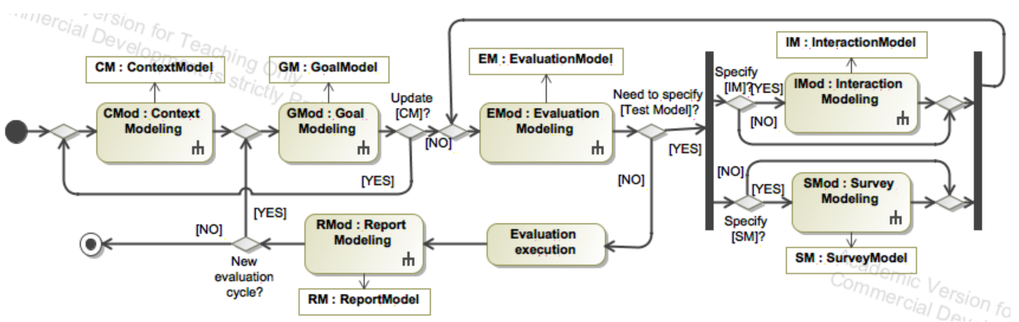
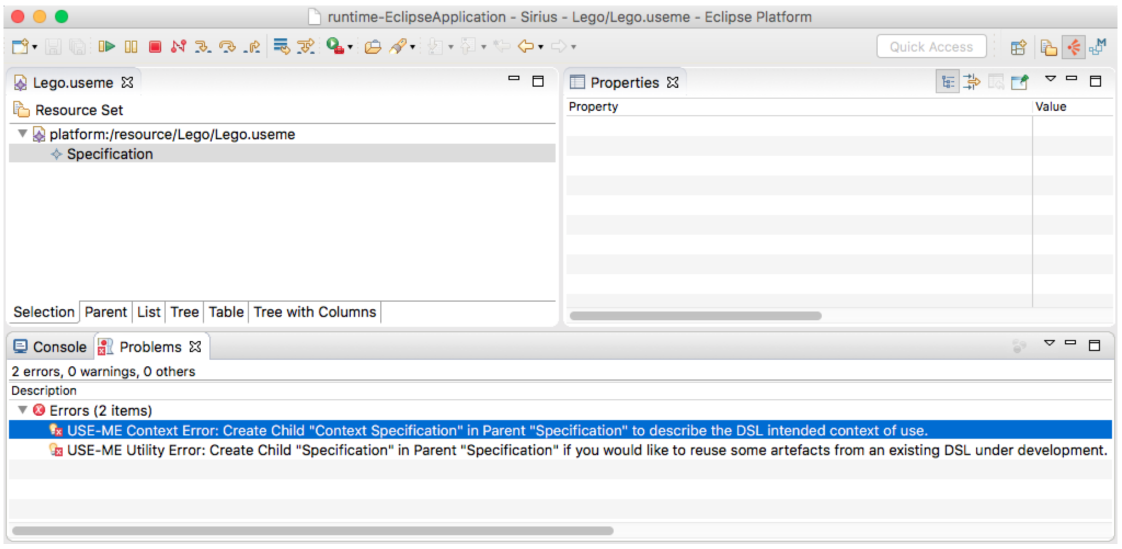


Figure 5.3: USE-ME activity diagram, taken from [Bar+17].

The first development phase of USE-ME, as we can check by analysing the USE-ME activity diagram (Figure 5.3), is the *Context Modeling* (if its the first iteration). Therefore, the first activity that should be performed by the user is to create a *Context Specification* in the USE-ME model, so that is the first feedback that the tool provides to the user. Also, since the activity is *mandatory* it appears in the feedback list as an *error*, as shown in figure 5.4:



Resource Set
platform:/resource/Lego/Lego.useme
Specification

2 errors, 0 warnings, 0 others

Errors (2 items)
USE-ME Context Error: Create Child "Context Specification" in Parent "Specification" to describe the DSL intended context of use.
USE-ME Utility Error: Create Child "Specification" in Parent "Specification" if you would like to reuse some artefacts from an existing DSL under development.

Figure 5.4: Context Specification creation.

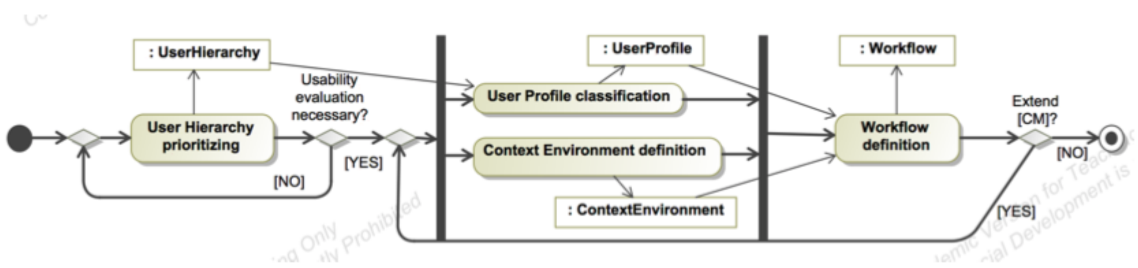

: UserHierarchy
User Hierarchy prioritizing
Usability evaluation necessary?
[YES]
[NO]
: UserProfile
User Profile classification
Context Environment definition
: ContextEnvironment
: Workflow
Workflow definition
Extend [CM]?
[NO]
[YES]

Figure 5.5: Context Modeling activity diagram, taken from [Bar+17].

When analysing the *Context Modeling* activity diagram (Figure 5.5) the next activity that should be performed by the user is the *User Hierarchy prioritizing* but, in order to do that, first the *Software Language Engineer* has to create the *Context Model*, since the *User Hierarchy* depends on the *Context Model* (Figure 5.6). Since it is a *mandatory* dependency it appears in the feedback list as an *error*.

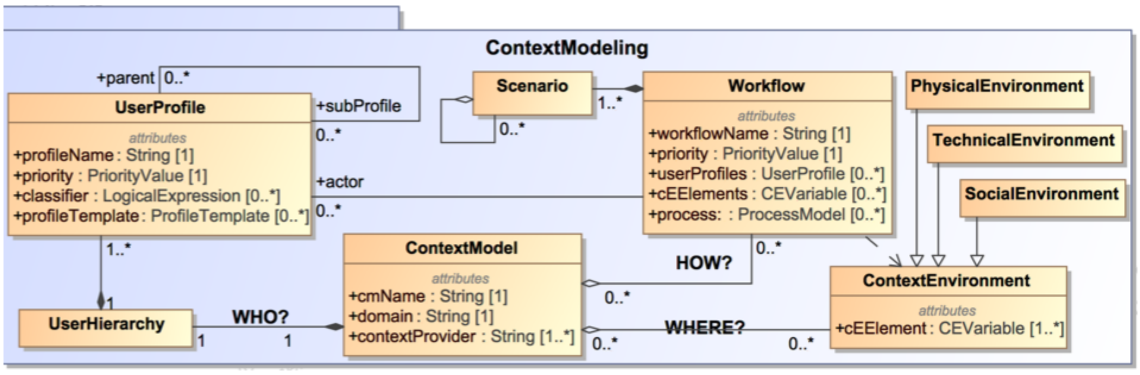
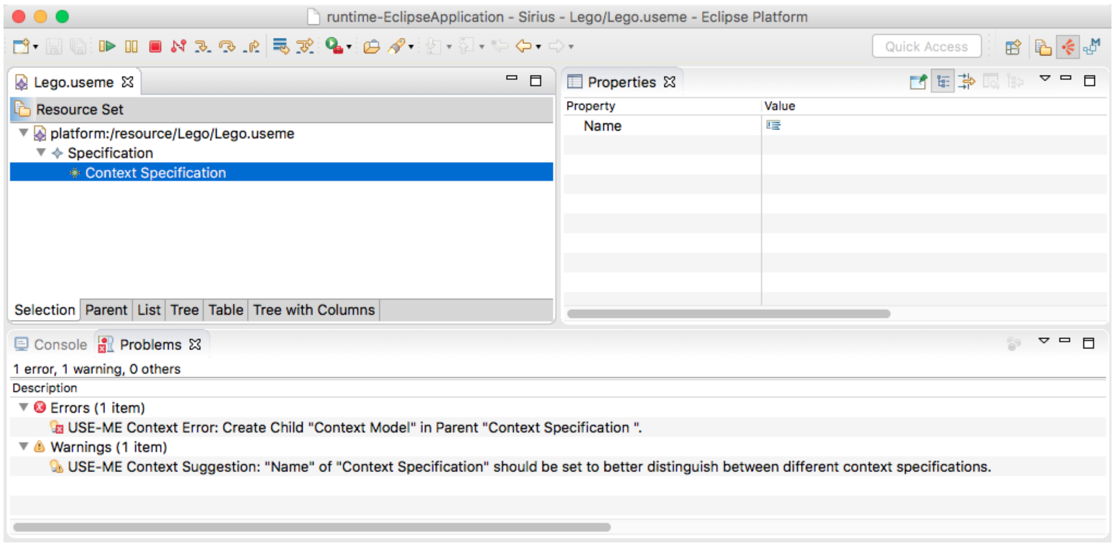

ContextModeling
+parent 0..*
UserProfile
attributes
+profileName : String [1]
+priority : PriorityValue [1]
+classifier : LogicalExpression [0..*]
+profileTemplate : ProfileTemplate [0..*]
+subProfile
0..*
+actor
0..*
1..*
1
UserHierarchy
WHO?
1
1
Scenario
0..*
1..*
Workflow
attributes
+workflowName : String [1]
+priority : PriorityValue [1]
+userProfiles : UserProfile [0..*]
+cEElements : CEVariable [0..*]
+process: : ProcessModel [0..*]
ContextModel
attributes
+cmName : String [1]
+domain : String [1]
+contextProvider : String [1..*]
HOW?
0..*
0..*
0..*
WHERE?
0..*
PhysicalEnvironment
TechnicalEnvironment
SocialEnvironment
ContextEnvironment
attributes
+cEElement : CEVariable [1..*]

Figure 5.6: Context Model class diagram, taken from [Bar+17].

When the user validates the model, after fixing the first error, a *suggestion* to add a name to the *Context Specification* also appears, so that the user distinguishes better between different *Context Specifications* since a USE-ME model can have more than one *Context Specification* because it is a iterative and incremental development process. The list of feedback messages is shown in figure 5.7:


runtime-EclipseApplication - Sirius - Lego/Lego.useme - Eclipse Platform

Lego.useme

Resource Set
- platform:/resource/Lego/Lego.useme
  - Specification
    - Context Specification

Selection | Parent | List | Tree | Table | Tree with Columns

Properties

Property	Value
Name	

Console | Problems

1 error, 1 warning, 0 others

Description
- Errors (1 item)
  - USE-ME Context Error: Create Child "Context Model" in Parent "Context Specification ".
- Warnings (1 item)
  - USE-ME Context Suggestion: "Name" of "Context Specification" should be set to better distinguish between different context specifications.

Figure 5.7: The Context Specification should have a name, and must include a Context Model.

After creating the *Context Model* the user can finally create the *User Hierarchy*, however the *User Hierarchy* is not complete until the *Software Language Engineer* specifies *Who* are the DSL stakeholders (*i.e.* DSL stakeholder, Language Engineer, Domain Expert, Language Evaluator, End User), what is the relation between them, what is their priority, etc.

The purpose of this example was to show that the USE-ME model development involves a fairly complex workflow which makes it error-prone, particularly for Software Language Engineers who are not yet experienced with the USE-ME approach, as discussed in section 3.2.3. The Validation Rules aim at providing specific feedback to mitigate this challenge. In the next chapter, we report on pilot studies conducted with the new version of the tool to assess the extent to which the Validation Rules successfully contribute to improve the USE-ME user experience.

5.5 Summary

In this chapter we discussed the implementation alternatives and why we chose to use the USE-ME framework as a starting point. We also explain what were the main problems with this framework, and provided a solution to mitigate them with validation rules. Regarding the *Validation Rules*, we presented its format, how we integrated them in the USE-ME tool, and how they provide specific feedback to the users in order to guide and validate their actions. We finished this chapter by detailing a use case scenario.

Chapter 6

Evaluation

In chapter 5 we proposed a new USE-ME framework, with Validation Rules, that guides the Software Language Engineers throughout the development process while also validating the model correctness. In this chapter we report the experiment conducted (section 6.1), including its goals (section 6.1.1), the tasks proposed (section 6.1.2), the experiment materials (section 6.1.3), the participants (section 6.1.4), the hypotheses (section 6.1.5), the design (section 6.1.6), the procedure (section 6.1.7), and the analysis (section 6.1.8). The results from the SUS and the Model Correctness assessment are then analysed, in section 6.2. In section 6.3, we evaluate the results and its implications (section 6.3.1), examined the validity of the process (section 6.3.2), and made some inferences regarding the results (section 6.3.3).

6.1 Experiment

6.1.1 Goals

We describe our two research goals using the *Goal Question Metric* research goals template [BR88]. Our first goal (G1) is to **analyse** the effect of validation rules on USE-ME, **for the purpose of** evaluation, **with respect to** its impact on the *System Usability Scale*, **from viewpoint of** researchers, **in the context of** an experiment conducted with participants with no experience with USE-ME framework at Universidade Nova de Lisboa (UNL). Our second goal (G2) is to **analyse** the effect of validation rules on USE-ME, **for the purpose of** evaluation, **with respect to** its impact on the *Model Correctness*, **from viewpoint of** researchers, **in the context of** an experiment conducted with participants with no experience with USE-ME framework at Universidade Nova de Lisboa (UNL). Since we are comparing the *System Usability Scale* (SUS) of two alternative USE-ME versions, we can break down the G1 into ten sub-goals one for each SUS question. So, the sub-goals can be obtained by replacing *System Usability Scale* with the questions from the SUS questionnaire (mentioned in section 2.3.1.1).

6.1.2 Tasks

Before starting the evaluation, each participant read and agreed with the terms of the consent letter present in the beginning of both (*i.e.* background and feedback) questionnaires (adapted from [Run+12]). After that, the participant filled the *Background questionnaire* with the demographic information (Appendix G), since only students with background knowledge on DSL development could participate in the experiment. Then, they saw a presentation on the USE-ME framework (Appendix F).

Each participant in this study had to complete two tasks: modeling a DSL development phase with the original USE-ME framework version, and modeling a DSL development phase with the new USE-ME framework version. The order of the USE-ME versions (*i.e.* original and new), the DSL development phases (*i.e.* utility, context, goal, evaluation, interaction, survey, and report), and the modeling exercise (*i.e.* Lego Mindstorms and Smart House) were randomly selected. The same participant did not use the same USE-ME version, model the same development phase and model the same exercise twice, in order to mitigate the learning effect. In both cases, we recorded the screen during the exercise execution, and saved the modeling files that were produced by the participants. We did not provide any feedback to the participant concerning whether they were able to successfully complete the tasks, in order not to affect the following exercise.

After each task, the participant filled in a *System Usability Scale* (SUS) questionnaire (Appendix H) to collect feedback about the USE-ME framework with respect to the modeling phase and exercise he had performed.

6.1.3 Experimental Materials

As mentioned in section 6.1.2, the experimental material for the evaluation included a consent letter in the beginning of each questionnaire, a *Background Questionnaire* on demographic information (Appendix G), one presentation on the USE-ME approach (Appendix F), four modeling exercises, and a feedback questionnaire about the SUS (Appendix H). We prepared four exercises: two with the original USE-ME version (Appendix B and C), and two with the new USE-ME version (Appendix D and E). Each USE-ME version had a Lego Mindstorms and a Smart House exercise, and the only difference between them was an introductory paragraph that described how the participants could validate their models. We designed modeling exercises equally and with similar complexity. Before the real experiment we also tested these materials in pilot evaluations.

6.1.4 Participants

As mentioned in section 6.1.3, we carried out pilot evaluations before the experiment. The pilot evaluations were performed by two participants: one was the USE-ME author, and one was a student representative of the experiment participants, with two different goals. The goal of the USE-ME author was to check if the *validation rules* were according to the USE-ME specifications, while the student validated the experiment materials (*i.e.* presentation, exercises and questionnaires). Our original idea was that each participant was going to model all USE-ME development phases in each exercise, but while performing the pilot evaluation we realized that it took too much time so we had to break-down the experiment to one development phase per exercise, in such a way that each participant performed two tasks (one using the original version and one using the new version) from different development phases.

The experiment was performed by 14 participants selected by convenience sampling, since each participant should have a background knowledge in *DSL* development. They were all Computer Science students at different levels at UNL. Each participant tested both USE-ME versions (*i.e.* original and new), and none of them had experience with the tool. With respect to the highest completed level of education, 8 had BSc degrees, and 6 had MSc degrees. The age of the participants was between 23 and 27. All the participants had used DSLs in academic context and two of them had also used it in industry.

We also performed a guided evaluation on a DSL related with Multi-Agent Systems (MAS). The DSL under evaluation, modelled with both USE-ME versions, was chosen since there is a ongoing Scientific and Technological Cooperation between NOVA-LINCS and Ege University International Computer Institute in Turkey. The Turkish group was composed by 4 participants: 2 Computer Science students, and 2 professors. With respect to the highest completed level of education, 1 had a BSc degree, 1 had a MSc degree, and 2 had PhD degrees.

6.1.5 Hypotheses, parameters and variables

For each goal, described in section 6.1.1, we defined the null H_0 and the alternative hypotheses H_1.

H_{0SUS}: Adding validation rules to USE-ME does not influence the *System Usability Scale*.

H_{1SUS}: Adding validation rules to USE-ME influences on the *System Usability Scale*. The independent variable is the USE-ME version, which can be *original*, or *new*. The dependent variables are the *System Usability Scale* items 2.3.1.1. Higher scores indicate better usability [Ban+08].

We followed the same approach and refined the null H_0 and the alternative hypotheses H_1 for the *Model Correctness*:

$H_{0ModelCorrectness}$: Adding validation rules to USE-ME does not influence the *Model Correctness*.

$H_{1ModelCorrectness}$: Adding validation rules to USE-ME influences on the *Model Correctness*.

6.1.6 Design

The participants used both USE-ME versions, *original* and *new*, the order was selected randomly. In order to reduce the learning effects, the user did not model the same DSL development phases (*i.e.* utility, context, goal, evaluation, interaction, survey, and report) and the same modeling exercises (*i.e.* Lego Mindstorms and Smart House) in both experiments. Seven of the fourteen participants started the experiment with the *original* version, and seven started the experiment with *new* version.

Table 6.1: Experimental design.

#	Background	Presentation	T1	Feedback-SUS	T2	Feedback-SUS
7	✓	✓	original	✓	New	✓
7	✓	✓	New	✓	original	✓

In Table 6.1 each line represents a set of participants that performed a sequence of activities. # refers to the number of participants, *Background* to the background questionnaire (*i.e.* demographic data), *Presentation* of the USE-ME approach, *T1* to the first exercise, *Feedback-SUS* to the *System Usability Scale* questionnaire relative to the first exercise, *T2* to the second exercise, *Feedback-SUS* to the *System Usability Scale* questionnaire relative to the second exercise. *original* stands for the USE-ME version without validation rules, and *New* stands for the USE-ME version with validation rules.

6.1.7 Procedure

We prepared the workspace so that all participants had similar conditions: one laptop with the USE-ME tool, one external monitor with the modeling exercise, and a mouse. In each evaluation session there was only one participant at a time. First, we introduced the USE-ME approach, and explained the tasks that the participant was going to perform. We also informed the participant that we were going to record the laptop screen while he was performing the exercise, so that we could analyse his session later. Finally, we explained that he could quit at any time, if he so desires. Then the participant read and agreed with the terms of the *Consent Letter* informing that he would freely participate in the evaluation. Each participant performed two tasks: modeling a DSL evaluation without *Validation Rules*, and with *Validation Rules*, and then filed the SUS questionnaire concerning the system usability. The tasks order varied from one participant to the next, as we already mentioned in section 6.1.6.

6.1.8 Analysis procedure

We collected descriptive statistics for *SUS* (*i.e. mean, standard deviation, minimum and maximum*) to get an overview of its values. Then we used the *Wilcoxon Signed-Rank Test* [1], this test is a nonparametric test equivalent to the *Paired Samples t Test*. However, the *Wilcoxon Signed-Rank Test* does not assume normality in the data and since the *SUS* provides ordinal data data for each score, we cannot assume normality. The *Wilcoxon Signed-Rank Test* is used to compare two sets of scores that come from the same participants, so we are going to compare the values from the participants that performed the tests *original-new* and *new-original*.

6.2 Results and Analysis

6.2.1 SUS

6.2.1.1 Descriptive statistics

In table 6.2 we present the descriptive statistics for the *SUS* organized by experiments. As we mentioned the participants could start the experiment with the *original* version and then use the *new* version, or the contrary. So we analysed the *SUS* results separately (*i.e. original-new* and *new-original*.) In the *Version* column we specify which of the USE-ME versions we are considering, *original* stands for the USE-ME version without validation rules and *new* stands for USE-ME version with validation rules. We also present the mean, standard deviation, minimum, and the *maximum* for the *SUS*. By analysing our data set we can assert that in both experiments, *original-new* (*i.e.* 82,85>57,14) and *new-original* (*i.e.* 79,64>57,85), the *SUS* mean value is always higher in the *new* version.

Table 6.2: SUS descriptive statistics organized by experiments.

	Version	**#**	**Mean**	**S.D.**	**Minimum**	**Maximum**
orig-new	original	7	57,14	15,50	37,50	77,50
	new	7	82,85	8,34	70,00	92,50
new-orig	new	7	79,64	17,10	45,00	100,00
	original	7	57,85	20,48	32,50	92,50

In table 6.3 we describe the results from *Wilcoxon Signed-Rank Test* (*i.e. negative rank, positive rank, and the significance*). When analysing the results from the experiment *original-new*, we notice that all participants (*i.e.* positive rank = 7) have attributed a higher *SUS* score for the *new* version. However, when analysing the results from the experiment

[1]Wilcoxon Signed-Rank Test. Latest access: September 2017. URL: https://statistics.laerd.com/spss-tutorials/wilcoxon-signed-rank-test-using-spss-statistics.php

Table 6.3: Wilcoxon Signed-Rank Test results organized by experiments.

Version	#	Neg. Rank	Pos. Rank	Sig
original-new	7	0	7	,018
new-original	7	1	6	,034

new-original we noticed that one participants (*i.e.* negative rank = 1) assigned a higher score to the *original* version, and six participants (*i.e.* positive rank = 6) assigned a higher score to the *new* version.

In table 6.4 we present the descriptive statistics for the *SUS* organized by versions. We calculated the mean, standard deviation, minimum, and the *maximum* for the *SUS*. By analysing our data set we can assert that in both versions, the *SUS* mean value is higher in the *new* version than the *original* version, as we already noticed in table 6.2.

Table 6.4: SUS descriptive statistics organized by versions.

Version	#	Mean	S.D.	Minimum	Maximum
original	14	57,50	17,45	32,50	92,50
new	14	81,25	13,03	45,00	100,00

Table 6.5: Wilcoxon Signed-Rank Test results organized by versions.

Version	#	Neg. Rank	Pos. Rank	Sig
total	14	1	13	,001

In table 6.5 we describe the results from *Wilcoxon Signed-Rank Test* (*i.e. negative rank, positive rank, and the significance*). When analysing the results we noticed that $\frac{13}{14}$ of the participants prefer the *new* version, and $\frac{1}{14}$ of the participants prefer the *original* version. To sum up, we can conclude that the majority opts for the *new* version.

6.2.1.2 Hypotheses testing

RQ1: Do validation rules improve the System Usability Scale of the USE-ME framework?

We summarise in table 6.3 and in table 6.5 the results of the *Wilcoxon Signed-Rank Test* for the *SUS*. The *Wilcoxon Signed-Rank Test* (see table 6.3) indicated that the *new* SUS scores were statistically significantly higher than the *original* SUS scores, in the *original-new* (Z=7; p-value=,018) and in the *new-original* experiment (Z=7; p-value=,034). Also, in the aggregated *Wilcoxon Signed-Rank Test* that compared the results by versions, and not by experiments as we mentioned before, indicated that the *new* (Mean=81,25) SUS scores were statistically significantly higher than the *original* (Mean=57,50) SUS scores (Z=14; p-value=,001). So, we can reject the null hypothesis and accept the alternative one, the addition of *validation rules* influence the *SUS* score of the USE-ME tool for better.

6.2.1.3 Difference in SUS scores

In order to check if there was a difference in the *SUS* results depending on the order that the experiments took place (*i.e.* original-new and new-original) we decided to analyse in more detail the *SUS* values. In figure 6.1 we have the results from the participants that started the experiment with the *original* version (*i.e.* V1) and then used the *new* version (*i.e.* V2). In figure 6.2 we have the results from the participants that started the experiment with the *new* version (*i.e.* V2) and then used the *original* version (*i.e.* V1).

The difference values were calculated by subtracting the V1 values to the V2 corresponding values, since V2 values were often higher. The *SUS* has half positive (*i.e.* 1,3,5,7 and 9) and half negative (*i.e.* 2,4,6,8 and 10) items, so some of the difference results tend to be positive and others negative. The positive values are represented in a scale from 1 to 3 (*i.e.* the green marks), and the negative values are represented in a scale from -4 to -1 (*i.e.* the red marks). The zero values mean that the participant did not find any difference between versions.

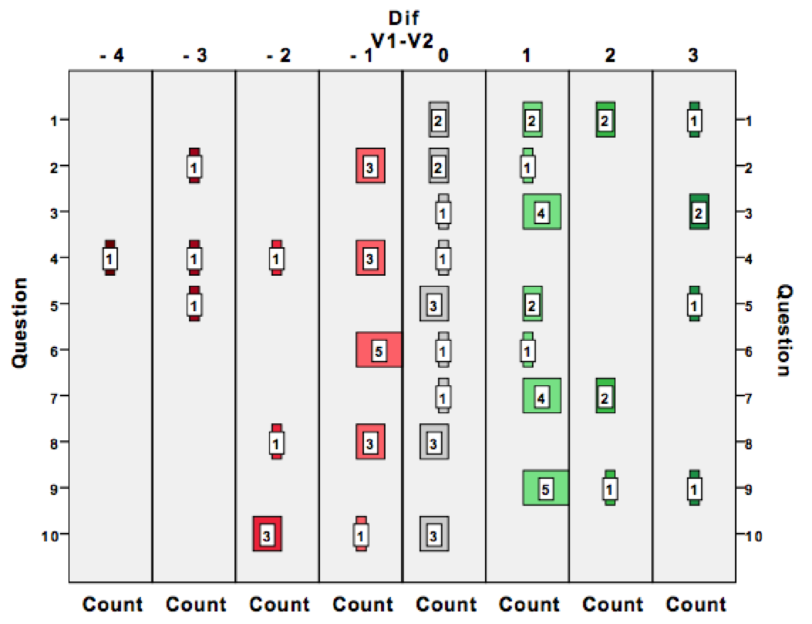


Figure 6.1: Results of the difference original-new.

1. I think that I would like to use this system frequently.

In figure 6.1, $\frac{2}{7}$ of the participants that used V1-V2 found 1 value of difference, $\frac{2}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. When analysing the results we found that the participants that used V1-V2 set a bigger difference between the versions than the V2-V1 participants. To sum up, $\frac{10}{14}$ of the participants would like to use V2 more than V1, and $\frac{4}{14}$ found no differences between versions.

2. I found the system unnecessarily complex.

In figure 6.1, $\frac{3}{7}$ of the participants that used V1-V2 found 1 value of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. When analysing the results we found that the participants that used V1-V2 set a bigger difference between the versions than the V2-V1 participants. To sum up, $\frac{7}{14}$ of the participants found that the V2 was not unnecessarily complex, $\frac{5}{14}$ found no differences between versions, and $\frac{2}{14}$ of the participants found that the V2 was unnecessarily complex.

3. I thought the system was easy to use.

In figure 6.1, $\frac{4}{7}$ of the participants that used V1-V2 found 1 value of difference, and $\frac{2}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{11}{14}$ of the participants thought that V2 was easier to use than V1, and $\frac{3}{14}$ found no differences between versions.

4. I think I would need the support of a technical person to be able to use this system.

In figure 6.1, $\frac{3}{7}$ of the participants that used V1-V2 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{1}{7}$ found 4 values of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. In figure 6.2, $\frac{4}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{2}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. When analysing the results we found that the participants that used V1-V2 set a bigger difference between the versions than the V2-V1 participants. To sum up, $\frac{11}{14}$ of the participants do not need the support of a technical person to be able to use V2, $\frac{1}{14}$ found no differences between versions, and $\frac{2}{14}$ of the participants think they would need the support of a technical person to be able to use V2.

5. I found the various functions in this system were well integrated.

In figure 6.1, $\frac{2}{7}$ of the participants that used V1-V2 found 1 value of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{1}{7}$ found 3 negative values of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, and $\frac{1}{7}$ found 2 values of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. When analysing the results we found that the participants that used V1-V2 set a bigger difference between the versions than the V2-V1 participants. To sum up, $\frac{7}{14}$ of the participants found the various functions of V2 were better integrated than V1, $\frac{6}{14}$ found no differences between versions, and $\frac{1}{14}$ of the participants found the various functions of V1 were better integrated than V2.

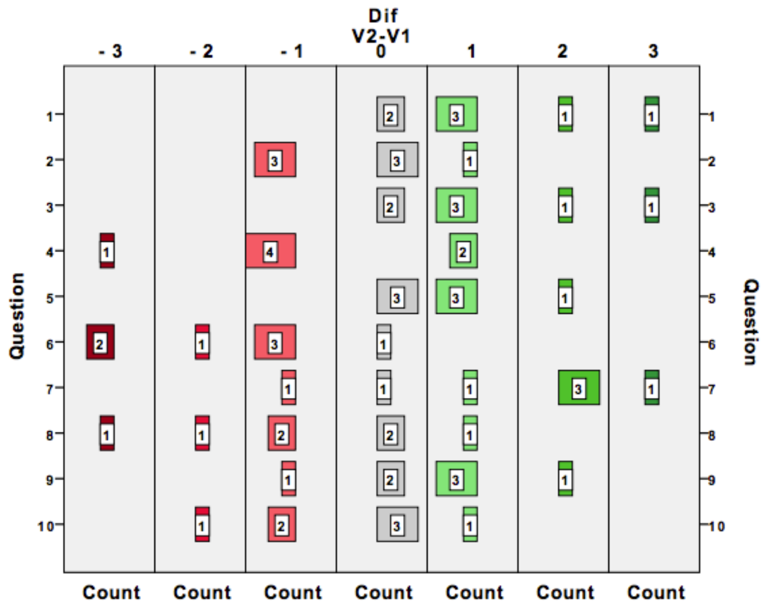


Figure 6.2: Results of the difference new-original.

6. I thought there was too much inconsistency in this system.

In figure 6.1, $\frac{5}{7}$ of the participants that used V1-V2 found 1 value of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{2}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{11}{14}$ of the participants thought that there was too much inconsistency in V1 more than in V2, $\frac{2}{14}$ found no differences between versions, and $\frac{1}{14}$ of the participants thought that there was too much inconsistency in V2 more than in V1.

7. I would imagine that most people would learn to use this system very quickly.

In figure 6.1, $\frac{4}{7}$ of the participants that used V1-V2 found 1 value of difference, and $\frac{2}{7}$ found 2 values of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. In figure 6.2, $\frac{1}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{3}{7}$ found 2 values of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{1}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{11}{14}$ of the participants imagine that most people would learn to use V2 quicker than V1, $\frac{2}{14}$ found no differences between versions, and $\frac{1}{14}$ of the participants imagines that most people would learn to use V1 quicker than V2.

8. I found the system very cumbersome to use.

In figure 6.1, $\frac{3}{7}$ of the participants that used V1-V2 found 1 value of difference, and $\frac{1}{7}$ found 2 values of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. In figure 6.2, $\frac{2}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, $\frac{1}{7}$ found 3 values of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{8}{14}$ of the participants found that V1 was more cumbersome to use than V2, $\frac{5}{14}$ found no differences between versions, and $\frac{1}{14}$ of the participants found that V2 was more cumbersome to use than V1.

9. I felt very confident using the system.

In figure 6.1, $\frac{5}{7}$ of the participants that used V1-V2 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 3 values of difference between V2 and V1 in the ordinal scale. In figure 6.2, $\frac{3}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{2}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{11}{14}$ of the participants felt more confident using V2 than V1, $\frac{2}{14}$ found no differences between versions, and $\frac{1}{14}$ felt more confident using V1 than V2.

10. I needed to learn a lot of things before I could get going with this system.

In figure 6.1, $\frac{1}{7}$ of the participants that used V1-V2 found 1 value of difference, and $\frac{3}{7}$ found 2 values of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. In figure 6.2, $\frac{2}{7}$ of the participants that used V2-V1 found 1 value of difference, $\frac{1}{7}$ found 2 values of difference, and $\frac{1}{7}$ found 1 negative value of difference between V2 and V1 in the ordinal scale. $\frac{3}{7}$ found no differences between versions. When analysing the results we found that the participants that used V2-V1 set a bigger difference between the versions than the V1-V2 participants. To sum up, $\frac{7}{14}$ thought they will needed to learn a lot of things before using V1 more than V2, $\frac{6}{14}$ found no differences between versions, and $\frac{1}{14}$ thought he will needed to learn a lot of things before using V2 more than V1.

To sum up, we did not find any significant impact on the *SUS* score regarding the order in which the participants performed the experiment.

6.2.2 Model Correctness

In section 6.1.1 we have defined another goal, G2, regarding the *Model Correctness* of the models produced with *original* version *(i.e. without validation rules)* and with the *new* version *(i.e. with validation rules)*. In table 6.6 we present the results of our inspection on the models produced. As already mentioned in section 6.1.2, the same participant did not use the same USE-ME version, model the same development phase and model the same exercise twice, in order to mitigate the learning effect. The results are grouped according to the order of the experiment, so original-new and new-original. In table 6.6 each line (*e.g.* the line highlighted in grey) represents one participant. For each participant we present the *development phase* (*i.e.* context, goal, evaluation, interaction, survey, report, and utility), the *errors* (*i.e.* the total of errors from the development phase), and the *warnings* (*i.e.* the total of warnings from the development phase). The modeling exercises (*i.e.* Lego Mindstorms and Smart House) are omitted from the list since they are similar.

Table 6.6: List of errors and warnings found in the models inspection.

original-New						**New-original**					
original			**New**			**New**			**original**		
DV.	Err.	Warn.	**DV.**	Err.	Warn.	**DV.**	Err.	Warn.	**DV.**	Err.	Warn.
I	1	4	**S**	0	0	**U**	0	0	**E**	7	9
U	0	5	**G**	0	0	**C**	0	0	**R**	4	7
E	7	9	**U**	0	0	**E**	0	0	**G**	0	0
S	2	8	**R**	0	0	**I**	0	0	**C**	12	5
R	4	7	**C**	0	0	**R**	0	0	**S**	2	8
G	0	2	**E**	0	0	**G**	0	0	**U**	0	0
C	12	5	**I**	0	0	**S**	0	0	**I**	0	0

As mentioned in section 5.3, *errors* indicate critical deficiencies, while *warnings* indicate non-critical issues, suggestions. So despite not having impact on the *model correctness warnings* help users to make decisions.

When analsying the table 6.6 we noticed that the models that were modelled with the *new* version do not have any errors and/or warnings, but when we analyse the models produced with the *original* version almost all models have errors and/or warnings and there are the same between modeling phases. The modeling exercises for both versions are similar, however, we omitted some details and we alerted the participants for this aspect at the beginning of each exercise. By omitting some details we were trying to understand if the user had learned anything from the first modeling exercise to the second one, and if so we would notice it by the list of errors and/or warnings. However, since the modeling phases were different it would be harder for the user to immediately understand what was missing. The participants that started with the *original* version and then used the *new* version learned less than the participants that started with the *new* version and then used the *original* version, as we can check by analysing the lines highlighted in blue in table 6.6. These particular participants wrote in *Feedback Questionnaire* (Appendix H) that "I don't know if it is right and if have done everything", so despite having learned from the previous exercise they did not feel confident enough without the validator "with the validator I know what is missing". The participants that started with the *original* version were less influenced by the learning effect, since they did not have used the version with the validation rules, but when they finished the second exercise with the *new* version they found that the first model was not correct "my first model is not correct a lot of things are missing".

6.3 Discussion

6.3.1 Evaluation of Results and Implications

RQ1: Do validation rules improve the System Usability Scale of the USE-ME framework?

We found evidence of improvements brought by the addition of *validation rules* (see section 6.2.1), in terms of the *SUS* with which our participants performed their *modeling tasks*. The *Wilcoxon Signed-Rank Test* indicated a statistically significant difference observed when we compare the results from the *SUS* questionnaire between experiments (*i.e. original-new* and *new-original*), and between versions (*i.e. original and new*). The *SUS* mean scores were higher in the *original-new* experiment, 57,14 and 82,85 respectively, with a *p-value*=,018. A similar statistically difference occurred in the *new-original* experiment, 57,85 and 79,64 respectively, with a *p-value*=,034. When we aggregated the results of the *SUS* by versions we found a greater statistically significant difference with a *p-value*=,001. That said, the results seem to convey a lower usability for the USE-ME framework without the *validation rules* than with *validation rules*. So, since the exercises only differed on the presence of *validation rules* or not, we can reject the null hypothesis and accept the alternative one (section 6.1.5), and affirm that the addition of *validation rules* to USE-ME influenced the *System Usability Scale*.

RQ2: Do validation rules improve the Model Correctness of the USE-ME framework?

We found evidence of improvements brought by the addition of *validation rules* (see section 6.2.2), in terms of the *Model Correctness* with which our participants performed their *modeling tasks*. We found a significant difference, regarding the *Model Correctness*, between the models produced with the *original* version and with the *new* version. Almost all the models developed with the *original* version had errors/warnings, since we choose to omit some details from the exercise, in order to see how the participants would react to such omissions. In contrast, all the models produced with the *new* version were correct. As we predicted the users that started the experiment with the *original* version learned less than the participants that start the experiment with the *new* version (see section 6.2.2). The main reason for that gap was the use of *validation rules* in the first exercise, which in our interpretation, educated the participants more, since they guided, suggested and validated the participants actions. The feedback from the users suggested that they felt more confident after using the *validation rules*, since they never used the tool before. So, since the modeling exercises only differed on the presence of *validation rules* or not, we can reject the null hypothesis and accept the alternative one (section 6.1.5), and affirm that the addition of *validation rules* to USE-ME influenced the *Model Correctness*.

6.3.2 Threats to Validity

The validity of our results strongly depends on factors in the experiment settings. We analyse three types of threads to validity [Woh+12]:

Conclusion validity: we had a low number of participants in the experiment compared with another experiments in the area of *Software Engineering*, and this aspect should improve in further experiments with this tool. The number and the selection of participants represent a threat since the USE-ME tool should be used by people that already have a solid knowledge in DSL development and the participants were mainly novices.

Internal validity: there is a potential learning effect from the first exercise to the second one. As we mentioned in 6.2.2 we noticed that the participants that performed the exercises *new-original* were more affected by the learning effect than the *original-new* participants, but that result was already expected, since they used the tool version with the *validation rules*.

External validity: despite the participants had some knowledge about DSLs they were mainly novices. So, there is a need to validate this approach with more experienced users. The USE-ME tool is complex and none of the participants used it before and the time was limited. We did not want to assess the user modeling skills, so we provided a guided exercise since learning to use the tool and model the exercise at the same time was too complex. Any experimental evaluation design has its own associated threats and ours is no exception. Our decisions concerning the participants profile, and quantity, the tasks they performed, etc., all entail some external validity threats (e. g. the results could be potentially different for more experienced DSL developers). As such, we recommend that the approach proposed in this dissertation should also be independently validated, through replications of our own evaluation that could be adapted to tackle different external threats (e.g. by having experienced DSL developers, instead of novices). To facilitate such replications, the experimental materials used in this evaluation are available from appendix B to appendix I.

6.3.3 Inferences

The results obtained suggest that the *new* version of the USE-ME framework, with *validation rules*, improved the *System Usability Scale* and the *Model Correctness* of the tool and of the models, respectively. Regarding the *SUS* score, as mentioned in section 3.2.3, USE-ME had already been tested with students with no expertise in DSL evaluation, and the results showed that they did not feel confident while using the tool because it was "complex" and had "too many steps to follow" (this experience was guided). So, we had to find a solution that mitigate this usability problem. We developed *validation rules* that allow to guide, suggest and validate user actions, so that users felt more confident while using the tool. During the design of the *validation rules* we decided to incorporate in the feedback messages information about the process, in order to educate the user and to help with the suggestion rules, so that the user could decided whether or not take the suggestion. After analysing the results from the questionnaires we found that the difference regarding the *SUS* score between versions is statistically significant, so we can reject the null hypothesis and accept the alternative one, that the addition of *validation rules* improved the *SUS* of the USE-ME framework.

Regarding the *Model Correctness*, no previous experience has been conducted on this topic. However, since the *validation rules* validated the user actions, so they alerted the user for errors/warnings we expected that the number of errors/warnings decreased in the *new* version of the tool. Also, since some users started the experiment with the *original* version and others started the experiment with *new* version, we were able to observe that the users that started with the *new* version learned more (regarding the evaluation process) than the users that started with the *new* version. When we designed the experiment we omitted information from the exercise so that we could observe how the users would behave with the lack of information. The participants that used the *new* version first did not notice that some *associations* between objects were missing, on the contrary, the participants that used the *original* version first noticed that some *associations* were missing since the *validation rules* alerted them, so they were able to produce a correct model with the help of the *validation rules*. After analysing the models, produced by the participants, we found that the difference regarding the *Model Correctness* between versions is significant, since almost all the models produced with the *original* version had errors/warnings and none of the models produced with the *new* version had errors/warnings. So, we can reject the null hypothesis and accept the alternative one, that the addition of *validation rules* improved the *Model Correctness* of the USE-ME framework.

Despite the significant results is necessary to evaluate the *SUS* and the *Model Correctness*, during real DSLs evaluation and with more experienced users. So that, we can compare the results and improve the USE-ME framework.

6.4 SEA-ML Guided Evaluation

As we mentioned in section 6.1.4, we performed a guided evaluation on a DSL related with Multi-Agent Systems (MAS). We executed this experiment since there is a ongoing Scientific and Technological cooperation between NOVA-LINCS and Ege University in Turkey. SEA-ML [2] is a domain-specific modeling language used to model and generate architectural artefacts for Multi-agent Systems (MAS) especially working on the Semantic Web". SEA-ML was modelled using the *original* and the *new* version of the USE-ME framework. The participants (2 Computer Science students and 2 professors) used the USE-ME to modelled previous experiments, and to prepare the tool for upcoming evaluations. So, since 2 participants (1 student and 1 professor) used the *original* and 2 participants (1 student and 1 professor) used the *new* version we cannot draw conclusions regarding this evaluation. However, after comparing usability experiences the group said that the *new*

[2]SEA-ML. Latest access: September 2017. URL: `http://serlab.ube.ege.edu.tr/resources.htmlSEA_ML`

version was easier to learn, since the *validation rules* educated the user, and also alerted for errors/warnings. They also pointed out that if the rules were marked in the tree model it would be easier for the user to locate the errors/warnings.

6.5 Summary

In this chapter we planned and executed an experiment in order to compare the two versions of the USE-ME framework. 14 participants performed the modeling exercises, using both versions, the *original* and the *new*. The data collected showed that the *new* version improved the *System Usability Scale*, and the *Model Correctness* of USE-ME models. Our explanation for this results are the *validation rules*, that add to the USE-ME tool guidance, suggestion and validation, which allow to help *SLE* during the DSL evaluation process. Despite the significant results is necessary to evaluate the *SUS* and the *Model Correctness*, during real DSLs evaluation and with more experienced users. So that, we can compare the results and improve the USE-ME framework.

Chapter 7

Conclusions

In the previous chapters we described our problem, designed a solution, implemented it, and tested it with potential users. In this chapter we sum up the main contributions of this dissertation 7.1, the limitations of our solution 7.2, and we propose future work in order to improve the USE-ME framework 7.3.

7.1 Contributions

The main contribution of this dissertation is an updated version of the USE-ME framework that allows to support the *SLE* in one of the most important steps of the *Software Language Engineering*, which is the evaluation (*i.e.* DSL evaluation). By supporting the *SLE* we hope to mitigate the risk of building inappropriate languages that often decrease users productivity, since the languages do not fit the end users, or that increase the maintenance costs since they often cannot be reused. We establish a process for the experimental validation of DSLs, with *validation rules*, during the development process, that validates, suggests, based on the *Software Language Engineers* actions, and that guides this stakeholder throughout a complex and error-prone DSL evaluation workflow. We were able to increase the *System Usability Scale* of USE-ME, and the *Model Correctness* of the USE-ME models, so we hope that *Software Language Engineers* feel more confident while using this *new* version of the framework, as our participants felt, and also more aware of the importance of assessing DSLs since the early stages of development.

We also present other important contributions, such as:

- a study and an analysis on experimental evaluations of Domain Specific Languages, and a comparison between these approaches;

- a study and an analysis on wizards/tools that are concerned with user guidance, the quality of the models produced, increasing productivity of the end user, and to decrease the maintenance costs;
- a discussion on the implementation alternatives, a solution and an architecture for the prototype;
- an experiment planning, conduction, and analysis regarding the usability (measured with *SUS*) and the *Model Correctness* of the USE-ME and the *Model Correctness* of the USE-ME models.

7.2 Limitations

One of the biggest limitations, regarding our solution, is that *validation rules* were developed to support the USE-ME meta-model, so they cannot be directly used in other applications. However, the knowledge that we were able to gather, from our solution, can be reused by other developers when they have validation, suggestions or guidance problems in their frameworks.

Since the USE-ME tool was developed with Eclipse technologies (section 3.2.1) we were restricted regarding the rules appearance. We wanted a more *continuous validation* approach, but this kind of *live validation*, the term used in eclipse for showing the Ecore validation messages, was not available for EVL integrations. So because of that, we had to change the appearance of the messages so that the user could easily locate the errors/-suggestions in the model. Also, because of this limitation the user needs to validate the model, every time he makes changes, to check the messages produced by the *validation rules*.

As we mentioned in chapter 3, the USE-ME meta-model is not flattered (*i.e.* each development phase has its own Ecore inside one specific package) so because of that the *validation rules* do not disappear when the user decides not to accept a suggestion given by the tool.

7.3 Future Work

For the future work, we intend to find ways to improve the USE-ME framework, with the *validation rules* proposed in this dissertation.

One of the aspects that could be improved in the USE-ME framework is the language meta-model (*i.e.* the ecore), since the *original* one could be more specific (*i.e.* introduce more restrictions in the ecore) regarding some objects in the USE-ME models. We cannot express all the language restrictions through the meta-model, however, the language meta-model should contain all the possible restrictions as this would allow to decrease the *validation rules* list. Also, Eclipse allows to *live validate* (*i.e.* feature that allows to validate the model continuously) the rules expressed in the Ecore, and in our opinion this would improve the user experience since the user would not need to validate the model every time he makes changes. However, this feature does not allow to guide the SLE in the process. It only works to validate the USE-ME models. For example, a few restrictions that could be expressed in the USE-ME meta-model are related with the User Hierarchy.

The USE-ME approach suggests that users create five stakeholders: *DSL Stakeholder*, *Domain Expert*, *Expert Evaluator*, *Language Engineer*, and *End User*. The *DSL Stakeholder* is the parent of all the other stakeholders, and each stakeholder has different priorities (already defined in the USE-ME approach) regarding the DSL developed. So, if we could express this suggestion through the language meta-model it would be easier for the users to understand the USE-ME stakeholders hierarchy. And also, this could be expressed graphically which leads to our next improvement. As we mentioned in section 3.2.1, Sirius allows the creation of custom graphical workbenches and it is already used in the USE-ME framework. However, the user does not know when he can generate Sirius models, or how to generate the models from the *Package Explorer*. So we propose, in further iterations of the USE-ME framework, to explore the integration between those two tools. Our idea is that USE-ME could have two views: one tree model (the original and current one), and one graphical view (with Sirius). The *validation rules* could easily be replicated to the Sirius validation language, AQL, which is similar to OCL and as such similar to EVL. So, users could use the tree editor to get an overview of the process, which is better for guidance between development phases, and use the graphical editor to generate the models. Also, the icons that are currently used in Sirius were not tested with users, so if the integrations between these two tools got better we suggest that the icons usability should be tested.

Bibliography

[924te] I. 9241-11. *Usability*. https://www.iso.org/obp/ui. Latest access: August 2017.

[Asste] U. E. P. Association. *How Low Can You Go? Is the System Usability Scale Range Restricted?* https://uxpa.org/jus/article/how-low-can-you-go-system-usability-scale-range-restricted. Latest access: August 2017.

[Ban+08] A. Bangor, P. T. Kortum, and J. T. Miller. "An Empirical Evaluation of the System Usability Scale". In: *International Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594. DOI: 10.1080/10447310802205776. URL: http://dx.doi.org/10.1080/10447310802205776.

[Ban+09] A. Bangor, P. Kortum, and J. Miller. "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale". In: *J. Usability Studies* 4.3 (May 2009), pp. 114–123. ISSN: 1931-3357. URL: http://dl.acm.org/citation.cfm?id=2835587.2835589.

[Bar+12a] A. Barišić, V. Amaral, and M. Goulão. "Usability Evaluation of Domain-Specific Languages". In: *2012 Eighth International Conference on the Quality of Information and Communications Technology*. 2012, pp. 342–347. DOI: 10.1109/QUATIC.2012.63.

[Bar+12b] A. Barišić, V. Amaral, M. Goulão, and B. Barroca. "How to reach a usable DSL? Moving toward a Systematic Evaluation". In: *Electronic Communications of the EASST* 50 (2012).

[Bar+17] A. Barišić, V. Amaral, and M. Goulão. "Usability Driven DSL development with USE-ME". In: *Computer Languages, Systems & Structures* (2017). ISSN: 1477-8424. DOI: http://dx.doi.org/10.1016/j.cl.2017.06.005. URL: http://www.sciencedirect.com/science/article/pii/S1477842417300477.

[BR88] V. R. Basili and H. D. Rombach. "The TAME project: towards improvement-oriented software environments". In: *IEEE Transactions on Software Engineering* 14.6 (1988), pp. 758–773. ISSN: 0098-5589. DOI: 10.1109/32.6156.

[Bro+96] J. Brooke et al. "SUS-A quick and dirty usability scale". In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[Dow+05] S. Dow, B. MacIntyre, J. Lee, C. Oezbek, J. D. Bolter, and M. Gandy. "Wizard of Oz support throughout an iterative design process". In: *IEEE Pervasive Computing* 4.4 (2005), pp. 18–26. ISSN: 1536-1268. DOI: 10.1109/MPRV.2005.93.

[Ecltea] Eclipse. *Eclipse Modeling Framework*. https://www.eclipse.org/modeling/emf. Latest access: August 2017.

[Eclteb] Eclipse. *Eclipse Modeling Tools*. https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neonr. Latest access: August 2017.

[Ecltec] Eclipse. *Epsilon*. https://www.eclipse.org/epsilon. Latest access: August 2017.

[Eclted] Eclipse. *Epsilon Validation Language*. https://www.eclipse.org/epsilon/doc/evl. Latest access: August 2017.

[Firte] U. First. *Introduction to User-Centered Design*. http://www.usabilityfirst.com/about-usability/introduction-to-user-centered-design. Latest access: August 2017.

[Fow10] M. Fowler. *Domain-specific languages*. Pearson Education, 2010.

[Gra+06] G. Grau, X. Franch, and S. Avila. "J-PRiM: A Java Tool for a Process Reengineering i* Methodology". In: *14th IEEE International Requirements Engineering Conference (RE'06)*. 2006, pp. 359–360. DOI: 10.1109/RE.2006.36.

[Gyrte] L. Gyr. *Jacob Nielsen usability attributes*. http://eu.landisgyr.com/better-tech/usability-is-a-key-element-of-user-experience. Latest access: August 2017.

[HM07] Ø. Haugen and P. Mohagheghi. "A multi-dimensional framework for characterizing domain specific languages". In: *Proceeding of the 7th OOPSLA Workshop on Domain Specific Modeling*. 2007.

[KB15] G. Kahraman and S. Bilgen. "A framework for qualitative assessment of domain-specific languages". In: *Software & Systems Modeling* 14.4 (2015), pp. 1505–1526. ISSN: 1619-1374. DOI: 10.1007/s10270-013-0387-8. URL: https://doi.org/10.1007/s10270-013-0387-8.

[KP09] S. Kelly and R. Pohjonen. "Worst Practices for Domain-Specific Modeling". In: *IEEE Software* 26.4 (2009), pp. 22–29. ISSN: 0740-7459. DOI: 10.1109/MS.2009.109.

[KT08] S. Kelly and J.-P. Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[Kle08] A. Kleppe. *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education, 2008.

[Kol+10] D. Kolovos, L. Rose, R. Paige, and A. García-Domínguez. “The epsilon book”. In: *Structure* 178 (2010), pp. 1–10.

[Kü+10] S. Kühne, H. Kern, V. Gruhn, and R. Laue. “Business process modeling with continuous validation”. In: *Journal of Software Maintenance and Evolution: Research and Practice* 22.6-7 (2010), pp. 547–566. ISSN: 1532-0618. DOI: 10.1002/smr.517. URL: http://dx.doi.org/10.1002/smr.517.

[Liu+02] W. Liu, S. Easterbrook, and J. Mylopoulos. “Rule-based detection of inconsistency in UML models”. In: *Workshop on Consistency Problems in UML-Based Software Development*. Vol. 5. 2002.

[Mer+05] M. Mernik, J. Heering, and A. M. Sloane. “When and How to Develop Domain-specific Languages”. In: *ACM Comput. Surv.* 37.4 (Dec. 2005), pp. 316–344. ISSN: 0360-0300. DOI: 10.1145/1118890.1118892. URL: http://doi.acm.org/10.1145/1118890.1118892.

[Nie94] J. Nielsen. *Usability engineering*. Elsevier, 1994.

[NW06] D. G. Novick and K. Ward. “Why Don’T People Read the Manual?” In: *Proceedings of the 24th Annual ACM International Conference on Design of Communication*. SIGDOC ’06. Myrtle Beach, SC, USA: ACM, 2006, pp. 11–18. ISBN: 1-59593-523-1. DOI: 10.1145/1166324.1166329. URL: http://doi.acm.org/10.1145/1166324.1166329.

[PW02] L. Padgham and M. Winikoff. “Prometheus: A Methodology for Developing Intelligent Agents”. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. AAMAS ’02. Bologna, Italy: ACM, 2002, pp. 37–38. ISBN: 1-58113-480-0. DOI: 10.1145/544741.544749. URL: http://doi.acm.org/10.1145/544741.544749.

[Pal+12] J. C. Palmeira, D. A. C. Neto, and D. R.P. C. do Nascimento. “PETIC Wizard Proposal: A Software Tool for Support PETIC Methodology”. In: *Proceedings of the 6th Euro American Conference on Telematics and Information Systems*. EATIS ’12. Valencia, Spain: ACM, 2012, pp. 407–411. ISBN: 978-1-4503-1012-3. DOI: 10.1145/2261605.2261672. URL: http://doi.acm.org/10.1145/2261605.2261672.

[Rab+12] R. Rabiser, P. Grünbacher, and M. Lehofer. “A Qualitative Study on User Guidance Capabilities in Product Configuration Tools”. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ASE 2012. Essen, Germany: ACM, 2012, pp. 110–119. ISBN: 978-1-4503-1204-2. DOI: 10.1145/2351676.2351693. URL: http://doi.acm.org/10.1145/2351676.2351693.

[Rad+90] J. Radatz, A. Geraci, and F. Katki. “IEEE standard glossary of software engineering terminology”. In: *IEEE Std* 610121990.121990 (1990), p. 3.

[RR00] J. Robbins and D. Redmiles. "Cognitive support, UML adherence, and XMI interchange in Argo/UML". In: *Information and Software Technology* 42.2 (2000), pp. 79 –89. ISSN: 0950-5849. DOI: http://dx.doi.org/10.1016/S0950-5849(99)00083-X. URL: http://www.sciencedirect.com/science/article/pii/S095058499900083X.

[Run+12] P. Runeson, M. Host, A. Rainer, and B. Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[Sin+10] G. Sinha, R. Shahi, and M. Shankar. "Human Computer Interaction". In: *2010 3rd International Conference on Emerging Trends in Engineering and Technology*. 2010, pp. 1–4. DOI: 10.1109/ICETET.2010.85.

[SV06] T. Stahl and M. Völter. *Model-driven software development: technology, engineering, management*. Chichester: J. Wiley & Sons, 2006. URL: http://cds.cern.ch/record/1692118.

[Ste+08] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[VD+00] A. Van Deursen, P. Klint, and J. Visser. "Domain-Specific Languages: An Annotated Bibliography." In: *Sigplan Notices* 35.6 (2000), pp. 26–36.

[Vis08] E. Visser. "WebDSL: A Case Study in Domain-Specific Language Engineering". In: *Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers*. Ed. by R. Lämmel, J. Visser, and J. Saraiva. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 291–373. ISBN: 978-3-540-88643-3. DOI: 10.1007/978-3-540-88643-3_7. URL: https://doi.org/10.1007/978-3-540-88643-3_7.

[Viste] E. Visser. "Composing Domain-Specific Languages". https://www.slideshare.net/eelcovisser/composing-domainspecific-languages-2488110. Latest access: August 2017.

[Voxte] Voxxed. *MDD*. https://www.voxxed.com/2015/02/a-look-at-openxava-a-lightweight-model-driven-java-framework. Latest access: August 2017.

[Vö+13] M. Völter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats, E. Visser, and G. Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013. ISBN: 978-1-4812-1858-0. URL: http://www.dslbook.org.

[WC99] D. M. Weiss and Chi. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley Professional; Har/Cdr edition, 1999.

[Woh+12] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

Appendix A

Validation Rules

In this appendix, we present the *valdiation rules* developed for the *new* USE-ME version. This rules allow to guide, suggest and validate *SLE* actions throughout the USE-ME development process. The rules are divided by table, so: the **Utility** rules are specified in table A.1, the **Context Model** rules are specified in table A.2, the **Goal Model** rules are specified in table A.3, the **Evaluation Model** rules are specified in table A.4, the **Interaction Model** rules are specified in table A.5, the **Survey Model** rules are specified in table A.6, the **Report Model** rules are specified in table A.7. For each rules we provide a:

- **description**: on the rule. It can be *defined* if the object must/should be defined (*i.e.* Utility Specification must have a Name defined), or it can be *associated* if an object was already created and should be associated to another object (*i.e.* DSL should have a Documentation associated);
- **type**: which can be *strict*, meaning that the rule is mandatory (*i.e.* produces an error), or it can be *suggestion*, meaning that the rule is a suggestion (*i.e.* produces a warning);
- **message**: shown to the user, in order to help the user to fix the error/warning. The *validation rules* were design in such a way that they can educate the user on the development process.

Table A.1: Utility validation rules.

Description	**Type**	**Message**
UtilitySpecification.defined()	strict	USE-ME Utility Error: Create Child " Utility Specification" in Parent "Specification" if you would like to reuse some artefacts from an existing DSL under development.
UtilitySpecification name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Utility Specification" should be set to better distinguish between different utility specifications.
UtilitySpecification DSL.defined()	strict	USE-ME Utility Error: Create Child "DSL" in Parent "Utility Specification" to specify the DSL under development.
DSL DslName.defined()	strict	USE-ME Utility Error: "Dsl Name" of "DSL" must be set to better distinguish between different DSLs.
DSL ConcreteSyntax.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Concrete Syntax" in Parent "DSL" to associate a Concrete Syntax.
ConcreteSyntax name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Concrete Syntax" should be set to better distinguish between different concrete syntaxes.
DSL AbstractSyntax.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Abstract Syntax" in Parent "DSL" to associate an Abstract Syntax.

Description	Type	Message
AbstractSyntax name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Abstract Syntax" should be set to better distinguish between different abstract syntaxes.
DSL AbstractSyntax or ConcreteSyntax.defined()	strict	USE-ME Utility Error: Create Child "Concrete Syntax" and/or "Abstract Syntax" in Parent "DSL".
DSL ExistingGoalModel.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Existing GM" in Parent "DSL" to reuse the knowledge from a previous developed Goal Model.
ExistingGoalModel name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Existing GM" should be set to better distinguish between different existing goal models.
UtilitySpecification FunctionalGoal.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Functional Goal" in Parent "Utility Specification" to specify/reuse functional goals.
FunctionalGoal name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Functional Goal" should be set to better distinguish between different functional goals.
FunctionalGoal ExistingGM.associated()	suggestion	USE-ME Utility Suggestion: "Existing GM" of "Functional Goal" should be associated to refer to a specific goal model.
UtilitySpecification ProcessModel.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Process Model" in Parent "Utility Specification" to refer to business process models previously designed.
ProcessModel name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Process Model" should be set to better distinguish between different process models.
UtilitySpecification SurveyEngine.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Survey Engine" in Parent "Utility Specification" to connected to an existing survey platform (e.g. Google Forms, ...).
UtilitySpecification Documentation.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Documentation" in Parent "Utility Specification" to reffer to existing documentation (e.g. videos, presentations, ...).
Documentation name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Documentation" should be set to better distinguish between different documents.
DSL Documentation.associated()	suggestion	USE-ME Utility Suggestion: "Documentation" of "DSL" should be associated to refer to existing documents (e.g. videos, presentations, ...).
UtilitySpecification OutRef.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Outside Ref" in Parent "Utility Specification" to refer to Process Model, Documentation or Survey Engine outside references.

Description	Type	Message
OutRef name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "OutsideRef" should be set to better distinguish between different outside references.
OutRef Link.defined()	suggestion	USE-ME Utility Suggestion: "Link" of "Outside Ref" should be set to define the link to the outside reference.
OutRef Tool.defined()	suggestion	USE-ME Utility Suggestion: "Tool" of "OutsideRef" should be set to define the tool used.
OutRef OutRef.associated()	suggestion	USE-ME Utility Suggestion: "Outside Ref" of "Outside Ref" should be associated.
Documentation OutRef.associated()	suggestion	USE-ME Utility Suggestion: "Outside Ref" of "Documentation" should be associated to refer to an existing outside reference.
ProcessModel OutRef.associated()	suggestion	USE-ME Utility Suggestion: "Outside Ref" of "Process Model" should be associated to refer to an existing outside reference.
UtilitySpecification Requirement.defined()	suggestion	USE-ME Utility Suggestion: Create Child "Requirement" in Parent "Utility Specification" to define the DSL requirements.
Requirement name.defined()	suggestion	USE-ME Utility Suggestion: "Name" of "Requirement" should be set to better distinguish between different requirements.
Requirement Description.defined()	suggestion	USE-ME Utility Suggestion: "Description" of "Requirement" should be set.
FunctionalGoal Requirements.associated()	suggestion	USE-ME Utility Suggestion: "Requirement" of "Functional Goal" should be associated to define the functional goal requirements.

Table A.2: Context validation rules.

Description	Type	Message
ContextSpecification.defined()	strict	USE-ME Context Error: Create Child "Context Specification" in Parent "Specification" to describe the DSL intended context of use.
ContextSpecification name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Context Specification" should be set to better distinguish between different context specifications.
ContextModel.defined()	strict	USE-ME Context Error: Create Child "Context Model" in Parent "Context Specification".
ContextModel cmName.defined()	strict	USE-ME Context Error: "Cm Name" of "Context Model" must be set to differentiate from other context models.
ContextModel contextProvider.defined()	strict	USE-ME Context Error: "Context Provider" of "Context Model" must be set to specify the entity/stakeholder that provides information about the DSL.
ContextModel domain.defined()	strict	USE-ME Context Error: "Domain" of "Context Model" must be set to specify the DSL domain.

Description	Type	Message
ContextModel Language.associated()	suggestion	USE-ME Context Suggestion: "Language" of "Context Model" should be associated to represent the DSL under evaluation.
for each ContextModel UserHierarchy.defined()	strict	USE-ME Context Error: Create Child "User Hierarchy" in Parent "Context Model" to prioritise the DSL users.
UserHierarchy UhDescription.defined()	suggestion	USE-ME Context Suggestion: "Uh Description" of "User Hierarchy" should be set to describe the user hierarchy.
UserProfileSpecification.defined()	strict	USE-ME Context Error: Create Child "User Profile Specification" in Parent "Context Specification" to specify who will use the DSL.
UserProfileSpecification name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "User Profile Specification" should be set to better distinguish between different user profile specifications
for each UserProfileSpecification UserProfile.defined()	strict	USE-ME Context Error: Create Child "User Profile" in "User Profile Specification" to specify the DSL end users.
for each UserProfile name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "User Profile" should be set to better distinguish between different user profiles.
UserProfileDSLStakeholder.defined()	suggestion	USE-ME Context Suggestion: Create Child "User Profile" with Name "DSL Stakeholder" in "User Profile Specification" to specify the DSL root stakeholder.
UserProfile where name="DSL Stakeholder"	suggestion	USE-ME Context Suggestion: Rename "Name" of "User Profile" to "DSL Stakeholder".
UserProfileLanguageEngineer.defined()	suggestion	USE-ME Context Suggestion: Create Child "User Profile" with Name "Language Engineer" in "User Profile Specification" to specify the DSL language engineer.
UserProfile where name="Language Engineer"	suggestion	USE-ME Context Suggestion: Rename "Name" of "User Profile" to "Language Engineer".
UserProfileDomainExpert.defined()	suggestion	USE-ME Context Suggestion: Create Child "User Profile" with Name "Domain Expert" in "User Profile Specification" to specify the DSL domain expert.
UserProfile where name="Domain Expert"	suggestion	USE-ME Context Suggestion: Rename "Name" of "User Profile" to "Domain Expert".
UserProfileExpertEvaluator.defined()	suggestion	USE-ME Context Suggestion: Create Child "User Profile" with Name "Expert Evaluator" in "User Profile Specification" to specify the DSL expert evaluator.
UserProfile where name="Expert Evaluator"	suggestion	USE-ME Context Suggestion: Rename "Name" of "User Profile" to "Expert Evaluator".
UserProfileEndUser.defined()	suggestion	USE-ME Context Error: Create Child "User Profile" Name "End User" in "User Profile Specification" to specify the DSL end user.
UserProfile where name="End User"	suggestion	USE-ME Context Suggestion: Rename "Name" of "User Profile" to "End User".
UserProfile name="Language Engineer" set Parent(DSL Stakeholder).associated()	suggestion	USE-ME Context Suggestion: "Parent" of "User Profile Language Engineer" should be set to "DSL Stakeholder".

Description	Type	Message
UserProfile name="Domain Expert" set Parent(DSL Stakeholder).associated()	suggestion	USE-ME Context Suggestion: "Parent" of "User Profile Domain Expert" should be set to "DSL Stakeholder".
UserProfile name="Expert Evaluator" set Parent(DSL Stakeholder).associated()	suggestion	USE-ME Context Suggestion: "Parent" of "User Profile Expert Evaluator" should be set to "DSL Stakeholder".
UserProfile name="End User" set Parent(DSL Stakeholder).associated()	suggestion	USE-ME Context Suggestion: "Parent" of "User Profile End User" should be set to "DSL Stakeholder".
UserProfile name="End User" set SubProfile.associated()	strict	USE-ME Context Suggestion: "Sub Profile" of "User Profile End User" must be set into at least 2 end user sub profiles
UserProfile name!="DSL, LE, DE, EE and EU" set Parent.associated()	strict	USE-ME Context Error: "Parent" of "User Profile" must be set.
UserProfile DSLStakeholder priority=high	suggestion	USE-ME Context Suggestion: Change the "Priority" of "User Profile DSL Stakeholder" to "High" level.
UserProfile LanguageEngineer priority=low	suggestion	USE-ME Context Suggestion: Change the "Priority" of "User Profile Language Engineer" to "Low" level.
UserProfile DomainExpert priority=medium	suggestion	USE-ME Context Suggestion: Change the "Priority" of "User Profile Domain Expert" to "Medium" level.
UserProfile ExpertEvaluator priority=low	suggestion	USE-ME Context Suggestion: Change the "Priority" of "User Profile Expert Evaluator" to "Low" level.
UserProfile EndUser priority=high	suggestion	USE-ME Context Suggestion: Change the "Priority" of "User Profile End User" to "High" level.
UserProfile if SubProfile>=1 check if at least one of UserProfile priority=parent.priority	strict	USE-ME Context Error: Change the "Priority" of one "Sub Profile" to match the "Parent" priority.
UserHierarchy UserProfile.associated()	strict	USE-ME Context Error: "User Profile" of "User Hierarchy" must be associated to define the user hierarchy root profile, it is suggested to associate the "DSL Stakeholder" as the user hierarchy root profile.
for each UserProfileSpecification ProfileTemplate.defined()	strict	USE-ME Context Error: Create Child "Profile Template" in Parent "User Profile Specification" to describe user profile characteristics such as background information (e.g. demographic data, education, ...) and relevant experience with domain activities (e.g. expected knowledge sets, ontology, ...).
ProfileTemplate name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Profile Template" should be set to better distinguish between different profile templates.
ProfileTemplate Category.defined()	suggestion	USE-ME Context Suggestion: "Category" of "Profile Template" should be set to specify the profile template category (e.g. demographic data, education, expected knowledge set)
UserProfile where priority=high ProfileTemplate.associated()	strict	USE-ME Context Error: "Profile Template" of "User Profile" must be associated to refer to a specific profile template.
for each UserProfileSpecification LogicalExpression.defined()	strict	USE-ME Context Error: Create Child "Logical Expression" in Parent "User Profile Specification" to justify the creation of new end user sub profiles.

Description	Type	Message
LogicalExpression name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Logical Expression" should be set to better distinguish between different logical expressions.
LogicalExpression Expression.defined()	strict	USE-ME Context Error: "Expression" of "Logical Expression" must be set it can contain concrete (e.g. age >7) or abstract (e.g. age = int) specifications.
LogicalExpression ClassifierName.defined()	suggestion	USE-ME Context Suggestion: "Classifier Name" of "Logical Expression" should be set to better distinguish between different classifier expressions.
ProfileTemplate Classifiers/LogicalExpression.associated()	strict	USE-ME Context Error: "Classifiers" of "Profile Template" should be associated to define which classifiers apply to a specific profile template.
UserProfile (priority=high&SubProfile>=1) Classifier/LogicalExpression.associated()	strict	USE-ME Context Error: "Classifier" of "User Profile" should be associated to the "Logical Expression" which classify the sub-profiles in distinct sets.
EnviromentSpecification.defined()	strict	USE-ME Context Error: Create Child "Enviroment Specification" in Parent "Context Specification" to define where will the DSL be used.
EnviromentSpecification name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Enviroment Specification" should be set to better distinguish between different enviroment specifications.
for each EnviromentSpecification Environment.defined()	strict	USE-ME Context Error: Create Child "Technical Environment" and/or "Physical Environment" and/or "Social Environment" in Parent "Enviroment Specification" to specify in which environments the DSL is going to be used.
TechnicalEnvironment name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Technical Environment" should be set to better distinguish between different technical enviroments.
TechnicalEnvironment OutRef.associated()	suggestion	USE-ME Context Suggestion: "Outside Ref" of "Technical Environment" should be associated to define an outside reference for the environment.
PhysicalEnvironment name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Physical Environment" should be set to better distinguish between different physical enviroments.
PhysicalEnvironment OutRef.defined()	suggestion	USE-ME Context Suggestion: "Outside Ref" of "Physical Environment" should be associated to define an outside reference for the environment.
SocialEnvironment name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Social Environment" should be set to better distinguish between different social enviroments.
SocialEnvironment OutRef.defined()	suggestion	USE-ME Context Suggestion: "Outside Ref" of "Social Environment" should be associated to define an outside reference for the environment.

Description	Type	Message
ContextModel contextEnvironment.associated()	strict	USE-ME Context Error: "Context Environment" of "Context Model" must be associated to define in which environment the DSL is going to be used.
for each EnviromentSpecification CEVariable.defined()	strict	USE-ME Context Error: Create Child "CE Variable" in Parent "Enviroment Specification" to represent an environment variable (e.g. operating system, computer, country).
CEVariable name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "CE Variable" should be set to better distinguish between different CE variables.
CEVariable Type.defined()	suggestion	USE-ME Context Suggestion: "Type" of "CE Variable" should be set (e.g. OS = {Windows, Mac, Linux}).
CEVariable isMandatory.defined()	suggestion	USE-ME Context Suggestion: "Mandatory" of "CE Variable" should be set to true if... false otherwise.
for each CEVariable type>=1 ComposedCEVariable.defined()	strict	USE-ME Context Error: Create Child "CE Variable" in Parent "CE Variable" to represent each type specified in "CE Variable" type.
CEVariable name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "CE Variable" should be set to better distinguish between different CE variables.
CEVariable Type.defined()	suggestion	USE-ME Context Suggestion: "Type" of "CE Variable" should be set (e.g. Mac = {El Capitan, Yosemite}).
CEVariable isMandatory.defined()	suggestion	USE-ME Context Suggestion: "Mandatory" of "CE Variable" should be set to true if... false otherwise.
CEVariable ContextEnvironment.associated()	suggestion	USE-ME Context Suggestion: "Context Environment" of "CE Variable" should be associated (e.g. technical, physical or social environment).
for each TechnicalEnvironment CEElement.associated()	strict	USE-ME Context Error: "CE Element" of "Technical Environment" must be associated to define which CE variables apply to technical environment.
for each PhysicalEnvironment CEElement. associated()	strict	USE-ME Context Error: "CE Element" of "Physical Environment" must be associated to define which CE variables apply to physical environment.
for each SocialEnvironment CEElement. associated()	strict	USE-ME Context Error: "CE Element" of "Social Environment" must be associated to define which CE variables apply to social environment.
WorkflowSpecification.defined()	strict	USE-ME Context Error: Create Child "Workflow Specification" in Parent "Context Specification" to define how the DSL is expected to be used.
WorkflowSpecification name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Workflow Specification" should be set to better distinguish between different workflow specifications.

Description	Type	Message
for each WorkflowSpecification Workflow.defined()	strict	USE-ME Context Error: Create Child "Workflow" in Parent "Workflow Specification" to specify a group of tasks.
Workflow name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Workflow" should be set to better distinguish between different workflows.
Workflow Actor.associated()	suggestion	USE-ME Context Suggestion: "Actor" of "Workflow" should be associated to define which user profiles perform the workflow.
Workflow priority=high ContextElement.associated()	suggestion	USE-ME Context Suggestion: "Context Element" of "Workflow" should be associated to define environment elements that are relevant.
Workflow ContextModel.associated()	suggestion	USE-ME Context Suggestion: "Context Model" of "Workflow" should be associated to define the context to which the workflow applies.
Workflow ProcessModel.associated()	suggestion	USE-ME Context Suggestion: "Process Model" of "Workflow" should be associated to define which process models apply to the workflow.
Workflow Priority.defined()	suggestion	USE-ME Context Suggestion: "Priority" of "Workflow" should be set to high if the workflow is very important.
for each Workflow priority=high Scenario.defined()	strict	USE-ME Context Error: Create Child "Scenario" in Parent "Workflow" to represent concrete tasks (i.e. use cases).
Scenario name.defined()	suggestion	USE-ME Context Suggestion: "Name" of "Scenario" should be set to better distinguish between different scenarios.
Scenario Doc.associated()	suggestion	USE-ME Context Error: "Doc" of "Scenario" should be associated to define which documents apply to the scenario.

Table A.3: Goal validation rules.

Description	Type	Message
GoalSpecification.defined()	strict	USE-ME Goal Error: If you don't want to extend "Context Model" create Child "Goal Specification" in Parent "Specification" to specify the objectives of the user while using the DSL.
GoalSpecification name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Goal Specification" should be set to better distinguish between different goal specifications.
for each GoalSpecification GoalModel.defined()	strict	USE-ME Goal Error: Create Child "Goal Model" in Parent "Goal Specification" to capture the various objectives of the system that should be achieved.
GoalModel name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Goal Model" should be set to better distinguish between different goal models.
GoalModel Language.associated()	suggestion	USE-ME Goal Suggestion: "Language" of "Goal Model" should be set to better define the DSL to which the goal model applies.

Description	Type	Message
for each GoalModel Usability-GoalQualityInUse.defined()	strict	USE-ME Goal Error: Create Child "Usability Goal Quality in Use" in Parent "Goal Model" to define usability goal that is the highest level objective for the developed DSL.
for each GoalModel Usability-Goal.defined()	strict	USE-ME Goal Error: Create Child "Usability Goal" in Parent "Goal Model" to define usability goal quality in use subgoals.
only one UsabilityGoal="Quality in Use".defined()	strict	USE-ME Goal Error: Rename "Usability Goal Quality in Use" only one Usability Goal Quality in Use can be defined.
UsabilityGoal Question.defined()	strict	USE-ME Goal Error: "Question" of "Usability Goal" must be set.
UsabilityGoal name='Quality in Use' priority=high	strict	USE-ME Context Error: Change the "Priority" of "Usability Goal Quality in Use" to high since it is the highest goal.
UsabilityGoal name="Quality in Use" SubGoal.defined()	strict	USE-ME Goal Error: "Sub Goal" of "Usability Goal" must be set since "Usability Goal Quality in Use" is the highest level objective.
UsabilityGoal name!="Quality in Use" ParentGoal.defined()	strict	USE-ME Goal Error: "Parent Goal" of "Usability Goal" must be set to specify the usability goal parent.
UsabilityGoal if SubGoal>=1 check if at least one of UsabilityGoal SubGoals priority=parent.priority	strict	USE-ME Goal Error: Change the "Priority" of one "Usability Goal" sub goals to match the "Parent" priority level.
for each GoalSpecification Scope.defined()	strict	USE-ME Goal Error: Create Child "Scope" in Parent "Goal Specification" to associate the context of use (e.g. User Profiles, Environments and Workflows) to a certain usability goal.
Scope name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Scope" should be set to better distinguish between different scopes.
Scope ContextModel.associated()	strict	USE-ME Goal Error: "Context Model" of "Scope" must be associated to define to which context of use it applies to.
Scope UsabilityGoal.associated()	strict	USE-ME Goal Error: "Usability Goal" of "Scope" must be associated to define to which usability goal it applies to.
Scope UserProfileSelection.associated()	suggestion	USE-ME Goal Error: "User Profile Selection" of "Scope" must be associated to define to which user profile it applies to.
Scope Workflow.associated()	suggestion	USE-ME Goal Error: "Workflow" of "Scope" must be associated to define to which workflows it applies to.
Scope ContextEnvironment.associated()	suggestion	USE-ME Goal Error: "Context Environment" of "Scope" must be associated to define to which environments it applies to.
for each GoalSpecification Actor.defined()	strict	USE-ME Goal Error: Create Child "Actor" in Parent "Goal Specification" which is a specialization of DSL stakeholder.
Actor name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Actor" should be set to better distinguish between different actors.
ActorExpertEvaluator.defined()	suggestion	USE-ME Goal Error: Create Child "Actor" with Name "Expert Evaluator" in Parent "Goal Specification" to represent the language expert evaluator.

Description	Type	Message
Actor where name="Expert Evaluator"	suggestion	USE-ME Goal Suggestion: Rename "Name" of "Actor" to "Expert Evaluator".
Actor Stakeholder.associated()	strict	USE-ME Goal Error: "Stakeholder" of "Actor " must be associated to define which DSL stakeholder is responsible.
Actor="Expert Evaluator" Stakeholder.size=1.defined()	suggestion	USE-ME Goal Suggestion: "Stakeholder" of "Actor Expert Evaluator" should only have "User Profile Expert Evaluator" has stakeholder.
Actor="Expert Evaluator" and Stakeholder=UPExpertEvaluator.associated()	strict	USE-ME Goal Suggestion: "Stakeholder" of "Actor Expert Evaluator" should be associated to "User Profile Expert Evaluator".
Actor Organization.defined()	suggestion	USE-ME Goal Error: "Organization" of "Actor " should be set to define the actor organization.
UsabilityGoal name="Quality in Use" All Actors are ResponsibleActor.defined()	strict	USE-ME Goal Suggestion: "Actor" should be associated as "Responsible Actor" of "Usability Goal Quality in Use".
UsabilityGoal with Subgoals = 0 only have one ResponsibleActor.defined()	suggestion	USE-ME Goal Suggestion: "Responsible Actor" of "Usability Goal" should be only associated to one single actor since the Usability Goal does not have any sub goals.
UsabilityGoal ResponsibleActor.associated()	suggestion	USE-ME Goal Suggestion: "Responsible Actor" of "Usability Goal" should be associated.
UsabilityGoal ResponsibleActor Name=ExpertEvaluator.defined()	suggestion	USE-ME Goal Suggestion: "Actor Expert Evaluator" should be associated as single "Responsible Actor" of one "Usability Goal" without "Sub Goals".
UsabilityGoal SubGoals=0 and ResponsibleActor=1 ProvidedFunctionality.associated()	suggestion	USE-ME Goal Suggestion: "Provided Functionality" of "Usability Goal" should be set to define the functional goal associated to the usability goal.
for each GoalSpecification Method.defined()	strict	USE-ME Goal Error: Create Child "Method" in Parent "Goal Specification" to define the measurable requirements that contribute to the achievement of the goal.
Method name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Method" should be set to better distinguish between different methods.
Method UsabilityGoal.associated()	strict	USE-ME Goal Error: "Usability Goal" of "Method" must be associated to define to which usability goal it contributes to achieve.
Method MethodDescription.defined()	suggestion	USE-ME Goal Suggestion: "Method Description" of "Method" should be set.
Method TestCase/Scenario.associated()	suggestion	USE-ME Goal Suggestion: "Test Case" of "Method" should be associated as it can be used to evaluate the requirement.
Method FunctionalGoal.associated()	suggestion	USE-ME Goal Suggestion: "Functional Goal" of "Method" should be associated as it represents the funcionalities that are provided to support the execution of certain test cases.
for each GoalSpecification UsabilityRequirement.defined()	strict	USE-ME Goal Error: Create Child "Usability Requirement" in Parent "Goal Specification" to define the usability requirements that contribute to the achievement of the goal.

Description	Type	Message
UsabilityRequirement name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Usability Requirement" should be set to better distinguish between different usability requirements.
UsabilityRequirement OldName.defined()	suggestion	USE-ME Goal Suggestion: "Old name" of "Usability Requirement" should be set to better distinguish between different old usability requirements.
UsabilityRequirement Metric.defined()	suggestion	USE-ME Goal Suggestion: "Metric" of "Usability Requirement" should be set to define which metrics are going to be use to measure the requirement.
UsabilityRequirement Description.defined()	suggestion	USE-ME Goal Suggestion: "Description" of "Usability Requirement" should be set.
UsabilityRequirement OldDescription.defined()	suggestion	USE-ME Goal Suggestion: "Description old" of "Usability Requirement" should be set.
Method UsabilityRequirement.associated()	suggestion	USE-ME Goal Suggestion: "Usability Requirement" of "Method" should be associated to define which measurable requirements (i.e. usability requirements) that contribute to the achievement of the goal.
for each GoalSpecification SuccessCoverage.defined()	suggestion	USE-ME Goal Error: Create Child "Success Coverage" in Parent "Goal Specification" to reflect the evaluated context coverage.
SuccessCoverage name.defined()	suggestion	USE-ME Goal Suggestion: "Name" of "Success Coverage" should be set to better distinguish between different success coverages.
SuccessCoverage Scope.associated()	strict	USE-ME Goal Error: "Scope" of "Success Coverage" must be associated.
SuccessCoverage UsabilityGoal.associated()	strict	USE-ME Goal Error: "Usability Goal" of "Success Coverage" must be associated.
SuccessCoverage SuccessFactor.associated()	strict	USE-ME Goal Error: "Success Factor" of "Success Coverage" must be associated.

Table A.4: Evaluation validation rules.

Description	Type	Message
EvaluationSpecification.defined()	strict	USE-ME Evaluation Error: If you don't want to extend "Goal Model" create Child "Evaluation Specification" in Parent "Specification" to evaluate the DSL.
EvaluationSpecification name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Evaluation Specification" should be set to better distinguish between different evaluation specifications.
for each EvaluationSpecification EvaluationModel.defined()	strict	USE-ME Evaluation Error: Create Child "Evaluation Model" in Parent "Evaluation Specification" to express the purpose of evaluating a certain objective for the DSL in a specific context.
EvaluationModel name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Evaluation Model" should be set to better distinguish between different evaluation models.

Description	Type	Message
Language.defined()	strict	USE-ME Evaluation Error: Create Child "Language" in Parent "Evaluation Specification" to define the language under evaluation.
Comparative Language.defined()	suggestion	USE-ME Evaluation Error: Create Child "Language" in Parent "Evaluation Specification" to compare other language with the language under evaluation.
Language name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Language" should be set to better distinguish between different languages.
Language Version.defined()	suggestion	USE-ME Evaluation Suggestion: "Version" of "Language" should be set to define language version.
Language DSL.associated()	suggestion	USE-ME Evaluation Suggestion: "DSL" of "Language" should be associated to define to which DSL applies to language.
Language EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Language" must be associated to define the language evaluation model.
EvaluationGoal.defined()	strict	USE-ME Evaluation Error: Create Child "Evaluation Goal" in Parent "Evaluation Specification" to define the experimental hypothesis and research questions.
EvaluationGoal name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Evaluation Goal" should be set to better distinguish between different evaluation goals.
EvaluationGoal Responsible/Actor.associated()	strict	USE-ME Evaluation Error: "Responsible" of "Evaluation Goal" should be associated to "Actor Expert Evaluator".
EvaluationGoal UsabilityGoal.associated()	strict	USE-ME Evaluation Error: "Usability Goal" of "Evaluation Goal" must be associated to relate to the usability goals specified in goal model.
EvaluationGoal UsabilityGoal has only "Actor Expert Evaluator" as "Responsible Actor".associated()	strict	USE-ME Evaluation Error: "Usability Goal" of "Evaluation Goal" should correspond to "Usability Goal" associated to "Actor" in which this "Actor" is the only responsible.
EvaluationGoal EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Evaluation Goal" must be associated to relate to the evaluation model.
EvaluationGoal ProblemDescription.defined()	suggestion	USE-ME Evaluation Suggestion: "Problem Description" of "Evaluation Goal" should be set to describe the problem.
EvaluationGoal ResearchQuestion.defined()	suggestion	USE-ME Evaluation Suggestion: "Research Question" of "Evaluation Goal" should be set to specify the research question.
EvaluationGoal Hypothesis.defined()	suggestion	USE-ME Evaluation Suggestion: "Hypothesis" of "Evaluation Goal" should be set to define the experimental hypothesis.
EvaluationGoal Mesurment.associated()	suggestion	USE-ME Evaluation Suggestion: "Mesurment" of "Evaluation Goal" should be associated to define which methods can be introduced as measurements.

Description	Type	Message
EvaluationGoal Language.associated()	suggestion	USE-ME Evaluation Suggestion: "Language" of "Evaluation Goal" should be associated to define the DSL under evaluation.
EvaluationGoal Comperative.defined()	suggestion	USE-ME Evaluation Suggestion: "Comperative" of "Evaluation Goal" should be set to true if a comparative evaluation (i.e. two languages) was defined.
EvaluationGoal UsabilityRequirement.associated()	suggestion	USE-ME Evaluation Suggestion: "Usability Requirement" of "Evaluation Goal" should be associated to specify which usability requirements are being evaluated.
Participant.defined()	strict	USE-ME Evaluation Error: Create Child "Participant" in Parent "Evaluation Specification" to define the study participants.
Participant name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Participant" should be set to better distinguish between different participants.
Participant Contact.defined()	strict	USE-ME Evaluation Error: "Contact" of "Participant" must be set to store information (e.g. email, phone, ...) about the participant.
Participant UserProfile.associated()	strict	USE-ME Evaluation Error: "User Profile" of "Participant" must be associated to match a specific "User Profile" end user.
Participant EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Participant" must be associated to match a specific evaluation model.
EvaluationContext.defined()	strict	USE-ME Evaluation Error: Create Child "Evaluation Context" in Parent "Evaluation Specification" to specify the user profiles, workflows, context environments taken into consideration during the experiment execution.
EvaluationContext name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Evaluation Context" should be set to better distinguish between different evaluation contexts.
EvaluationContext EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Evaluation Context" must be associated to define the evaluation model related with the evaluation.
EvaluationContext ContextModel.associated()	strict	USE-ME Evaluation Error: "Context Model" of "Evaluation Context" must be associated to specify the context of use related with the evaluation.
EvaluationContext ContextEnvironment.associated()	suggestion	USE-ME Evaluation Suggestion: "Context Environment" of "Evaluation Context" should be associated to define the context environment related with the evaluation.
EvaluationContext EnviromentInstance.defined()	suggestion	USE-ME Evaluation Suggestion: "Environment Instance" of "Evaluation Context" should be set to instantiate the environment variables (e.g. [OS]: Windows 7 from CE-Variables) related with the evaluation.
EvaluationContext UserProfileSelection.associated()	suggestion	USE-ME Evaluation Suggestion: "User Profile Selection" of "Evaluation Context" should be associated to specify the user profiles related with the evaluation.

Description	Type	Message
EvaluationContext UserProfileSelection.associated() matches the Participant UserProfile.associated()	suggestion	USE-ME Evaluation Suggestion: "User Profile Selection" of "Evaluation Context" should contain all "User Profile" defined in "Participant".
EvaluationContext UsabilityGoal.associated()	suggestion	USE-ME Evaluation Suggestion: "Usability Goal" of "Evaluation Context" should be associated to define the usability goals related with the evaluation.
EvaluationContext UsabilityGoal.associated() matches the EvalutionGoal Usability Goal.associated()	suggestion	USE-ME Evaluation Suggestion: "Usability Goal" of "Evaluation Context" should contain all "Usability Goal" defined in "Evaluation Goal".
EvaluationContext Workflow.associated()	suggestion	USE-ME Evaluation Suggestion: "Workflow" of "Evaluation Context" should be associated to define the workflows related with the evaluation.
EvaluationContext Scenario.associated()	suggestion	USE-ME Evaluation Suggestion: "Scenario" of "Evaluation Context" should be associated to define the scenarios related with the evaluation.
EvaluationContext Workflow from Scenario.associated()	suggestion	USE-ME Evaluation Suggestion: "Workflow" of "Evaluation Context" should contain "Scenario" associated "Workflow".
Documentation.defined()	strict	USE-ME Evaluation Error: Create Child "Documentation" in Parent "Evaluation Specification" to define teaching materials for the DSL (e.g. videos, guided examples, videos).
Documentation name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Documentation" should be set to better distinguish between different documents.
Documentation EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Documentation" must be associated to define for which evaluation the documents apply to.
Documentation OutRef.associated()	suggestion	USE-ME Evaluation Suggestion: "Outside Ref" of "Documentation" should be associated to define outside references.
Documentation Scenario.associated()	suggestion	USE-ME Evaluation Suggestion: "Scenario" of "Documentation" should be associated to define which scenarios are covered by the documentation.
all Scenario from EvaluationContext Scenario.associated()	suggestion	USE-ME Evaluation Suggestion: "Scenario" of "Documentation" should contain all "Scenario" associated in "Evalaution Context".
Process.defined()	strict	USE-ME Evaluation Error: Create Child "Process" in Parent "Evaluation Specification" to define the concrete design for the evaluation by modelling the activities that should be performed.
Process name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Process" should be set to better distinguish between different processes.
Process EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Process" must be associated to specify the learning activities that are modelled in the evaluation process.

Description	Type	Message
Process ProcessModel.associated()	suggestion	USE-ME Evaluation Suggestion: "Process Model" of "Process" should be associated to related process model.
TestModel.defined()	strict	USE-ME Evaluation Error: Create Child "Test Model" in Parent "Evaluation Specification" .
TestModel name.defined()	suggestion	USE-ME Evaluation Suggestion: "Name" of "Test Model" must be set.
TestModel EvaluationModel.associated()	strict	USE-ME Evaluation Error: "Evaluation Model" of "Test Model" should be associated.

Table A.5: Interaction validation rules.

Description	Type	Message
IS and/or SS.defined()	strict	USE-ME Interaction/Survey Error: Create Child "Interaction Specification" and/or "Survey Specification" in Parent "Specification".
InteractionSpecification.defined()	suggestion	USE-ME Interaction Error: If you need to specify "Test Model" create Child "Interaction Specification" in Parent "Specification" to measure usability over concrete tasks that involve interaction with the DSL.
InteractionSpecification name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Interaction Specification" should be set to better distinguish between different interaction specifications.
for each InteractionSpecification InteractionModel.defined()	strict	USE-ME Interaction Error: Create Child "Interaction Model" in Parent "Interaction Specification" to support the capture of predefined events and to provide statistics about the occurrences.
InteractionModel name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Interaction Model" should be set to better distinguish between different interaction models.
InteractionModel EvaluationModel.associated()	strict	USE-ME Interaction Error: "Evaluation Model" of "Interaction Model" must be associated to specify the evaluation model to which the interaction model applies to.
InteractionModel Participant.associated()	suggestion	USE-ME Interaction Suggestion: "Participant" of "Interaction Model" should be associated to define which user profiles will perform in the interaction.
Task.defined()	strict	USE-ME Interaction Error: Create Child "Task" in Parent "Interaction Specification" to represent a concrete task for which the interaction will be analysed.
Task name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Task" should be set to better distinguish between different tasks.
Task Scenario.associated()	strict	USE-ME Interaction Error: "Scenario" of "Task" must be associated to specify which scenarios the task covers.

Description	Type	Message
Task InteractionModel.associated()	suggestion	USE-ME Interaction Suggestion: "Interaction Model" of "Task'" should be associated to specify the interaction model the task applies to.
Task Documentation.associated()	suggestion	USE-ME Interaction Suggestion: "Documentation" of "Task" should be associated to specify the interaction model documentation.
InteractionSyntax.defined()	strict	USE-ME Interaction Error: Create Child "Interaction Syntax" in Parent "Interaction Specification" to reflect the interaction elements from the version of the language.
for each Language from Evaluation Specification InteractionSyntax.defined()	suggestion	USE-ME Interaction Suggestion: Create Child "Interaction Syntax" in Parent "Interaction Specification" for each "Language" specified in "Evaluation Specification".
InteractionSyntax name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Interaction Syntax" should be set to better distinguish between different interaction syntaxes.
InteractionSyntax InteractionModel.associated()	strict	USE-ME Interaction Error: "Interaction Model" of "Interaction Syntax" must be associated to specify to which interaction model the interaction syntax applies to.
InteractionSyntax AS and/or ConcreteSyntax.associated()	strict	USE-ME Interaction Error: "Concrete Syntax" and/or "Abstract Syntax" of "Interaction Syntax" should be associated.
InteractionSyntax ConcreteSyntax.associated()	suggestion	USE-ME Interaction Suggestion: "Concrete Syntax" of "Interaction Syntax" should be associated to define the concrete syntax.
InteractionSyntax AbstractSyntax.associated()	suggestion	USE-ME Interaction Suggestion: "Abstract Syntax" of "Interaction Syntax" should be associated to define the abstract syntax.
InteractionSyntax Documentation.associated()	suggestion	USE-ME Interaction Suggestion: "Documentation" of "Interaction Syntax" should be associated.
InteractionSyntax OutsideRef.associated()	suggestion	USE-ME Interaction Suggestion: "Outside Ref" of "Interaction Syntax" should be associated to an outside reference.
Event.defined()	strict	USE-ME Interaction Error: Create Child "Event" in Parent "Interaction Specification" to represent the type of data that will be captured from different interaction devices.
Event name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Event" should be set to better distinguish between different events.
Event InteractionModel.associated()	suggestion	USE-ME Interaction Error: "Interaction Model" of "Event" should be associated to specify to which interaction model the event applies to.
Event AnalysisType.defined()	suggestion	USE-ME Interaction Suggestion: "Analysis Type" of "Event" should be set to describe how the event is going to be analysed (e.g observation, time observation, sucess/fail).
Event Capture.defined()	suggestion	USE-ME Interaction Suggestion: "Capture" of "Event" should be set to describe how the event is going to be captured.

Description	Type	Message
Event RecordEvent.defined()	suggestion	USE-ME Interaction Suggestion: "Record Event" of "Event" should be set to define how the event is going to be recorded (e.g. screen record, live observation).
Event UsabilityRequirement.associated()	suggestion	USE-ME Interaction Suggestion: "Usability Requirement" of "Event" should be associated to specify to which usability requirements the event applies to.
if Event Capture size>=1 then CaptureAction.defined()	suggestion	USE-ME Interaction Suggestion: Create Child "Capture Action" in Parent "Event" to define capture actions.
CaptureAction name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Capture Action" should be set to better distinguish between different capture actions.
InteractionResult.defined()	strict	USE-ME Interaction Error: If the Evaluation was executed create Child "Interaction Result" in Parent "Interaction Specification" to include statistical analysis and results of the executed interaction model.
InteractionResult name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Interaction Result" should be set to better distinguish between different interaction results.
InteractionResult Event.associated()	strict	USE-ME Interaction Error: "Event" of "Interaction Result" must be associated to describe the event performed.
InteractionResult InteractionModel.associated()	strict	USE-ME Interaction Error: "Interaction Model" of "Interaction Result" must be associated to specify to which interaction model the interaction result applies to.
InteractionResult OutRef.associated()	suggestion	USE-ME Interaction Suggestion: "Outsife Reference" of "Interaction Result" should be set to specify outside references (e.g. google forms).
ResultValue.defined()	suggestion	USE-ME Interaction Suggestion: Create Child "Result Value" in Parent "Interaction Result" to define the results obtained.
for each Capture Action ResultValue.defined()	suggestion	USE-ME Interaction Suggestion: Create Child "Result Value" in Parent "Interaction Result" for each "Capture Action" defined.
ResultValue name.defined()	suggestion	USE-ME Interaction Suggestion: "Name" of "Result Value" should be set to better distinguish between different result values.
ResultValue ResultValue.defined()	strict	USE-ME Interaction Error: "Result Value" of "Result Value" must be set.
ResultValue Language.associated()	strict	USE-ME Interaction Error: "Language" of "Result Value" must be associated to a language under evaluation.
ResultValue AssociatedRequirement.associated()	suggestion	USE-ME Interaction Suggestion: "Associated Requirement" of "Result Value" should be associated to a specific requirement.
ResultValue RelatedAction.associated()	suggestion	USE-ME Interaction Suggestion: "Related Action" of "Result Value" should be associated to a capture action involved.

Table A.6: Survey validation rules.

Description	Type	Message
SurveySpecification.defined()	suggestion	USE-ME Survey Error: If you need to specify "Test Model" create Child "Survey Specification" in Parent "Specification" to support formative methods for measuring usability.
SurveySpecification name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Survey Specification" should be set to better distinguish between different survey specifications.
for each SurveySpecification SurveyModel.defined()	strict	USE-ME Survey Error: Create Child "Survey Model" in Parent "Survey Specification" to collect information to measure usability.
SurveyModel name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Survey Model" should be set to better distinguish between different survey models.
SurveyModel EvaluationModel.associated()	strict	USE-ME Survey Error: "Evaluation Model" of "Survey Model" must be associated to specify to which evaluation model the survey model applies to.
SurveyEngine SurveyModel.associated()	strict	USE-ME Survey Error: "Survey Model" of "Survey Engine" must be set.
SurveyModel Participant.associated()	suggestion	USE-ME Survey Suggestion: "Participant" of "Survey Model" should be associated to define the participants involved in the survey.
Questionnaire.defined()	strict	USE-ME Survey Error: Create Child "Questionnaire" in Parent "Survey Specification" to define a set of questions.
Questionnaire name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Questionnaire" should be set to better distinguish between different questionnaires.
Questionnaire SurveyModel.associated()	strict	USE-ME Survey Error: "Survey Model" of "Questionnaire" must be associated to specify to which survey model the questionnaire applies to.
Questionnaire UsabilityRequirement.associated()	suggestion	USE-ME Survey Suggestion: "Usability Requirement" of "Questionnaire" should be associated to specify which usability requirements are addressed in the questionnaire.
FeedbackQs and/or BackgroundQs.defined()	strict	USE-ME Survey Error: Create Child "Feedback Qs" to collect opinions about the DSL, and/or create Child "Background Qs" to collect information about the participant in Parent "Survey Specification".
BackgroundQs name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Background Qs" should be set to better distinguish between different background questions.
BackgroundQs UserProfile.associated()	strict	USE-ME Survey Error: "User Profile" of "Background Qs" should be set to define the user profile involved in the questionnaire.
BackgroundQs LogicalExpression.associated()	suggestion	USE-ME Survey Suggestion: "Logical Expression" of "Background Qs" should be associated to define to which logical expression is the question related.
BackgroundQs Question.defined()	suggestion	USE-ME Survey Suggestion: "Question" of "Background Qs" should be set (e.g. demographic data, education).

Description	Type	Message
BackgroundQs Questionnaire.associated()	suggestion	USE-ME Survey Suggestion: "Questionnaire" of "Background Qs" should be associated to define which questionnaire is related to the background question.
BackgroundQs Scale.defined()	suggestion	USE-ME Survey Suggestion: "Scale" of "Background Qs" should be set (e.g int, M/F, scale).
BackgroundQs Type.defined()	suggestion	USE-ME Survey Suggestion: "Type" of "Background Qs" should be set (e.g. demographics, experience, tendency).
FeedbackQs name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Feedback Qs" should be set to better distinguish between different feedback questions.
FeedbackQs Question.defined()	suggestion	USE-ME Survey Suggestion: "Question" of "Feedback Qs" should be set (e.g. did you enjoy the activity?, did you find it difficult?).
FeedbackQs Questionnaire.associated()	suggestion	USE-ME Survey Suggestion: "Questionnaire" of "Feedback Qs" should be associated to define which questionnaire is related to the feedback question.
FeedbackQs Scale.defined()	suggestion	USE-ME Survey Suggestion: "Scale" of "Feedback Qs" should be set (e.g. smiles scale).
FeedbackQs Scenario.associated()	suggestion	USE-ME Survey Suggestion: "Scenario" of "Feedback Qs" should be associated to collect opinions and reactions for a specific scenario.
FeedbackQs Type.defined()	suggestion	USE-ME Survey Suggestion: "Type" of "Feedback Qs" shoauld be set (e.g. confidence, likeable).
SurveyResult.defined()	strict	USE-ME Survey Error: If the Evaluation was executed create Child "Survey Result" in Parent "Survey Specification" to include the statistical analysis and results.
SurveyResult name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Survey Result" should be set to better distinguish between different survey results.
SurveyResult Questionnaire.associated()	strict	USE-ME Survey Error: "Questionnaire" of "Survey Result" must be associated to a specific questionnaire.
SurveyResult SurveyModel.associated()	strict	USE-ME Survey Error: "Survey Model" of "Survey Result" must be associated to a specific survey model.
SurveyResult OutRef.associated()	suggestion	USE-ME Survey Suggestion: "Outsife Reference" of "Survey Result" should be associated to a specific outside reference (e.g. google forms).
for each SurveyResult ResultValue.defined()	suggestion	USE-ME Survey Suggestion: Create Child "Result Value" in Parent "Survey Result" to define the results obtained.
for each Quesiton ResultValue.defined()	suggestion	USE-ME Survey Suggestion: Create Child "Result Value" in Parent "Survey Result" for each question "Background Qs" and "Feedback Qs" defined.
ResultValue name.defined()	suggestion	USE-ME Survey Suggestion: "Name" of "Result Value" should be set to better distinguish between different result values.

Description	Type	Message
ResultValue ResultValue.defined()	strict	USE-ME Survey Error: "Result Value" of "Result Value" must be set.
ResultValue Language.associated()	strict	USE-ME Survey Error: "Language" of "Result Value" must be associated to a language under evaluation.
ResultValue AssociatedRequirement.associated()	suggestion	USE-ME Survey Suggestion: "Associated Requirement" of "Result Value" should be associated to a specific requirement.
ResultValue RelatedAction.associated()	suggestion	USE-ME Survey Suggestion: "Related Action" of "Result Value" should be associated to a capture action involved.
ResultValue RelatedQuestion.associated()	suggestion	USE-ME Survey Suggestion: "Related Question" of "Result Value" should be associated to a question in the questionnaire.

Table A.7: Report validation rules.

Description	Type	Message
ReportSpecification.defined()	strict	USE-ME Report Error: If you don’t need to specify "Test Model" or you already specify it create Child "Report Specification" in Parent "Specification" to construct a final report on the DSL evaluation.
ReportSpecification name.defined()	suggestion	USE-ME Report Suggestion: "Name" of "Report Specification" should be set to better distinguish between different report specifications.
for each ReportSpecification ReportModel.defined()	strict	USE-ME Report Error: Create Child "Report Model" in Parent "Report Specification" to encapsulate the results of the experiment and to take into consideration suggestions.
ReportModel name.defined()	suggestion	USE-ME Report Suggestion: "Name" of "Report Model" should be set to better distinguish between different report models.
EvaluationResult.defined()	strict	USE-ME Report Error: Create Child "Evaluation Result" in Parent "Report Specification" to represent the results obtained.
EvaluationResult name.defined()	suggestion	USE-ME Report Suggestion: "Name" of "Evaluation Result" should be set to better distinguish between different evaluation results.
EvaluationResult EvaluationContext.associated()	strict	USE-ME Report Error: "Evaluation Context" of "Evaluation Result" must be associated to specify the evaluation context.
EvaluationResult ReportModel.associated()	strict	USE-ME Report Error: "Report Model" of "Evaluation Result" must be associated to specify the report model.
EvaluationResult InteractionResult.associated()	suggestion	USE-ME Report Suggestion: "Interaction Result" of "Evaluation Result" should be associated to specify the interaction results.
EvaluationResult OutRef.associated()	suggestion	USE-ME Report Suggestion: "Outsife Reference" of "Evaluation Result" should be associated to refer to an outside reference.

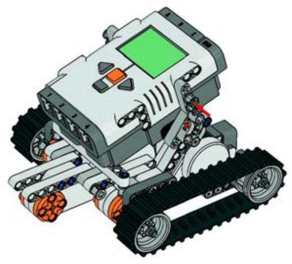
Description	Type	Message
EvaluationResult SurveyResult.associated()	suggestion	USE-ME Report Suggestion: "Survey Result" of "Evaluation Result" should be associated to define the survey result.
for each EvaluationResult ResultValue.defined()	suggestion	USE-ME Report Suggestion: Create Child "Result Value" in Parent "Evaluation Result" to define the results obtained.
for each Interact+SurveyResult ResultValue.defined()	suggestion	USE-ME Report Suggestion: Create Child "Result Value" in Parent "Evaluation Result" for each "Interaction Result" and "Survey Result".
ResultValue name.defined()	suggestion	USE-ME Report Suggestion: "Name" of "Result Value" should be set to better distinguish between different result values.
ResultValue Language.associated()	strict	USE-ME Report Error: "Language" of "Result Value" must be associated to a language under evaluation.
ResultValue ResultValue.defined()	strict	USE-ME Report Error: "Result Value" of "Result Value" must be set.
ResultValue AssociatedRequirement.associated()	suggestion	USE-ME Report Suggestion: "Associated Requirement" of "Result Value" should be associated to a specific requirement.
ResultValue RelatedAction.associated()	suggestion	USE-ME Report Suggestion: "Related Action" of "Result Value" should be associated to a capture action involved.
ResultValue RelatedQuestion.associated()	suggestion	USE-ME Report Suggestion: "Related Question" of "Result Value" should be associated to a question in the questionnaire.
RecommendedGM.defined()	strict	USE-ME Report Error: Create Child "Recommended GM" in Parent "Report Specification" to include updates (changes or new goals) to the previous goal model.
RecommendedGM name.defined()	suggestion	USE-ME Report Suggestion: "Name" of "Recommended GM" should be set to better distinguish between different recommended goal models.
RecommendedGM RefersTo.associated()	strict	USE-ME Report Error: "Refers To" of "Recommended GM" must be associated to the goal model evaluated.
RecommendedGM ReportModel.associated()	strict	USE-ME Report Error: "Report Model" of "Recommended GM" must be associated to the report model that was evaluated.
RecommendedGM FunctionalGoal.associated()	suggestion	USE-ME Report Suggestion: "Functional Goal" of "Recommended GM" should be associated to the functional goals that should improve in the next iteration.
RecommendedGM SuggestedRequirements.associated()	suggestion	USE-ME Report Suggestion: "Suggested Requirements" of "Recommended GM" should be associated to suggest requirements that should improve.
RecommendedGM UsabilityGoal.associated()	suggestion	USE-ME Report Suggestion: "Usability Goal" of "Recommended GM" should be associated to the usability goal that should improve in the next iteration.

Appendix B

USE-ME Original Version: Lego Exercise

In this appendix, we present the Lego exercise for the USE-ME original version.

LEGO Mindstorms



The main goal of this exercise is to perform an usability evaluation on a DSL under development.

LEGO Mindstorms are programmable robots designed for children. The main purpose of these robots is to teach kids to code, with some basics notions on coding while they play, and for that purpose a DSL Lego was developed.

1. Specification

First, you should create **Specification** with name *US*, within **Specification US** you should create a **DSL** with name **Lego**. Within **DSL Lego** you should create a **Concrete Syntax** with name **csLego,** an **Abstract Syntax** with name **asLego,** and an **ExistingGM** with name **gmLego**.
In **Specification US** you should also create a **Functional Goal** with name **fgLego**, a **Process Model** with name **pmLego**, a **Survey Engine**, a **Documentation** with name **docLego**, an **Outside Ref** with name **refLegoModeling** (Link: cameo.com; Tool: Cameo System Modeler), and a **Requirement** with name **Zooming** (Description: improve zooming option).

2. Context Specification

First, you should create **Context Specification** with name **cs.** Within **Context Specification cs** you should create a **Context Model** with Cm name **cmLego** (Context Provider: FCT; Domain: Program a robot). Within **Context Model cmLego** you should create an **User Hierarchy** (Uh Description: uhLego).
Next, you should create an **User Profile Specification** with name **upsLego**. Within you should create **User Profile** with name **DSL Stakeholder** (Priority: high, Sub Profile: End User),
User Profile with name **End User** (Priority: high, Sub Profile: Children and Adults),
User Profile with name **Children** (Priority: high, Sub Profile: Grade-Schoolers and Teens),
User Profile with name **Grade-Schoolers** (Priority: high),
User Profile with name **Teens** (Priority: medium), and
User Profile with name **Adults** (Priority: medium).

Next, you should create: **Profile Template** with name **DSL Stakeholder, Profile Template** with name **End User, Profile Template** with name **Children, Profile Template** with name

Grade-Schoolers, Profile Template with name **Teens**, and **Profile Template** with name **Adults** with **Category:** background demographics knowledge.

After that, you should create:
Logical Expression with name **Age** (Classifier: Age; Expression: >5; Profile Template: DSL Stakeholder, End User),
Logical Expression with name **Age Children** (Classifier: Age Children; Expression: 5-18; Profile Template: Children),
Logical Expression with name **Age Grade-Schoolers** (Classifier: Age Grade-Schoolers; Expression: 5-12; Profile Template: Grade Schoolers),
Logical Expression with name **Age Teens** (Classifier: Age Teens; Expression: 13-18; Profile Template: Teens),
Logical Expression with name **Age Adults** (Classifier: Age Adults; Expression: >18; Profile Template: Adults),
Logical Expression with name **School Grade** (Classifier: School Grade; Expression: 1-12; Profile Template: Children, Grade Schoolers, Teens),
Logical Expression with name **Computers** (Classifier: Computers; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates),
Logical Expression with name **Programming** (Classifier: Programming; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates).

Next, you should create an **Environment Specification** with name **esLego**. Within **Environment Specification esLego** you should create: **Technical Environment** with name **teLego**, **Physical Environment** with name **peLego**, and **Social Environment** with name **seLego**.
After that, you should create:
CE Variable with name **Robot** (Context Environment: Physical Environment pe; Mandatory: false; Type: Mindstorms),
CE Variable with name **Computer** (Context Environment: Physical Environment pe; Mandatory: true; Type: Desktop),
CE Variable with name **Application** (Context Environment: Technical Environment te; Mandatory: true; Type: Computer App), and
CE Variable with name **Workplace** (Context Environment: Social Environment se; Mandatory: true; Type: Classroom).
You should also create inside each CE Variable the correspondent CE Variable type (e.g. **CE Variable** with name **Mindstorms** within **CE Variable Robot**).

Lastly, you should create **Workflow Specification** with name **wsLego**, and then:
Workflow with name **W1: Program the robot** (Actor: End User; Context Element: Robot; Context Model: cmLego, Priority: High; Process Model: pmLego), and
Workflow with name **W2: Modify the language** (Actor: Language Engineer; Context Element: Application; Context Model: cmLego, Priority: Low; Process Model: pmLego).
You should create two scenarios for **Workflow W1: Program the robot:**
Scenario with name **Move front and back,** and
Scenario with name **Move forward until it hits an obstacle and then stop.**

3. Goal Specification

First, you should create **Goal Specification** with name **gs**. Then within **Goal Specification gs** you should create a **Goal Model** with name **gmLego**. Within **Goal Model gmLego** you should create:
Usability Goal with name **Quality in Use** (Priority: High; Question: Is the Quality in Use achieved?; Sub Goal: UG1 and UG2) this means that the DSL is usable,
Usability Goal with name **UG1: Capability to program the robot** (Priority: High; Question: Are the End Users capable of program the robot?; Sub Goal: UG1.1),
Usability Goal with name **UG1.1: Usability of programming the robot** (Priority: High; Question: Is it usable to program the robot?), and
Usability Goal with name **UG2: Evolve the language** (Priority: Medium; Question: Are Language Engineers capable of evolve the language?).

Next, you should create:
Scope with name **QualityInUse** (Context Environment: all; Context Model: cmLego; Usability Goal: Quality in Use; User Profile Selection: DSL Stakeholder; Workflow: all),
Scope with name **CapabilityProgramRobot** (Context Environment: all; Context Model: cmLego; Usability Goal: UG1 and UG1.1; User Profile Selection: End User; Workflow: W1), and
Scope with name **EvolveLanguage** (Context Environment: all; Context Model: cmLego; Usability Goal: UG2; User Profile Selection: Language Engineer; Workflow: W2).

After that, you should create:
Actor with name **Lego Development** (Organization: Lego Dev; Responsible For: Quality in Use, UG1 and UG2; Stakeholder: Language Engineer), and
Actor with name **Expert Evaluator** (Organization: Language Evaluator; Responsible For: all; Stakeholder: Expert Evaluator).

You should also create **Method** with name **Programming Robot is usable** (Method Description: Programming the robot is usable from end user perspective; Test Case: all; Usability Goal: Quality in Use and UG1.1; Usability Requirement: all).

Lastly, you should create:
Usability Requirement with name **Effectiveness** (Description/Description old: programming the robot is effective; Metric: Correctness of programmed code; Old Name: Effectiveness),
Usability Requirement with name **Learnability** (Description/Description old: programming the robot is easy to learn; Metric: Reused knowledge; Old Name: Learnability) ,
Usability Requirement with name **Satisfaction** (Description/Description old: programming the robot is satisfactory; Metric: Satisfaction questions; Old Name: Satisfaction) , and
Usability Requirement with name **Efficiency** (Description/Description old: programming the robot is efficient; Metric: Efficient programming; Old Name: Efficiency).

4. Evaluation Specification

First, you should create **Evaluation Specification** with name **es**. Then within **Evaluation Specification es** you should create **Evaluation Model** with name **emLego**. After that you should create only one **Language** with name **Lego** (DSL: Lego; Version: Mindstorms), since you are not doing a comparative evaluation.
Next, you should create:
Evaluation Goal with name **egLegoEffectiveness** (Comparative: false; Hypothesis = {effectiveness has no impact in robot programming, effectiveness has impact in robot programming}; Problem Description: analyse the impact of effectiveness on robot programming; Research Question: is it effective to program the robot?; Usability Goal: UG1.1), and
Evaluation Goal with name **egLegoSatisfaction** (Comparative: false; Hypothesis: {satisfaction has no impact in robot programming, satisfaction has impact in robot programming}; Problem Description: analyse the impact of satisfaction on robot programming; Research Question: is it satisfactory to program the robot?; Usability Goal: UG1.1).

The participants chosen for the Evaluation are children, so you should create:
Participant with name **Children** (Contact: Teacher contact; User Profile: Children).

After that you should create **Evaluation Context** with name **ecLego** (Context Environment: all; Context Model: cmLego; Environment Instance: Robot={Lego Mindstorms}, Computer={Desktop}, Application={Computer app}, Workplace={Classroom}; Scenario: all; Usability Goal: UG1.1; User Profile Selection: Children; Workflow: W1).

Next, you should provide some **Documentation** with name **doc** (Evaluation Model: emLego; Scenario: all) for the Scenarios.

Lastly, you should create the **Process** with name **EvaluationProcess** (Evaluation Model: emLego).

5. Interaction Specification

First, you should create **Interaction Specification** with name **is.** Then within **Interaction Specification is** you should create **Interaction Model** with name **imLego** (Evaluation Model: emLego; Participant: Children). Next, you should create **Task** with name **taskLego** (Documentation: all; Scenario: all).
After that, you should create **Interaction Syntax** with name **isLego** (Documentation: all; Interaction Model: imLego; Outside Ref: refLegoModeling).

Next, you should create **Event** with name **EffectivenessVideo** (Analysis Type: Observation; Capture: {Move front, Move back, Bump}; Interaction Model: imLego; Manual: true; Record Event: Screen Record; Usability Requirement: Effectiveness). And within **Event EffectivenessVideo,** create:
Capture Action with name **Move front**, **Capture Action** with name **Move back**, and **Capture Action** with name **Bump**.

After evaluating the DSL you created **Interaction Result** with name **irLego** (Event: Effectiveness Video; Interaction Model: imLego; Outside Ref: refLegoModeling) and extracted the following result values**:**
Result Value with name **MoveFront** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,75),
Result Value with name **MoveBack** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,75), and
Result Value with name **Bump** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,78).

6. Survey Specification

First, you should create **Survey Specification** with name **ss.** Then within **Survey Specification ss** you should create **Survey Model** with name **smLego** (Participant: Children; Survey Engine: Survey Engine).

Next you should create: **Questionnaire** with name **Background Questions** (Survey Model: smLego) and **Questionnaire** with name **Feedback Questions** (Survey Model: smLego; Usability Requirement: Satisfaction).

After that you should define:
Background Qs with name **Q1** (Logical Expression: Age Children; Question: Age; Scale: Integer; Type: Demographics; User Profile: Children),
Background Qs with name **Q2** (Logical Expression: School Grade; Question: School grade; Scale: Integer; Type: Demographics; User Profile: Children),
Background Qs with name **Q3** (Logical Expression: Computers; Question: How often do you play computer games?; Scale: {Every day, Sometimes, Rarely}; Type: Experience; User Profile: Children),
Background Qs with name **Q4** (Logical Expression: Programming; Question: Have you ever programmed?; Scale: {Yes, No}; Type: Experience; User Profile: Children),
Feedback Qs with name **F1** (Question: Did you enjoy the activity?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Likeability),
Feedback Qs with name **F2** (Question: Did you find it hard to move the robot front and back?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence),
Feedback Qs with name **F3** (Question: And to move front until it hits an obstacle and then stop?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence), and
Feedback Qs with name **F4** (Question: Would you like to repeat this activity?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence).

After evaluating the DSL you should create **Survey Result** with name **srLego** (Outside Reference: refLegoModeling; Questionnaire: Background and Feedback Questions) and extracted the following result values**:**
Result Value with name **Q1** (Language: Lego; Related Question: Q1; Result Value: 11,5),
Result Value with name **Q2** (Language: Lego; Related Question: Q2; Result Value: 6),
Result Value with name **Q3** (Language: Lego; Related Question: Q3; Result Value: 0,71),
Result Value with name **Q4** (Language: Lego; Related Question: Q4; Result Value: 0,8),

Result Value with name **F1** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F1; Result Value: 0,94),
Result Value with name **F2** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F2; Result Value: 0,52),
Result Value with name **F3** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F3; Result Value: 0,53), and
Result Value with name **F4** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F4; Result Value: 0,95).

7. Report Specification

First, you should create **Report Specification** with name **rs.** Then within **Report Specification rs** you should create **Report Model** with name **rmLego**. Next you should define **Evaluation Result** with name **erLego** and within:
Result Value with name **EffectivenessLego** (Related Action: all; Result Value: 0,78), and
Result Value with name **SatisfactionLego** (Related Question: F1, F2, F3, F4; Result Value: 0,85).

Next, you should create **Recommended GM** with name **rgmLego** (Functional Goal: fgLego; Refers To: gmLego; Report Model: rmLego; Suggested Requirements: Zooming; Usability Goal: UG1.1).

Lastly, you should create **Success Coverage** with name **scLego** (Scope: all; Success Factor: all; Usability Goal: UG1.1) in **Goal Specification** with name **gs**.

Thank you!

Appendix C

USE-ME Original Version: Smart House Exercise

In this appendix, we present the Smart House exercise for the USE-ME original version.

Smart House

The main goal of this exercise is to perform an usability evaluation on a DSL under development.

A Smart House is a collection of technical home automation concepts that are integrated together to meet the user goals, and for that purpose a DSL Smart House was developed.

1. Specification

First, you should create **Specification** with name **us.** Then within **Specification us** you should create a **DSL** with name **Smart House**. Within **DSL Smart House** you should create a **Concrete Syntax** with name **csSH,** an **Abstract Syntax** with name **asSH,** and an **ExistingGM** with name **gmSH**.
In **Specification us** you should also create a **Functional Goal** with name **fgSH**, a **Process Model** with name **pmSH**, a **Survey Engine**, a **Documentation** with name **docSH**, an **Outside Ref** with name **refSHModeling** (Link: cameo.com; Tool: Cameo System Modeler) , and a **Requirement** with name **Zooming** (Description: improve zooming option).

2. Context Specification

First, you should create **Context Specification** with name **cs**. Then within **Context Specification cs** you should create **Context Model** with Cm name **cmSH** (Context provider: FCT; Domain: Program a smart house). Within **Context Model cmSH** you should create an **User Hierarchy** (Uh Description: uhSH).
Next, you should create an **User Profile Specification** with name **upsSH**. Within you should create **User Profile** with name **DSL Stakeholder** (Priority: high, Sub Profile: End User),
User Profile with name **End User** (Priority: high, Sub Profile: Adults and Teens),
User Profile with name **Adults** (Priority: high, Sub Profile: Young adults and Middle adults),
User Profile with name **Young Adults** (Priority: high),
User Profile with name **Middle Adults** (Priority: medium), and
User Profile with name **Teens** (Priority: medium).

Next, you should create: **Profile Template** with name **DSL Stakeholder, Profile Template** with name **End User, Profile Template** with name **Adults, Profile Template** with name **Young Adults, Profile Template** with name **Middle Adults**, and **Profile Template** with name **Teens** with **Category:** background demographics knowledge.

After that, you should create:
Logical Expression with name **Age** (Classifier: Age; Expression: >13; Profile Template: DSL Stakeholder, End User),
Logical Expression with name **Age Adults** (Classifier: Age Adults; Expression: 20-64; Profile Template: Adults),
Logical Expression with name **Age Young Adults** (Classifier: Age Young Adults; Expression: 20-40; Profile Template: Young Adults),
Logical Expression with name **Age Middle Adults** (Classifier: Age Middle Adults; Expression: 40-64; Profile Template: Middle Adults),
Logical Expression with name **Age Teens** (Classifier: Age Teens; Expression: 13-19; Profile Template: Teens),
Logical Expression with name **Computers** (Classifier: Computers; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates),
Logical Expression with name **House Automation** (Classifier: House Automation; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates).

Next, you should create an **Environment Specification** with name **esSH**. Within **Environment Specification esSH** you should create: **Technical Environment** with name **teSH**, **Physical Environment** with name **peSH**, and **Social Environment** with name **seSH**.
After that, you should create:
CE Variable with name **Smart House** (Context Environment: Physical Environment pe; Mandatory: false; Type: NOVA-LINCS),
CE Variable with name **Computer** (Context Environment: Physical Environment pe; Mandatory: true; Type: Desktop),
CE Variable with name **Application** (Context Environment: Technical Environment te; Mandatory: true; Type: Computer App), and
CE Variable with name **Workplace** (Context Environment: Social Environment se; Mandatory: true; Type: Classroom).
You should also create inside each CE Variable the correspondent CE Variable type (e.g. **CE Variable** with name **NOVA-LINCS** within **CE Variable Smart House**).

Lastly, you should create **Workflow Specification** with name **wsSH**, and then:
Workflow with name **W1: Program the Smart House** (Actor: End User; Context Element: Smart House; Context Model: cmSH, Priority: High; Process Model: pmSH), and
Workflow with name **W2: Modify the language** (Actor: Language Engineer; Context Element: Application; Context Model: cmSH, Priority: Low; Process Model: pmSH).
You should create two scenarios for **Workflow W1: Program the Smart House:**
Scenario with name **When front door opens says Hello,** and
Scenario with name **When alarm rings Smart House opens windows**.

3. Goal Specification

First, you should create **Goal Specification** with name **gs**. Then within **Goal Specification gs** you should create a **Goal Model** with name **gmSH**. Within **Goal Model gmSH** you should create:

Usability Goal with name **Quality in Use** (Priority: High; Question: Is the Quality in Use achieved?; Sub Goal: UG1 and UG2) this means that the DSL is usable,
Usability Goal with name **UG1: Capability to program the smart house** (Priority: High; Question: Are the End Users capable of program the smart house?; Sub Goal: UG1.1),
Usability Goal with name **UG1.1: Usability of programming the smart house** (Priority: High; Question: Is it usable to program the smart house?), and
Usability Goal with name **UG2: Evolve the language** (Priority: Medium; Question: Are Language Engineers capable of evolve the language?).

Next, you should create:
Scope with name **QualityInUse** (Context Environment: all; Context Model: cmSH; Usability Goal: Quality in Use; User Profile Selection: DSL Stakeholder; Workflow: all),
Scope with name **CapabilityProgramSmartHouse** (Context Environment: all; Context Model: cmSH; Usability Goal: UG1 and UG1.1; User Profile Selection: End User; Workflow: W1), and
Scope with name **EvolveLanguage** (Context Environment: all; Context Model: cmSH; Usability Goal: UG2; User Profile Selection: Language Engineer; Workflow: W2).

After that, you should create:
Actor with name **Smart House Development** (Organization: Smart House Dev; Responsible For: Quality in Use, UG1 and UG2; Stakeholder: Language Engineer), and
Actor with name **Expert Evaluator** (Organization: Language Evaluator; Responsible For: all; Stakeholder: Expert Evaluator).
You should also create **Method** with name **Programming Smart House is usable** (Method Description: Programming the smart house is usable from end user perspective; Test Case: all; Usability Goal: Quality in Use and UG1.1; Usability Requirement: all).

Lastly, you should create:
Usability Requirement with name **Effectiveness** (Description/Description old: programming the smart house is effective; Metric: Correctness of programmed code; Old Name: Effectiveness),
Usability Requirement with name **Learnability** (Description/Description old: programming the smart house is easy to learn; Metric: Reused knowledge; Old Name: Learnability) ,
Usability Requirement with name **Satisfaction** (Description/Description old: programming the smart house is satisfactory; Metric: Satisfaction questions; Old Name: Satisfaction) , and
Usability Requirement with name **Efficiency** (Description/Description old: programming the smart house is efficient; Metric: Efficient programming; Old Name: Efficiency).

4. Evaluation Specification

First, you should create **Evaluation Specification** with name **es**. Then within **Evaluation Specification es** you should create **Evaluation Model** with name **emSH**. After that you should create only one **Language** with name **Smart House** (DSL: Smart House; Version: NOVA-LINCS), since you are not doing a comparative evaluation.
Next, you should create:

Evaluation Goal with name **egSHEffectiveness** (Comparative: false; Hypothesis = {effectiveness has no impact in smart house programming, effectiveness has impact in smart house programming}; Problem Description: analyse the impact of effectiveness on smart house programming; Research Question: is it effective to program the smart house?; Usability Goal: UG1.1), and
Evaluation Goal with name **egSHSatisfaction** (Comparative: false; Hypothesis: {satisfaction has no impact in smart house programming, satisfaction has impact in smart house programming}; Problem Description: analyse the impact of satisfaction on smart house programming; Research Question: is it satisfactory to program the smart house?; Usability Goal: UG1.1).

The participants chosen for the Evaluation are adults, so you should create:
Participant with name **Adults** (Contact: Personal contact; User Profile: Adults).

After that you should create **Evaluation Context** with name **ecSH** (Context Environment: all; Context Model: cmSH; Environment Instance: Smart House={NOVA-LINCS}, Computer={Desktop}, Application={Computer app}, Workplace={Classroom}; Scenario: all; Usability Goal: UG1.1; User Profile Selection: Adults; Workflow: W1).

Next, you should provide some **Documentation** with name **doc** (Evaluation Model: emSH; Scenario: all) for the Scenarios.

Lastly, you should create the **Process** with name **EvaluationProcess** (Evaluation Model: emSH).

5. Interaction Specification

First, you should create **Interaction Specification** with name **is.** Then within **Interaction Specification is** you should create **Interaction Model** with name **imSH** (Evaluation Model: emSH; Participant: Adults).
Next, you should create **Task** with name **taskSH** (Documentation: all; Scenario: all).
After that, you should create **Interaction Syntax** with name **isSH** (Documentation: all; Interaction Model: imSH; Outside Ref: refSHModeling).

Next, you should create **Event** with name **EffectivenessVideo** (Analysis Type: Observation; Capture: {Wake up alarm rings, Opens Windows}; Interaction Model: imSH; Manual: true; Record Event: Screen Record; Usability Requirement: Effectiveness). And within **Event EffectivenessVideo**, create:
Capture Action with name **Wake up alarm rings,** and **Capture Action** with name **Opens Windows**.

After evaluating the DSL you created **Interaction Result** with name **irSH** (Event: Effectiveness Video; Interaction Model: imSH; Outside Ref: refSHModeling) and extracted the following result values**:**
Result Value with name **Wake up alarm rings** (Associated Requirement: Effectiveness; Language: Smart House; Result Value: 0,78), and

Result Value with name **Opens Windows** (Associated Requirement: Effectiveness; Language: Smart House; Result Value: 0,75).

6. Survey Specification

First, you should create **Survey Specification** with name **ss.** Then within **Survey Specification ss** you should create **Survey Model** with name **smSH** (Participant: Adults; Survey Engine: Survey Engine).

Next you should create: **Questionnaire** with name **Background Questions** (Survey Model: smSH) and **Questionnaire** with name **Feedback Questions** (Survey Model: smSH; Usability Requirement: Satisfaction).

After that you should define:
Background Qs with name **Q1** (Logical Expression: Age Adults; Question: Age; Scale: Integer; Type: Demographics; User Profile: Adults),
Background Qs with name **Q2** (Logical Expression: Computers; Question: Have you ever programmed?; Scale: {Yes, No}; Type: Experience; User Profile: Adults),
Background Qs with name **Q3** (Logical Expression: House Automation; Question: Have you ever interacted with a smart house?; Scale: {Yes, No}; Type: Experience; User Profile: Adults),
Feedback Qs with name **F1** (Question: Did you enjoy the activity?; Scale: {Yes, No}; Scenario: all; Type: Likeability),
Feedback Qs with name **F2** (Question: Did you find it hard to program the smart house to detect the wake up alarm?; Scale: {Yes, No}; Scenario: all; Type: Confidence),
Feedback Qs with name **F3** (Question: And to open the windows after detecting the alarm?; Scale: {Yes, No}; Scenario: all; Type: Confidence), and
Feedback Qs with name **F4** (Question: Would you like to repeat this activity?; Scale: {Yes, No}; Scenario: all; Type: Confidence).

After evaluating the DSL you should create **Survey Result** with name **srSH** (Outside Reference: refSHModeling; Questionnaire: Background and Feedback Questions) and extracted the following result values**:**
Result Value with name **Q1** (Language: Smart House; Related Question: Q1; Result Value: 25,5),
Result Value with name **Q2** (Language: Smart House; Related Question: Q2; Result Value: 0,6),
Result Value with name **Q3** (Language: Smart House; Related Question: Q3; Result Value: 0,71),
Result Value with name **F1** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F1; Result Value: 0,94),
Result Value with name **F2** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F2; Result Value: 0,52),
Result Value with name **F3** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F3; Result Value: 0,53), and

Result Value with name **F4** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F4; Result Value: 0,95).

7. Report Specification

First, you should create **Report Specification** with name **rs.** Then within **Report Specification rs** you should create **Report Model** with name **rmSH**. Next you should define **Evaluation Result** with name **erSH** and within:
Result Value with name **EffectivenessSH** (Related Action: all; Result Value: 0,78), and **Result Value** with name **SatisfactionSH** (Related Question: F1, F2, F3, F4; Result Value: 0,85).

Next, you should create **Recommended GM** with name **rgmSH** (Functional Goal: fgSH; Refers To: gmSH; Report Model: rmSH; Suggested Requirements: Zooming; Usability Goal: UG1.1).

Lastly, you should create **Success Coverage** with name **scSH** (Scope: all; Success Factor: all; Usability Goal: UG1.1) in **Goal Specification** with name **gs**.

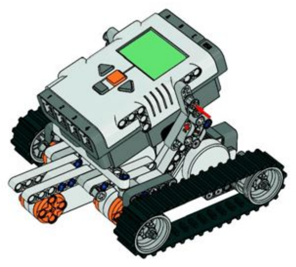
Thank you!

Appendix D

USE-ME New Version: Lego Exercise

In this appendix, we present the Lego exercise for the new USE-ME version.

LEGO Mindstorms



The main goal of this exercise is to perform an usability evaluation on a DSL under development.
__Important__ __note__: You **must** "Validate" the model every time you make changes, and focus only on the errors that start with "USE-ME (Development phase) Error/Suggestion" (e.g. "USE-ME Utility Specification Error or Suggestion").

How to "Validate"?

Right click on the UseMe Model root "platform:/resource/(Modeling Project name)/(UseMe Model name).useme" > "Validate".
It is **recommended** to follow the rules that the tool generates.

Also the "Validate" action **does not save** the file, so you should save it every time you make important changes.

LEGO Mindstorms are programmable robots designed for children. The main purpose of these robots is to teach kids to code, with some basics notions on coding while they play, and for that purpose a DSL Lego was developed.

1. Specification

First, you should create **Specification** with name **US**, within **Specification US** you should create a **DSL** with name **Lego**. Within **DSL Lego** you should create a **Concrete Syntax** with name **csLego,** an **Abstract Syntax** with name **asLego,** and an **ExistingGM** with name **gmLego**.
In **Specification US** you should also create a **Functional Goal** with name **fgLego**, a **Process Model** with name **pmLego**, a **Survey Engine**, a **Documentation** with name **docLego**, an **Outside Ref** with name **refLegoModeling** (Link: cameo.com; Tool: Cameo System Modeler), and a **Requirement** with name **Zooming** (Description: improve zooming option).

2. Context Specification

First, you should create **Context Specification** with name **cs.** Within **Context Specification cs** you should create a **Context Model** with Cm name **cmLego** (Context Provider: FCT; Domain: Program a robot). Within **Context Model cmLego** you should create an **User Hierarchy** (Uh Description: uhLego).

Next, you should create an **User Profile Specification** with name **upsLego**. Within you should create **User Profile** with name **DSL Stakeholder** (Priority: high, Sub Profile: End User),
User Profile with name **End User** (Priority: high, Sub Profile: Children and Adults),
User Profile with name **Children** (Priority: high, Sub Profile: Grade-Schoolers and Teens),
User Profile with name **Grade-Schoolers** (Priority: high),
User Profile with name **Teens** (Priority: medium), and
User Profile with name **Adults** (Priority: medium).

Next, you should create: **Profile Template** with name **DSL Stakeholder, Profile Template** with name **End User, Profile Template** with name **Children, Profile Template** with name **Grade-Schoolers, Profile Template** with name **Teens**, and **Profile Template** with name **Adults** with **Category:** background demographics knowledge.

After that, you should create:
Logical Expression with name **Age** (Classifier: Age; Expression: >5; Profile Template: DSL Stakeholder, End User),
Logical Expression with name **Age Children** (Classifier: Age Children; Expression: 5-18; Profile Template: Children),
Logical Expression with name **Age Grade-Schoolers** (Classifier: Age Grade-Schoolers; Expression: 5-12; Profile Template: Grade Schoolers),
Logical Expression with name **Age Teens** (Classifier: Age Teens; Expression: 13-18; Profile Template: Teens),
Logical Expression with name **Age Adults** (Classifier: Age Adults; Expression: >18; Profile Template: Adults),
Logical Expression with name **School Grade** (Classifier: School Grade; Expression: 1-12; Profile Template: Children, Grade Schoolers, Teens),
Logical Expression with name **Computers** (Classifier: Computers; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates),
Logical Expression with name **Programming** (Classifier: Programming; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates).

Next, you should create an **Environment Specification** with name **esLego**. Within **Environment Specification esLego** you should create: **Technical Environment** with name **teLego**, **Physical Environment** with name **peLego**, and **Social Environment** with name **seLego**.
After that, you should create:
CE Variable with name **Robot** (Context Environment: Physical Environment pe; Mandatory: false; Type: Mindstorms),
CE Variable with name **Computer** (Context Environment: Physical Environment pe; Mandatory: true; Type: Desktop),
CE Variable with name **Application** (Context Environment: Technical Environment te; Mandatory: true; Type: Computer App), and
CE Variable with name **Workplace** (Context Environment: Social Environment se; Mandatory: true; Type: Classroom).

You should also create inside each CE Variable the correspondent CE Variable type (e.g. **CE Variable** with name **Mindstorms** within **CE Variable Robot**).

Lastly, you should create **Workflow Specification** with name **wsLego**, and then:
Workflow with name **W1: Program the robot** (Actor: End User; Context Element: Robot; Context Model: cmLego, Priority: High; Process Model: pmLego), and
Workflow with name **W2: Modify the language** (Actor: Language Engineer; Context Element: Application; Context Model: cmLego, Priority: Low; Process Model: pmLego).
You should create two scenarios for **Workflow W1: Program the robot:**
Scenario with name **Move front and back,** and
Scenario with name **Move forward until it hits an obstacle and then stop.**

3. Goal Specification

First, you should create **Goal Specification** with name **gs**. Then within **Goal Specification gs** you should create a **Goal Model** with name **gmLego**. Within **Goal Model gmLego** you should create:
Usability Goal with name **Quality in Use** (Priority: High; Question: Is the Quality in Use achieved?; Sub Goal: UG1 and UG2) this means that the DSL is usable,
Usability Goal with name **UG1: Capability to program the robot** (Priority: High; Question: Are the End Users capable of program the robot?; Sub Goal: UG1.1),
Usability Goal with name **UG1.1: Usability of programming the robot** (Priority: High; Question: Is it usable to program the robot?), and
Usability Goal with name **UG2: Evolve the language** (Priority: Medium; Question: Are Language Engineers capable of evolve the language?).

Next, you should create:
Scope with name **QualityInUse** (Context Environment: all; Context Model: cmLego; Usability Goal: Quality in Use; User Profile Selection: DSL Stakeholder; Workflow: all),
Scope with name **CapabilityProgramRobot** (Context Environment: all; Context Model: cmLego; Usability Goal: UG1 and UG1.1; User Profile Selection: End User; Workflow: W1), and
Scope with name **EvolveLanguage** (Context Environment: all; Context Model: cmLego; Usability Goal: UG2; User Profile Selection: Language Engineer; Workflow: W2).

After that, you should create:
Actor with name **Lego Development** (Organization: Lego Dev; Responsible For: Quality in Use, UG1 and UG2; Stakeholder: Language Engineer), and
Actor with name **Expert Evaluator** (Organization: Language Evaluator; Responsible For: all; Stakeholder: Expert Evaluator).

You should also create **Method** with name **Programming Robot is usable** (Method Description: Programming the robot is usable from end user perspective; Test Case: all; Usability Goal: Quality in Use and UG1.1; Usability Requirement: all).

Lastly, you should create:

Usability Requirement with name **Effectiveness** (Description/Description old: programming the robot is effective; Metric: Correctness of programmed code; Old Name: Effectiveness),
Usability Requirement with name **Learnability** (Description/Description old: programming the robot is easy to learn; Metric: Reused knowledge; Old Name: Learnability) ,
Usability Requirement with name **Satisfaction** (Description/Description old: programming the robot is satisfactory; Metric: Satisfaction questions; Old Name: Satisfaction) , and
Usability Requirement with name **Efficiency** (Description/Description old: programming the robot is efficient; Metric: Efficient programming; Old Name: Efficiency).

4. Evaluation Specification

First, you should create **Evaluation Specification** with name **es**. Then within **Evaluation Specification es** you should create **Evaluation Model** with name **emLego**. After that you should create only one **Language** with name **Lego** (DSL: Lego; Version: Mindstorms), since you are not doing a comparative evaluation.
Next, you should create:
Evaluation Goal with name **egLegoEffectiveness** (Comparative: false; Hypothesis = {effectiveness has no impact in robot programming, effectiveness has impact in robot programming}; Problem Description: analyse the impact of effectiveness on robot programming; Research Question: is it effective to program the robot?; Usability Goal: UG1.1), and
Evaluation Goal with name **egLegoSatisfaction** (Comparative: false; Hypothesis: {satisfaction has no impact in robot programming, satisfaction has impact in robot programming}; Problem Description: analyse the impact of satisfaction on robot programming; Research Question: is it satisfactory to program the robot?; Usability Goal: UG1.1).

The participants chosen for the Evaluation are children, so you should create:
Participant with name **Children** (Contact: Teacher contact; User Profile: Children).

After that you should create **Evaluation Context** with name **ecLego** (Context Environment: all; Context Model: cmLego; Environment Instance: Robot={Lego Mindstorms}, Computer={Desktop}, Application={Computer app}, Workplace={Classroom}; Scenario: all; Usability Goal: UG1.1; User Profile Selection: Children; Workflow: W1).

Next, you should provide some **Documentation** with name **doc** (Evaluation Model: emLego; Scenario: all) for the Scenarios.

Lastly, you should create the **Process** with name **EvaluationProcess** (Evaluation Model: emLego).

5. Interaction Specification

First, you should create **Interaction Specification** with name **is.** Then within **Interaction Specification is** you should create **Interaction Model** with name **imLego** (Evaluation

Model: emLego; Participant: Children). Next, you should create **Task** with name **taskLego** (Documentation: all; Scenario: all).
After that, you should create **Interaction Syntax** with name **isLego** (Documentation: all; Interaction Model: imLego; Outside Ref: refLegoModeling).

Next, you should create **Event** with name **EffectivenessVideo** (Analysis Type: Observation; Capture: {Move front, Move back, Bump}; Interaction Model: imLego; Manual: true; Record Event: Screen Record; Usability Requirement: Effectiveness). And within **Event EffectivenessVideo,** create:
Capture Action with name **Move front**, **Capture Action** with name **Move back**, and **Capture Action** with name **Bump**.

After evaluating the DSL you created **Interaction Result** with name **irLego** (Event: Effectiveness Video; Interaction Model: imLego; Outside Ref: refLegoModeling) and extracted the following result values**:**
Result Value with name **MoveFront** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,75),
Result Value with name **MoveBack** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,75), and
Result Value with name **Bump** (Associated Requirement: Effectiveness; Language: Lego; Result Value: 0,78).

6. Survey Specification

First, you should create **Survey Specification** with name **ss.** Then within **Survey Specification ss** you should create **Survey Model** with name **smLego** (Participant: Children; Survey Engine: Survey Engine).

Next you should create: **Questionnaire** with name **Background Questions** (Survey Model: smLego) and **Questionnaire** with name **Feedback Questions** (Survey Model: smLego; Usability Requirement: Satisfaction).

After that you should define:
Background Qs with name **Q1** (Logical Expression: Age Children; Question: Age; Scale: Integer; Type: Demographics; User Profile: Children),
Background Qs with name **Q2** (Logical Expression: School Grade; Question: School grade; Scale: Integer; Type: Demographics; User Profile: Children),
Background Qs with name **Q3** (Logical Expression: Computers; Question: How often do you play computer games?; Scale: {Every day, Sometimes, Rarely}; Type: Experience; User Profile: Children),
Background Qs with name **Q4** (Logical Expression: Programming; Question: Have you ever programmed?; Scale: {Yes, No}; Type: Experience; User Profile: Children),
Feedback Qs with name **F1** (Question: Did you enjoy the activity?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Likeability),
Feedback Qs with name **F2** (Question: Did you find it hard to move the robot front and back?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence),

Feedback Qs with name **F3** (Question: And to move front until it hits an obstacle and then stop?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence), and **Feedback Qs** with name **F4** (Question: Would you like to repeat this activity?; Scale: {Smiley face, Neutral face, Sad face}; Scenario: all; Type: Confidence).

After evaluating the DSL you should create **Survey Result** with name **srLego** (Outside Reference: refLegoModeling; Questionnaire: Background and Feedback Questions) and extracted the following result values**:**
Result Value with name **Q1** (Language: Lego; Related Question: Q1; Result Value: 11,5),
Result Value with name **Q2** (Language: Lego; Related Question: Q2; Result Value: 6),
Result Value with name **Q3** (Language: Lego; Related Question: Q3; Result Value: 0,71),
Result Value with name **Q4** (Language: Lego; Related Question: Q4; Result Value: 0,8),
Result Value with name **F1** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F1; Result Value: 0,94),
Result Value with name **F2** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F2; Result Value: 0,52),
Result Value with name **F3** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F3; Result Value: 0,53), and
Result Value with name **F4** (Associated Requirement: Satisfaction; Language: Lego; Related Action: all; Related Question: F4; Result Value: 0,95).

7. Report Specification

First, you should create **Report Specification** with name **rs.** Then within **Report Specification rs** you should create **Report Model** with name **rmLego**. Next you should define **Evaluation Result** with name **erLego** and within:
Result Value with name **EffectivenessLego** (Related Action: all; Result Value: 0,78), and
Result Value with name **SatisfactionLego** (Related Question: F1, F2, F3, F4; Result Value: 0,85).

Next, you should create **Recommended GM** with name **rgmLego** (Functional Goal: fgLego; Refers To: gmLego; Report Model: rmLego; Suggested Requirements: Zooming; Usability Goal: UG1.1).

Lastly, you should create **Success Coverage** with name **scLego** (Scope: all; Success Factor: all; Usability Goal: UG1.1) in **Goal Specification** with name **gs**.

Thank you!

Appendix E

USE-ME New Version: Smart House Exercise

In this appendix, we present the Smart House exercise for the new USE-ME version.

Smart House

The main goal of this exercise is to perform an usability evaluation on a DSL under development.
<u>Important</u> <u>note</u>: You **must** "Validate" the model every time you make changes, and focus only on the errors that start with "USE-ME (Development phase) Error/Suggestion" (e.g. "USE-ME Utility Specification Error or Suggestion").

How to "Validate"?

Right click on the UseMe Model root "platform:/resource/(Modeling Project name)/(UseMe Model name).useme" > "Validate".
It is **recommended** to follow the rules that the tool generates.

Also the "Validate" action **does not save** the file, so you should save it every time you make important changes.

A Smart House is a collection of technical home automation concepts that are integrated together to meet the user goals, and for that purpose a DSL Smart House was developed.

1. Specification

First, you should create **Specification** with name **us.** Then within **Specification us** you should create a **DSL** with name **Smart House**. Within **DSL Smart House** you should create a **Concrete Syntax** with name **csSH,** an **Abstract Syntax** with name **asSH,** and an **ExistingGM** with name **gmSH**.
In **Specification us** you should also create a **Functional Goal** with name **fgSH**, a **Process Model** with name **pmSH**, a **Survey Engine**, a **Documentation** with name **docSH**, an **Outside Ref** with name **refSHModeling** (Link: cameo.com; Tool: Cameo System Modeler) , and a **Requirement** with name **Zooming** (Description: improve zooming option).

2. Context Specification

First, you should create **Context Specification** with name **cs**. Then within **Context Specification cs** you should create **Context Model** with Cm name **cmSH** (Context provider: FCT; Domain: Program a smart house). Within **Context Model cmSH** you should create an **User Hierarchy** (Uh Description: uhSH).

Next, you should create an **User Profile Specification** with name **upsSH**. Within you should create **User Profile** with name **DSL Stakeholder** (Priority: high, Sub Profile: End User),
User Profile with name **End User** (Priority: high, Sub Profile: Adults and Teens),
User Profile with name **Adults** (Priority: high, Sub Profile: Young adults and Middle adults),
User Profile with name **Young Adults** (Priority: high),
User Profile with name **Middle Adults** (Priority: medium), and
User Profile with name **Teens** (Priority: medium).

Next, you should create: **Profile Template** with name **DSL Stakeholder, Profile Template** with name **End User, Profile Template** with name **Adults, Profile Template** with name **Young Adults, Profile Template** with name **Middle Adults**, and **Profile Template** with name **Teens** with **Category:** background demographics knowledge.

After that, you should create:
Logical Expression with name **Age** (Classifier: Age; Expression: >13; Profile Template: DSL Stakeholder, End User),
Logical Expression with name **Age Adults** (Classifier: Age Adults; Expression: 20-64; Profile Template: Adults),
Logical Expression with name **Age Young Adults** (Classifier: Age Young Adults; Expression: 20-40; Profile Template: Young Adults),
Logical Expression with name **Age Middle Adults** (Classifier: Age Middle Adults; Expression: 40-64; Profile Template: Middle Adults),
Logical Expression with name **Age Teens** (Classifier: Age Teens; Expression: 13-19; Profile Template: Teens),
Logical Expression with name **Computers** (Classifier: Computers; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates),
Logical Expression with name **House Automation** (Classifier: House Automation; Expression: Ordinal, Scale, Experience; Profile Template: all Profile Templates).

Next, you should create an **Environment Specification** with name **esSH**. Within **Environment Specification esSH** you should create: **Technical Environment** with name **teSH**, **Physical Environment** with name **peSH**, and **Social Environment** with name **seSH**.
After that, you should create:
CE Variable with name **Smart House** (Context Environment: Physical Environment pe; Mandatory: false; Type: NOVA-LINCS),
CE Variable with name **Computer** (Context Environment: Physical Environment pe; Mandatory: true; Type: Desktop),
CE Variable with name **Application** (Context Environment: Technical Environment te; Mandatory: true; Type: Computer App), and
CE Variable with name **Workplace** (Context Environment: Social Environment se; Mandatory: true; Type: Classroom).
You should also create inside each CE Variable the correspondent CE Variable type (e.g. **CE Variable** with name **NOVA-LINCS** within **CE Variable Smart House**).

Lastly, you should create **Workflow Specification** with name **wsSH**, and then:

Workflow with name **W1: Program the Smart House** (Actor: End User; Context Element: Smart House; Context Model: cmSH, Priority: High; Process Model: pmSH), and
Workflow with name **W2: Modify the language** (Actor: Language Engineer; Context Element: Application; Context Model: cmSH, Priority: Low; Process Model: pmSH).
You should create two scenarios for **Workflow W1: Program the Smart House:**
Scenario with name **When front door opens says Hello,** and
Scenario with name **When alarm rings Smart House opens windows**.

3. Goal Specification

First, you should create **Goal Specification** with name **gs**. Then within **Goal Specification gs** you should create a **Goal Model** with name **gmSH**. Within **Goal Model gmSH** you should create:
Usability Goal with name **Quality in Use** (Priority: High; Question: Is the Quality in Use achieved?; Sub Goal: UG1 and UG2) this means that the DSL is usable,
Usability Goal with name **UG1: Capability to program the smart house** (Priority: High; Question: Are the End Users capable of program the smart house?; Sub Goal: UG1.1),
Usability Goal with name **UG1.1: Usability of programming the smart house** (Priority: High; Question: Is it usable to program the smart house?), and
Usability Goal with name **UG2: Evolve the language** (Priority: Medium; Question: Are Language Engineers capable of evolve the language?).

Next, you should create:
Scope with name **QualityInUse** (Context Environment: all; Context Model: cmSH; Usability Goal: Quality in Use; User Profile Selection: DSL Stakeholder; Workflow: all),
Scope with name **CapabilityProgramSmartHouse** (Context Environment: all; Context Model: cmSH; Usability Goal: UG1 and UG1.1; User Profile Selection: End User; Workflow: W1), and
Scope with name **EvolveLanguage** (Context Environment: all; Context Model: cmSH; Usability Goal: UG2; User Profile Selection: Language Engineer; Workflow: W2).

After that, you should create:
Actor with name **Smart House Development** (Organization: Smart House Dev; Responsible For: Quality in Use, UG1 and UG2; Stakeholder: Language Engineer), and
Actor with name **Expert Evaluator** (Organization: Language Evaluator; Responsible For: all; Stakeholder: Expert Evaluator).
You should also create **Method** with name **Programming Smart House is usable** (Method Description: Programming the smart house is usable from end user perspective; Test Case: all; Usability Goal: Quality in Use and UG1.1; Usability Requirement: all).

Lastly, you should create:
Usability Requirement with name **Effectiveness** (Description/Description old: programming the smart house is effective; Metric: Correctness of programmed code; Old Name: Effectiveness),
Usability Requirement with name **Learnability** (Description/Description old: programming the smart house is easy to learn; Metric: Reused knowledge; Old Name: Learnability) ,

Usability Requirement with name **Satisfaction** (Description/Description old: programming the smart house is satisfactory; Metric: Satisfaction questions; Old Name: Satisfaction) , and **Usability Requirement** with name **Efficiency** (Description/Description old: programming the smart house is efficient; Metric: Efficient programming; Old Name: Efficiency).

4. Evaluation Specification

First, you should create **Evaluation Specification** with name **es**. Then within **Evaluation Specification es** you should create **Evaluation Model** with name **emSH**. After that you should create only one **Language** with name **Smart House** (DSL: Smart House; Version: NOVA-LINCS), since you are not doing a comparative evaluation.
Next, you should create:
Evaluation Goal with name **egSHEffectiveness** (Comparative: false; Hypothesis = {effectiveness has no impact in smart house programming, effectiveness has impact in smart house programming}; Problem Description: analyse the impact of effectiveness on smart house programming; Research Question: is it effective to program the smart house?; Usability Goal: UG1.1), and
Evaluation Goal with name **egSHSatisfaction** (Comparative: false; Hypothesis: {satisfaction has no impact in smart house programming, satisfaction has impact in smart house programming}; Problem Description: analyse the impact of satisfaction on smart house programming; Research Question: is it satisfactory to program the smart house?; Usability Goal: UG1.1).

The participants chosen for the Evaluation are adults, so you should create:
Participant with name **Adults** (Contact: Personal contact; User Profile: Adults).

After that you should create **Evaluation Context** with name **ecSH** (Context Environment: all; Context Model: cmSH; Environment Instance: Smart House={NOVA-LINCS}, Computer={Desktop}, Application={Computer app}, Workplace={Classroom}; Scenario: all; Usability Goal: UG1.1; User Profile Selection: Adults; Workflow: W1).

Next, you should provide some **Documentation** with name **doc** (Evaluation Model: emSH; Scenario: all) for the Scenarios.

Lastly, you should create the **Process** with name **EvaluationProcess** (Evaluation Model: emSH).

5. Interaction Specification

First, you should create **Interaction Specification** with name **is.** Then within **Interaction Specification is** you should create **Interaction Model** with name **imSH** (Evaluation Model: emSH; Participant: Adults).
Next, you should create **Task** with name **taskSH** (Documentation: all; Scenario: all).
After that, you should create **Interaction Syntax** with name **isSH** (Documentation: all; Interaction Model: imSH; Outside Ref: refSHModeling).

Next, you should create **Event** with name **EffectivenessVideo** (Analysis Type: Observation; Capture: {Wake up alarm rings, Opens Windows}; Interaction Model: imSH; Manual: true; Record Event: Screen Record; Usability Requirement: Effectiveness). And within **Event EffectivenessVideo**, create:
Capture Action with name **Wake up alarm rings,** and **Capture Action** with name **Opens Windows**.

After evaluating the DSL you created **Interaction Result** with name **irSH** (Event: Effectiveness Video; Interaction Model: imSH; Outside Ref: refSHModeling) and extracted the following result values**:**
Result Value with name **Wake up alarm rings** (Associated Requirement: Effectiveness; Language: Smart House; Result Value: 0,78), and
Result Value with name **Opens Windows** (Associated Requirement: Effectiveness; Language: Smart House; Result Value: 0,75).

6. Survey Specification

First, you should create **Survey Specification** with name **ss.** Then within **Survey Specification ss** you should create **Survey Model** with name **smSH** (Participant: Adults; Survey Engine: Survey Engine).

Next you should create: **Questionnaire** with name **Background Questions** (Survey Model: smSH) and **Questionnaire** with name **Feedback Questions** (Survey Model: smSH; Usability Requirement: Satisfaction).

After that you should define:
Background Qs with name **Q1** (Logical Expression: Age Adults; Question: Age; Scale: Integer; Type: Demographics; User Profile: Adults),
Background Qs with name **Q2** (Logical Expression: Computers; Question: Have you ever programmed?; Scale: {Yes, No}; Type: Experience; User Profile: Adults),
Background Qs with name **Q3** (Logical Expression: House Automation; Question: Have you ever interacted with a smart house?; Scale: {Yes, No}; Type: Experience; User Profile: Adults),
Feedback Qs with name **F1** (Question: Did you enjoy the activity?; Scale: {Yes, No}; Scenario: all; Type: Likeability),
Feedback Qs with name **F2** (Question: Did you find it hard to program the smart house to detect the wake up alarm?; Scale: {Yes, No}; Scenario: all; Type: Confidence),
Feedback Qs with name **F3** (Question: And to open the windows after detecting the alarm?; Scale: {Yes, No}; Scenario: all; Type: Confidence), and
Feedback Qs with name **F4** (Question: Would you like to repeat this activity?; Scale: {Yes, No}; Scenario: all; Type: Confidence).

After evaluating the DSL you should create **Survey Result** with name **srSH** (Outside Reference: refSHModeling; Questionnaire: Background and Feedback Questions) and extracted the following result values**:**

Result Value with name **Q1** (Language: Smart House; Related Question: Q1; Result Value: 25,5),
Result Value with name **Q2** (Language: Smart House; Related Question: Q2; Result Value: 0,6),
Result Value with name **Q3** (Language: Smart House; Related Question: Q3; Result Value: 0,71),
Result Value with name **F1** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F1; Result Value: 0,94),
Result Value with name **F2** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F2; Result Value: 0,52),
Result Value with name **F3** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F3; Result Value: 0,53), and
Result Value with name **F4** (Associated Requirement: Satisfaction; Language: Smart House; Related Action: all; Related Question: F4; Result Value: 0,95).

7. Report Specification

First, you should create **Report Specification** with name **rs.** Then within **Report Specification rs** you should create **Report Model** with name **rmSH**. Next you should define **Evaluation Result** with name **erSH** and within:
Result Value with name **EffectivenessSH** (Related Action: all; Result Value: 0,78), and
Result Value with name **SatisfactionSH** (Related Question: F1, F2, F3, F4; Result Value: 0,85).

Next, you should create **Recommended GM** with name **rgmSH** (Functional Goal: fgSH; Refers To: gmSH; Report Model: rmSH; Suggested Requirements: Zooming; Usability Goal: UG1.1).

Lastly, you should create **Success Coverage** with name **scSH** (Scope: all; Success Factor: all; Usability Goal: UG1.1) in **Goal Specification** with name **gs**.

Thank you!

Appendix F

Experiment Presentation

In this appendix, we present the presentation on the USE-ME framework. The main goal of this presentation was to introduce participants to USE-ME development phases, interface, stakeholders, etc.

NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

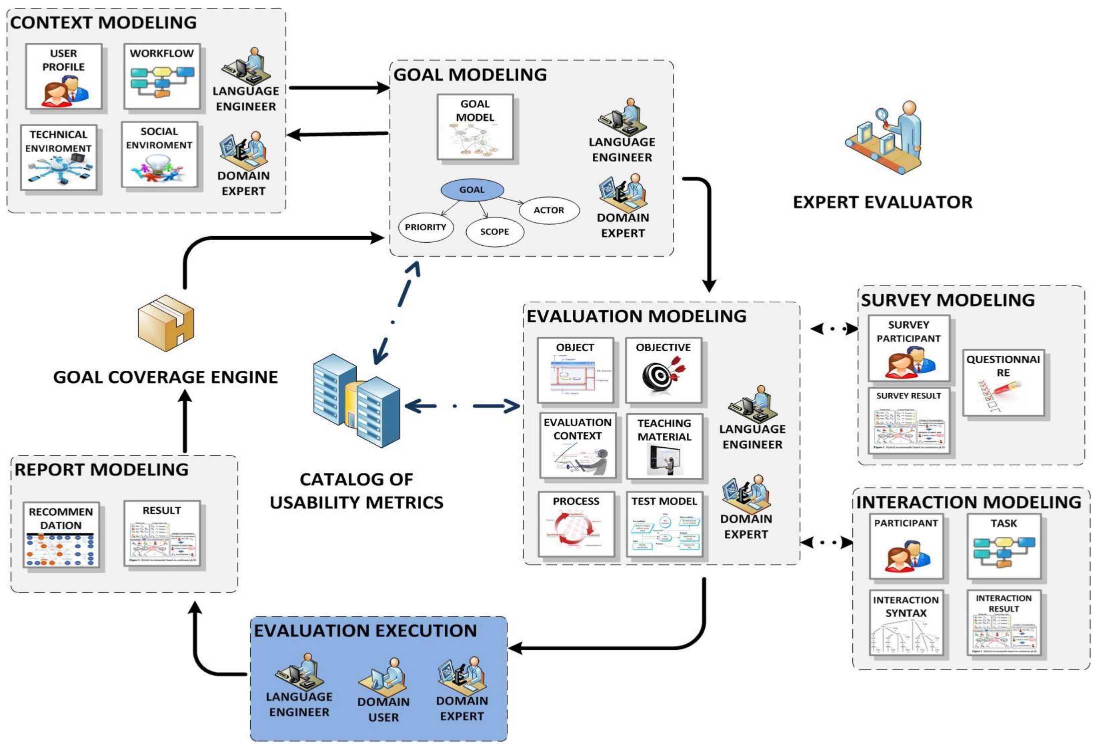
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

A Framework for Experimental Validation of Domain Specific Languages

Prof. Vasco Amaral, Prof. Miguel Goulão, Ankica Barišić and **Sara Rosa**

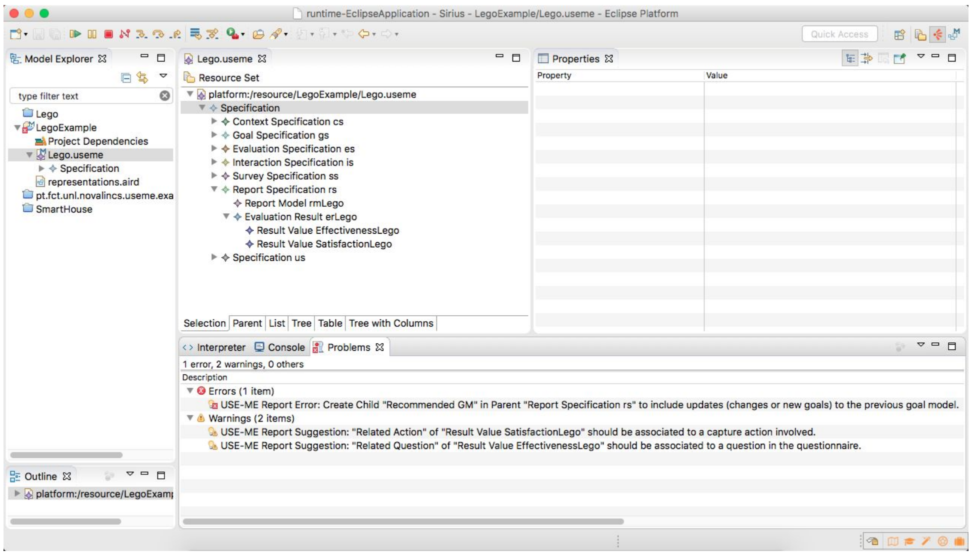
Universidade NOVA de Lisboa

USE-ME Lifecycle



Validation Rules

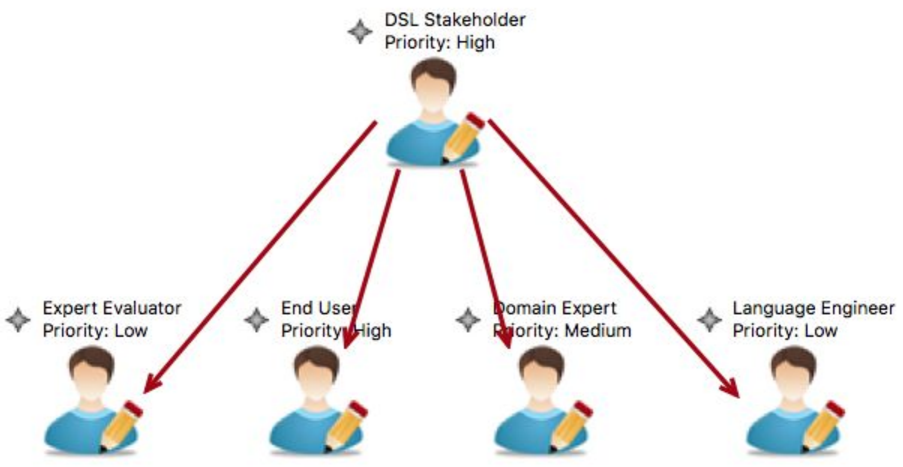
- The **rules** are classified in the following way:
  - Strict: if an activity is mandatory;
  - Suggestion: if an activity is not mandatory.



1 error, 2 warnings, 0 others

Description
- Errors (1 item)
  - USE-ME Report Error: Create Child "Recommended GM" in Parent "Report Specification rs" to include updates (changes or new goals) to the previous goal model.
- Warnings (2 items)
  - USE-ME Report Suggestion: "Related Action" of "Result Value SatisfactionLego" should be associated to a capture action involved.
  - USE-ME Report Suggestion: "Related Question" of "Result Value EffectivenessLego" should be associated to a question in the questionnaire.

User Hierarchy



Appendix G

Background Questionnaire

In this appendix, we present the background questionnaire that was performed before the first exercise. The main goal of this questionnaire was to gather participants demographic data, background knowledge, etc.

Background Questionnaire

This experimental work is conducted within the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). NOVA LINCS is a new unit of the national Science & Technology network in the area of Computer Science and Engineering, launched in 2014/2015, and hosted at the Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa (DI-NOVA), a leading academic department in Portugal.

All information stated as part of this experiment is confidential and will be kept as such.

Prof. Vasco Amaral and Prof. Miguel Goulão are responsible for this experiment and can be contacted at:

- Prof. Vasco Amaral: vasco.amaral@fct.unl.pt; +351 212 948 300 (ext. 10712); Office P2/3
- Prof. Miguel Goulão: mgoul@fct.unl.pt; +351 212 948 536 (ext. 10731); Office P2/17

We would like to emphasize that:
- your participation is entirely voluntary;
- you are free to refuse to answer any question;
- you are free to withdraw at any time.

The experiment will be kept strictly confidential and will be made available only to members of the research team of the study or, in case external quality assessment takes place, to assessors under the same confidentiality conditions. Data collected in this experiment may be part of the final research report, but under no circumstances will your name or any identifying characteristic be included in the report.

*Obrigatório

1. **I agree with the terms.** *

Marcar apenas uma oval.

- Yes
- No *Pare de preencher este formulário.*

2. **Age** *

____

3. **Completed academic degree** *

Marcar apenas uma oval.

- BSc
- MSc
- PhD

4. **Degree in** *

Marcar apenas uma oval.

- Computer Science and Engineering
- Outra: ______________________________

5. **Have you ever used Domain Specific Languages?** *

Marcar apenas uma oval.

- ◯ Yes
- ◯ No *Pare de preencher este formulário.*

6. **Please state in which context have you used Domain Specific Languages** *

Marcar tudo o que for aplicável.

- ☐ Academic
- ☐ Industry
- ☐ Outra: ____________________

7. **Have you ever modeled Domain Specific Languages?** *

Marcar apenas uma oval.

- ◯ Yes
- ◯ No *Pare de preencher este formulário.*

8. **Please state in which context have you modeled Domain Specific Languages** *

Marcar tudo o que for aplicável.

- ☐ Academic
- ☐ Industry
- ☐ Outra: ____________________

Appendix H

Feedback Questionnaire

In this appendix, we present the feedback questionnaire that was performed after the modeling exercises. The main goal of this questionnaire was to collect participants feedback about the USE-ME versions (original and new).

Feedback Questionnaire

This experimental work is conducted within the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). NOVA LINCS is a new unit of the national Science & Technology network in the area of Computer Science and Engineering, launched in 2014/2015, and hosted at the Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa (DI-NOVA), a leading academic department in Portugal.

All information stated as part of this experiment is confidential and will be kept as such.

Prof. Vasco Amaral and Prof. Miguel Goulão are responsible for this experiment and can be contacted at:

- Prof. Vasco Amaral: vasco.amaral@fct.unl.pt; +351 212 948 300 (ext. 10712); Office P2/3
- Prof. Miguel Goulão: mgoul@fct.unl.pt; +351 212 948 536 (ext. 10731); Office P2/17

We would like to emphasize that:
- your participation is entirely voluntary;
- you are free to refuse to answer any question;
- you are free to withdraw at any time.

The experiment will be kept strictly confidential and will be made available only to members of the research team of the study or, in case external quality assessment takes place, to assessors under the same confidentiality conditions. Data collected in this experiment may be part of the final research report, but under no circumstances will your name or any identifying characteristic be included in the report.

*Obrigatório

1. **I agree with the terms.** *
   Marcar apenas uma oval.

   ◯ Yes
   ◯ No *Pare de preencher este formulário.*

2. **USE-ME version:** *
   Marcar apenas uma oval.

   ◯ 1 (i.e. no validation rules)
   ◯ 2 (i.e. with validation rules)

3. **Modeling exercise:** *
   Marcar apenas uma oval.

   ◯ Lego
   ◯ Smart House

4. **Modeling activity:** *
Marcar apenas uma oval.

- Utility Specification
- Context Specification
- Goal Specification
- Evaluation Specification
- Interaction Specification
- Survey Specification
- Report Specification

5. **I think that I would like to use this system frequently.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

6. **I found the system unnecessarily complex.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

7. **I thought the system was easy to use.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

8. **I think I would need the support of a technical person to be able to use this system.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

9. **I found the various functions in this system were well integrated.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

10. **I thought there was too much inconsistency in this system.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree						Strongly agree

11. **I would imagine that most people would learn to use this system very quickly.** *

Marcar apenas uma oval.

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

12. **I found the system very cumbersome (i.e. difficult) to use.** *

Marcar apenas uma oval.

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

13. **I felt very confident using the system.** *

Marcar apenas uma oval.

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

14. **I needed to learn a lot of things before I could get going with this system.** *

Marcar apenas uma oval.

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

15. **Suggestions**

____

____

____

____

____

APPENDIX I

INSTALLATION GUIDE

In this appendix, we present a *step-by-step* installation guide.

I.1 System Requirements

1. Install **Java 1.8**;
2. Install **Eclipse Modeling Tools** for **Neon Release 3**;
3. Run Eclipse Neon;
4. Go to *Help* → *Install New Software*;
5. On the *Work with* form select *Neon*;
6. On the *Modeling* package select **EMF - Eclipse Modeling Framework SDK** and **Diagram Editor for Ecore (SDK)**;
7. Follow the instructions provided by Eclipse;
8. Restart your Eclipse after the installation;
9. Go to *Help* → *Eclipse Marketplace*;
10. Search for **Sirius** → *Install*;
11. Follow the instructions provided by Eclipse;
12. Restart your Eclipse after the installation;
13. Go to *Help* → *Eclipse Marketplace*;
14. Search for **Epsilon** → *Install*;

15. Follow the instructions provided by Eclipse;
16. Restart your Eclipse after the installation.

I.2 Download USE-ME and examples

1. Go to `https://github.com/akki55/useme/tree/master/language`;
2. Download USE-ME:
   - **pt.fct.unl.novalincs.useme.model**;
   - **pt.fct.unl.novalincs.useme.model.edit**;
   - **pt.fct.unl.novalincs.useme.model.editor**;
   - **pt.fct.unl.novalincs.useme.model.tests**;
   - **pt.fct.unl.novalincs.useme.sirius.design**;
3. Go to `https://github.com/akki55/useme/tree/master/examples`;
4. Download the examples:
   - **pt.fct.unl.novalincs.useme.example.Lego**;
   - **pt.fct.unl.novalincs.useme.example.SmartHouse**;
   - **pt.fct.unl.novalincs.useme.example.Visualino**.

I.3 Import USE-ME

1. On Eclipse workplace, go to *File* $\rightarrow$ *Import* $\rightarrow$ *Projects from Folder or Archive* $\rightarrow$ *Next* and import into your workspace the following files:
   - **pt.fct.unl.novalincs.useme.model**;
   - **pt.fct.unl.novalincs.useme.model.edit**;
   - **pt.fct.unl.novalincs.useme.model.editor**;
   - **pt.fct.unl.novalincs.useme.model.tests**;
2. After importing all the files, right click on **pt.fct.unl.novalincs.useme.model** $\rightarrow$ *Run As* $\rightarrow$ *Eclipse Application*.

I.4 Import USE-ME examples

1. On runtime environment, go to *File* $\rightarrow$ *Import* $\rightarrow$ *Projects from Folder or Archive* $\rightarrow$ *Next* and import into your runtime environment the following files:
   - **pt.fct.unl.novalincs.useme.sirius.design**;

- **pt.fct.unl.novalincs.useme.example.Lego**;
- **pt.fct.unl.novalincs.useme.example.SmartHouse**;
- **pt.fct.unl.novalincs.useme.example.Visualino**.

2. After importing all the files, check the file ***Name*.useme** for more details.

I.5 Create an USE-ME model

1. Go to *File* → *New* → *Modeling Project* → add the *Project name* → *Finish*;
2. Right click on the new Modeling Project → *New* → *Other* → *UseMe Model* → *Next* → fill the *File name* → *Finish*.

I.6 Validate an USE-ME model

1. Open the *.useme* file inside the Modeling Project;
2. Right Click on the root *platform:/resource/(Modeling Project name)/(UseMe model name))* → *Validate*;
3. Repeat the previous step every time you want to validate your model, or if you need guidance/suggestions. We recommend that you *Validate* the model every time you make changes.

Important note: the *Validate* action **does not save the file**, so you should save it every time you make important changes.