Computer Languages, Systems & Structures 51 (2018) 118–157

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Usability driven DSL development with USE-ME

Ankica Barišić*, Vasco Amaral, Miguel Goulão

NOVA-LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Campus de Caparica, 2829-516 Caparica, Portugal

ARTICLE INFO

Article history:
Received 24 March 2017
Revised 22 May 2017
Accepted 21 June 2017
Available online 6 July 2017

Keywords:
Domain-specific languages
Software language engineering
Experimental software engineering
Usability engineering
Quality of DSLs
Quality in use

ABSTRACT

The adoption of Domain-Specific Languages (DSLs) is regarded as an approach to reduce the accidental complexity of software systems development. The availability of sophisticated language workbenches facilitates the development of DSLs making them increasingly more popular. The adoption of DSLs at large comes at the risk that a poorly designed DSL can be too hard to adopt by its domain users. As such, Usability is one of the essential characteristics to mitigate this risk as it has an important impact on the productivity achieved by DSL users.

The current state of practice in Software Language Engineering (SLE) neglects the Usability of DSLs. A pertinent research question in SLE is how to engineer Usability into DSLs systematically. We argue that a timely systematic approach based on User Interface experimental evaluation techniques should be used to assess the impact of DSLs during their development process, while the cost of fixing the usability problems is relatively small, when compared to fixing them at the end of the development process.

For that purpose, we introduce a conceptual framework, called USE-ME, which supports the iterative incremental development process of DSLs concerning the issue of their Usability evaluation. We illustrate the feasibility of the approach on a case study of the development of a DSL meant for children to program robots.

1. Introduction

The increasing pace at which the software is being adopted in daily tasks, including those of users not necessarily proficient with computing, is pushing the need for rapid production of a growing number of complex software applications. The degree of specialisation in certain areas is pushing for the involvement of domain aspects in the software development process, as complex software configuration tasks. A *Domain-Specific Language (DSL)* is specialised in a particular application domain [1]. It can be defined as a user empowerment tool to increase productivity in software systems development [2,3]. It offers the expressiveness required to specify the software applications at a higher level of abstraction, after which they can be automatically deployed or even simulated, with notations more close to the end user. DSLs 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) [4]. Bridging this gap is expected to increase language users' productivity.

Practitioners often experience some practical difficulties when adopting DSLs [5]. During the language development, the importance of aligning the DSLs with the needs of their end users seems to be underestimated [6,7]. The necessity for

* Corresponding author.
E-mail addresses: a.barisic@campus.fct.unl.pt, barisic.ankica@gmail.com (A. Barišić), vma@fct.unl.pt (V. Amaral), mgoul@fct.unl.pt (M. Goulão).

http://dx.doi.org/10.1016/j.cl.2017.06.005
1477-8424/

assessing the impact of introducing a DSL in a domain workflow has been discussed in the literature, often with a focus on the business value that DSL can bring [2]. This business value often translates into productivity gains resulting from the extent to which the domain users can use the DSL in practice [4].

Although building and adopting DSLs may seem intuitive, we need to have means to evaluate their impact. The measure of success has to be determined by assessing the impact of using DSL, in a realistic context of use, by its target domain users [8]. Investment into this assessment, commonly called Usability evaluation, is justified by a reduction of development costs and increased revenues for other software products, brought by an improved effectiveness and efficiency by their end users [9,10]. We expect a similar effect by introducing this practice into DSL development.

The software industry does not often report investment on the assessment of DSLs [11,12]. Most of the reported DSLs evaluations are performed only at final stages of a development cycle, when changes in the DSL have a significant impact on the budget. The lack of systematic approaches, guidelines and comprehensive set of tools may explain this shortcoming in the current state of practice. We argue that this situation is due to the perceived high costs of DSL evaluation, which lacks a consistent and computer-aided integration of two different and demanding complementary software processes: DSL development and usability engineering.

Software Language Engineering (SLE) is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages. Although the phases of DSL life cycle are systematically defined [1], we claim that this process lacks one crucial step, namely language evaluation [13]. Existing *Experimental Software Engineering (ESE)* techniques [14] combined with *Usability Engineering* techniques [15] can be adopted to support the DSLs' evaluations. Our goal is to *promote quality in use of DSLs by building up a conceptual framework that supports their development process by leveraging usability as a first-class concern*.

We can engineer DSLs to become more usable with a combination of both proactive and reactive approaches. Current proactive approaches that can be used to improve DSL usability, such as guidelines of visual notation [16], usability heuristics [17], cognitive dimensions of notations framework [18,19], or even quality assessment framework for DSLs [20]. Reactive approaches are necessary as well [11]; DSLs should be tested experimentally with humans using systematic techniques to confirm the impact of design decisions in a real context of use. We highlighted the experimental approach in [21], based on four DSL evaluation experiments that are examples of best practices [22–25]. Recently, we can find more examples of performing this kind of assessments in practice (e.g. [26–30]).

Usability concerns need to be addressed from the early stages of the DSL life cycle so that practitioners can perform timely evaluations [31]. Rather than designing the complete DSL before the implementation, abstractions should be evaluated iteratively and incrementally, in the context of a development cycle [8]. Building the systematic iterative usability evaluation approach is supposed to mitigate the risk of producing the inappropriate solutions that often cannot be reused. This work is expected to enhance the community's awareness to the relevance of DSLs' usability assessments to bridge the gap between the domain users and abstractions provided by language engineers.

The main contribution of this paper is a conceptual approach, which highlights the complexity of relevant knowledge that should be traced in the DSL development process to support the usability evaluations and experimental design. In Section 2 the DSL development cycle and the notion of usability engineering is introduced. In Section 3 a research process and previous work are described. In Section 4 we present a USE-ME conceptual framework, namely its domain concepts, meta-models and activity specifications. Section 5 provides a proof of concept for USE-ME approach by illustrating the conducted usability trials with the Visualino language using an early framework prototype. In Section 6 we discuss the applicability of our approach during a DSL development. Section 7 discuss related work. Finally, we conclude in Section 8.

2. Background

The immersion of computer technology in a wide range of domains leads to a situation where the users' needs become increasingly demanding and complex. Consequently, the development of successful software systems becomes increasingly more complicated. Software engineers need to cope with the growing of both essential and accidental complexity [32]. They have to provide solutions that solve a class of crucial problems in a given domain, which is sometimes very difficult to learn, such as the rules and technical jargon found in knowledge areas such as Physics, Finance, Medicine, etc. Also, it is necessary to deal with the accidental complexity of the used technology, e.g., the use of low-level abstraction programming languages while integrating a wide plethora of different tools and libraries.The adoption of DSLs is regarded as an approach to reduce the accidental complexity of software systems development [33–35].

2.1. Domain-Specific Language (DSL)

A *DSL* is a language that supports solutions to essential problems from a given domain (e.g. Physics Computing, Financial Domain, Health-care, Control Systems, and Gaming). They are intended to raise the level of abstraction closer to users' domain understanding. Opposing to a GPL, such as Java or C++, that is meant to be applicable across domains, a DSL offers the end user the possibility to express his needs in terms of the domain of the problem instead of in terms of the computational solution [36]. DSLs provide a notation tailored towards an application domain as they are based on models of relevant concepts and features of the domain [1]. As DSLs are used to describe and generate members of a family of systems or products in the application domain, they give the expressive power to model the required family members more

easily. DSLs are claimed to match users' mental model of the problem domain by constraining the user to the given problem [37]. Finally, DSLs simplify the development of applications in specialised domains at the cost of their generality.

The use of the *Model-Driven Development (MDD)* techniques and tools are seen as a viable approach for dealing with accidental complexity [34]. MDD is grounded on the notion of providing explicit Models, seen as 'first class artefacts' in the process, that are further transformed into other lower level, more detailed, models. These transformations are also considered as development artefacts and can be explicitly modelled by using transformation models. This approach has the special impact in dealing with the complexity of large-scale problems while enabling rapid prototyping, simulation, validation and verification techniques [2,38].

In general, *'A model is a representation of something, constructed and used for a particular purpose'* [39]. Kühne defines this concept more appropriate to our context as *'A model is an abstraction of a (real or language-based) system allowing predictions or conferences to be made'* [40]. Modellers build models to represent something. But a model has no meaning by itself. The information in the model can be understood if the model is combined with an interpretation. Extracting the correct meaning from the model can only be achieved if a common understanding of the concepts between modeller and the interpreter (who will typically use the model) is established. This common understanding leads us to a language. In practice, DSLs can be used to represent domain-specific models in MDD approaches. Each different model adopted by an MDD approach can be seen as a DSL that addresses the modelling of abstractions at a specific stage of software development. We have several guides that discuss how to implement a DSL based on MDD approach [36,41,42].

2.1.1. DSL stakeholders

A *Language Engineer* is a professional who is skilled in the application of the engineering discipline to the creation of software languages. This professional manages the implementation priorities, designs the software language and is responsible for making the language functional at the system level. In general, Language Engineers are involved in the language specification, implementation, and evaluation, as well as in providing templates and scripts [43].

A *DSL User* or *Domain User* is any person who uses software languages to create applications (e.g. application developers) [43]. The possible user base of the models can easily be broader as domain-specific modelling allows application users to be better involved in the application development process. In that case, customers, other than typical application developers, can read, accept and in some cases change application specifications, being directly involved in the application development process. A Domain User can work with models which apply concepts directly related to specific characteristics of the configuration, such as specifying deployment of software units to hardware, or describing high-availability settings for uninterrupted services with redundancy for various fault recovery scenarios [2].

A *Domain Expert* is a person involved in the language development process, also sometimes known as a knowledge engineer. In the case of domain-specific modelling, they do not need to have the software development background, but they can specify the application for code generation. A Domain Expert specifies models for concept prototyping or concept demonstration, and Language Engineers can proceed from these models. They are responsible for managing system goals and iterations. In contrast with Domain Users, they should have domain knowledge that includes areas of all target model applications.

DSLs are usually built by Language Engineers in cooperation with Domain Experts [34]. In practice, Domain Users will use the DSL. These Domain Users are the real target audience for the DSL. The Domain Experts and Language Engineers can play the Domain User role, but they are, often, just a small subset of target end-users population. Although Domain Users are familiar with the domain, they are not necessarily as experienced as the Domain Experts. They may also lack the experience of Language Engineers in using languages. So, it may turn out that the language is valid by construction for Domain Experts and Language Engineers, but not necessarily to other Domain Users. Neglecting Domain Users in the development process may lead to a DSL they are not really able to work with.

2.1.2. DSL life-cycle

Several steps are detailed by Thibault [44], Völter et al. [34], Mernik et al. [1], Visser [45], Strembeck and Zdun [42], to develop a DSL, contributing to the formal definition of the DSL life-cycle. The following are listed as DSL development phases: decision, analysis, design, implementation and deployment. Mernik et al. [1] provided a set of patterns that describe common situations that potential developers may find themselves in, and were already tackled successfully by previous DSL development.

The first phase is the *Decision* that corresponds to the 'when' part of the DSL development, while the remaining phases to the 'how' part. Its goal is to identify the need for a DSL to the domain and its validity, which includes justifying that the effort to invest in its creation can be compensated. To make the decision, the stakeholders (including Domain Experts and Language Engineers) need to discuss the requirements of the domain following DSL development patterns.

The following phase is the *Analysis*. Its goal is to define the domain model with support to the DSL. The analysis has to take into account the particularities of the domain that will be explored, such as the terms and expressions intrinsic to a problem. In this phase Domain Experts helps Language Engineers to define the description of domain concepts, the feature models, the functional and technical requirements, and the goal model. The Analysis phase primarily produces a domain model, representing common and varying properties of the system within the domain [46]. The domain model will be used to assist with the creation of configurable architectures and components.

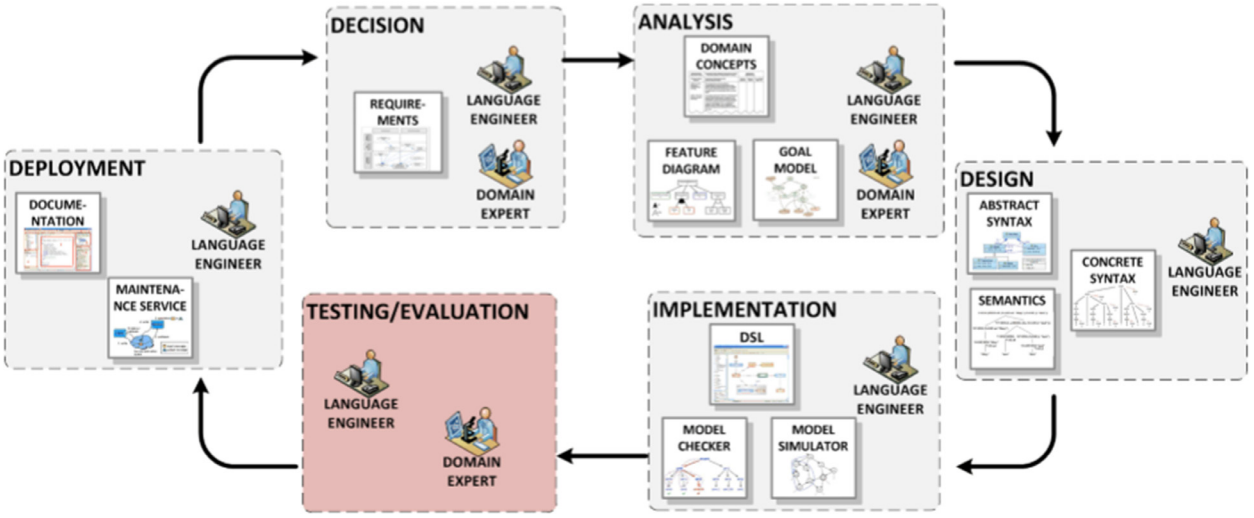

Fig. 1. DSL life-cycle.

The next phase is *Design*, where Language Engineers formalise a language abstract syntax (i.e. meta-model) and define the representations for the model elements and production/composition rules. Additionally, the semantics of the language is defined. The design approaches of DSL design can be characterised along two orthogonal dimensions: the relationship between the DSL and existing languages, and the formal nature of the design description [1]. A DSL can be designed from scratch or based on an already existing language.

The fourth phase is the actual *Implementation*, which includes the integration of DSL artefacts with the infrastructure, as well as the implementation of the necessary DSL to platform transformations. A DSL can be implemented by different approaches (e.g., interpreter, compiler, preprocessing, embedding, extensible compiler/interpreter, COTS, hybrid [1]), each having its own merits [47]. In this phase the developers produce the model checkers and simulators, which will help the modeler to validate the specified models. Finally, the DSL is delivered with its documentation in the *Deployment* phase. While Domain Experts themselves can understand, validate and modify the software by adapting the models expressed in DSLs, more substantial changes may involve altering a DSL implementation. Therefore, the DSL should have a migration strategy, as any other software product.

We argue that this process lacks an important step: *Evaluation* [8,48], just before the deployment (see Fig. 1), that should include the verification (testing if the right functionality is provided by the DSL) as well as its validation (testing if this DSLs is right for its users). According to the current practice, the focus of evaluation is only on the language engineering, and not on its usability, which is a clear lack of validation which involves the end-users [11]. Current verifications are supported by model checkers and simulators. As the DSL promote the modifications of models which are claimed to be easier to produce and their impact to be easier understood, supporting language should be systematically evaluated by real user actions. This phase is expected to help mitigate a most pervasive problem of software engineering, i.e. software comprehension [49], that addresses the problem of associating human oriented concepts with their counterpart solution domain concepts (e.g. computational terms).

Visser [45] recommends the inductive approach which, in opposition to designing the complete DSL before implementation, incrementally introduces abstractions that allow capturing a set of common programming patterns in software development for a particular domain. Visser also states that developing the DSL in iterations can mitigate the risk of failure. Instead of a big project that produces a functional DSL in the end, an iterative process produces useful DSL early on. The availability of sophisticated language workbenches facilitates the development of DSLs making them increasingly more popular. This comes at the risk that a badly designed DSL can bring more harm than benefits and decrease productivity when compared a GPL alternative. In particular, a poorly designed DSL can be too hard to adopt by its domain users. As such, to evaluate usability is one of the key characteristics to mitigate this risk as it has an important impact on the achieved productivity of DSL users.

2.2. Approach for evaluating usability of DSL

While thinking about programming languages, DSLs are conceived as communication interfaces between human and computers. Therefore, if we take into account the main purpose of Human–Computer Interaction (HCI), we can conclude that to evaluate DSLs has several similarities to assessing regular User Interfaces (UIs) [22]. We argue that any UI is a realisation of a language, where a language is considered as a theoretical object (i.e. model) that describes the allowed terms and how to compose them into the sentences involved in a particular HCI.

Usability engineering is a field that is generally concerned with HCI and specifically with devising UIs that have high usability. It provides structured methods for achieving efficiency and elegance in interface design [15]. Empirical (i.e. experimental) evaluation studies of UIs with real users is a crucial phase of the usability engineering life-cycle [50]. A relevant set of quantitative and qualitative measurements must be inferred and combined together to lead to a useful assessment of the several dimensions that define software Quality in Use, often referred to as Usability [51].

2.2.1. Usability i.e. quality in use

The notion of *Usability* is used in many different contexts and its definitions developed progressively. Usability is defined by Shackel and Richardson [52] as the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfill technically specified range of tasks, within the specified range of environmental scenarios. Shortly, Usability is *'the capability to be used by humans easily and effectively, where 'easily = to a specified level of subjective assessment'; 'effectively = to a specified level of (human) performance'*.

Usability is the quality characteristic that measures the ease of use of any software system that interacts directly with an end user. It is a subjective non-functional requirement that can only be measured indirectly by the extent to which it satisfies its corresponding needs based on the the users' cognitive capacity. It focuses on features of the HCI. Usability is the result of the achieved level of quality in use of a software system i.e. a user's view of quality. It is dependent on achieving the necessary external and internal quality that is influenced by the achievement of different quality attributes dependent on a context of use. Tests of language usability are based on measurements of the users' experiences with it.

International Organisation for Standardization (ISO) firstly defines Usability as 'the effectiveness, efficiency, and satisfaction with which specified users achieve specified goals in particular environments' (ISO 9241–11 [53]). Later on, the notion of Usability is integrated into the software quality framework under the term Quality in Use (ISO 9126 [54]). Finally, Usability is defined as *'degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use'*. Usability can either be specified or measured as a product quality characteristic in terms of its sub-characteristics, or specified or measured directly by measures that are a subset of quality in use (ISO/IEC 25010 [51]).

There is an increasing awareness to the quality in use of languages, fostered by the competition of language providers. Better usability is a competitive advantage, although evaluating it remains challenging. While evaluating competing languages, it is hard to:

1. interpret the existing metrics in a fair, unbiased way;
2. provide relevant design changes; and
3. assure that the scope of their evaluation is preserved to target user groups.

Software *Expert Evaluators* typically implement complex experimental evaluation studies. Their expertise is essential to design the evaluation sessions properly and to gather, interpret, and synthesise significant results. Although it is desirable to have an Expert Evaluator within in the teams, it is not always possible. This calls for the need of automated tools that support these experts, as well as other DSL stakeholders.

2.2.2. Usability design – how and when?

Usability has two complementary roles in design: as an attribute that must be designed into the product, and as the highest-level quality objective, which should be the overall purpose of design [55]. Two important issues are how and when to assess DSL usability.

Concerning the *how*, we can think of DSLs as communication interfaces between their users and a computing platform, making DSL usability evaluation a particular case of evaluating UIs [13]. This implies identifying the key quality criteria from the perspective of the most relevant stakeholders, to instantiate an evaluation model for that particular DSL [20,56]. These criteria are the evaluation goals, for which a set of relevant quantitative and qualitative measurements must be identified and collected. We borrow from UI evaluation several practices, including obtaining these measurements by observing or interviewing, users [57]. In general, it is crucial that the evaluation of HCIs includes real users [50], for the sake of its validity. In the context of DSLs, the 'real users' are the Domain Users (see Section 2.1.1).

Interactive usability investigation methods apply contextual inquiry and formative usability testing as crucial for successful usability design [58]. *Usability Testing* (UT) includes task analysis that studies the way people perform tasks with existing systems. By a high-level abstraction study of cognitive processes, we could identify what are the individual tasks that the language is expected to support. The cognitive activities that should be analysed are:

1. Composition of the syntax required to perform a function.
2. Comprehension of function syntax composed by someone else.
3. Debugging of syntax or semantics written by ourselves or others.
4. Modification of a function written by ourselves or others.

Experimenters in human factors developed a list of tasks to capture these particular aspects [59]: Sentence Writing, Sentence Reading, Sentence Interpretation, Comprehension, Memorisation, and Problem solving. We can evaluate these tasks using tests such as Final exams, Immediate Comprehension, Reviews, Productivity, Retention, and Re-learning. Performing

exhaustive evaluation of different tasks in the language usage is interesting, but would be too expensive. Therefore, the evaluation usually concerns only the most critical activities.

More general method in accessing usability are *Heuristic evaluation* (HE) [60], but are often regarded as not being capable to encompass all usability attributes. Recently, new evaluation method called *Domain Specific Inspection (DSI)* is developed using traditional evaluations (HE and UT) in novel ways [58]. The lack of a methodological framework that can be used to generate a domain-specific evaluation method, which can then be used to improve the usability assessment process for a product in any chosen domain, was proposed by Alroobaea [61]. It is shown that this adaptive framework is able to build a formative and summative evaluation method that provides optimal results with regard to the identification of comprehensive usability problem areas and relevant usability evaluation method metrics, with minimum input in terms of the cost and time usually spent on employing usability evaluation method.

Concerning the **when**, we argued that we should adopt a systematic approach to obtaining a timely frequent usability feedback, while developing the DSL, to better monitor its impact [8]. This implies the integration of two different and demanding complementary processes: language development and usability evaluation. Language Engineers should be aware of usability concerns during development, to minimise rework caused by unforeseen usability shortcomings. In turn, Expert Evaluators should have enough understanding of the domain-specific models involved in software language development to be able to properly design the evaluation sessions, gather, interpret, and synthesise meaningful results that can support the DSL improvements in a timely way.

The timely frequent usability testing is in line with agile practices, making them a good fit for this combined DSL building (i.e. software development) and evaluation process (i.e. usability design) [62]. Agile development process breaks products into small increments, and each iteration should fit in short time-boxes that typically not last more than a month [63]. This iterative, incremental development process is also in alignment with Visser's inductive DSL development suggestion [45] (see Section 2.1.2). Agile practices provide a method for gathering user feedback by conducting a focus group after a feature was implemented, and ask for users' opinions. However, this approach is insufficient in supporting usability design which is largely based on observing user behaviour by utilisation of usability investigation before product implementation [1].

Therefore, it is necessary to apply a **User-Centred Design (UCD)** that is comprised of end user involvement in the development of software products at different points of the life-cycle. UCD include techniques that support usability design such as ethnographic research, participatory design, focus group research, surveys, walk through, preliminary prototyping, expert or heuristic evaluation, usability testing, as well as follow-up studies [57,65,66]. UCD can be characterised as a multistage problem-solving process. It foresees how users are likely to use a product and tests the validity of their assumptions concerning user behaviour in real world tests with actual users. Such testing is necessary as it is often tough for the developers of a product to understand what is intuitive for an end user of their design experiences, and what each user's learning curve may look like. The essential activities required to implement UCD are described in ISO 13407 [67].

2.2.3. Contextual aspects of DSL usability evaluation

In this particular research, we only consider languages that are used as communication interfaces between humans and computers (i.e. UIs). Therefore human–human languages, e.g. natural languages, and machine-machine languages, e.g. communication protocols, are not relevant for the work described in this paper.

Semiotics, the study of the structure and meaning of languages, is a part of linguistics that studies the dependencies and influences among syntax, semantics, and pragmatics. The *syntax* of a language defines what signs we can use in that language, and how we can compose those signs to form sentences. The *semantics* of a language defines the conceptual meaning of the sentences in that language by stating how they can be logically interpreted. Finally, the *pragmatics* sets the context of use from which the sentences of that language can have some logical meaning [68].

The **Context of Use** i.e. *'the users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used'* [54] is one of the characteristics that we must be considered while evaluating DSLs usability. This is to pragmatically distinguish between DSLs scope of the assessments. Different DSLs, especially the ones that are developed for various domains, have a different Context of Use. We can infer that the users of those DSLs, most likely, will have different knowledge sets, each one with a minimum amount of ontological concepts required to be able to use each language.

We need a rigorous and collaborative usability evaluation procedure for DSL (both during and after the particular DSL development) that supports validation of DSL sentences (called instance models) in a correct Context of Use. According to the context of communication, these sentences can have different interpretations. If the context is not clear, interpretations can be ambiguous. Notice that, in this pragmatic perspective, languages that do not even share the same base syntax of those same sentences may share the same domain concepts, i.e. the intersection of their domain concepts is not empty for a given non-empty intersection of contexts of use. If the intersection of their contexts of use is empty, then they do not share any of the identified domain concepts.

3. The USE-ME research process

The design science methodology [69–71] fosters the creation of artefacts that are driven to problem-solving projects. Wieringa [72] see design science projects as a set of nested regulative cycles that solve practical (i.e. engineering) and a

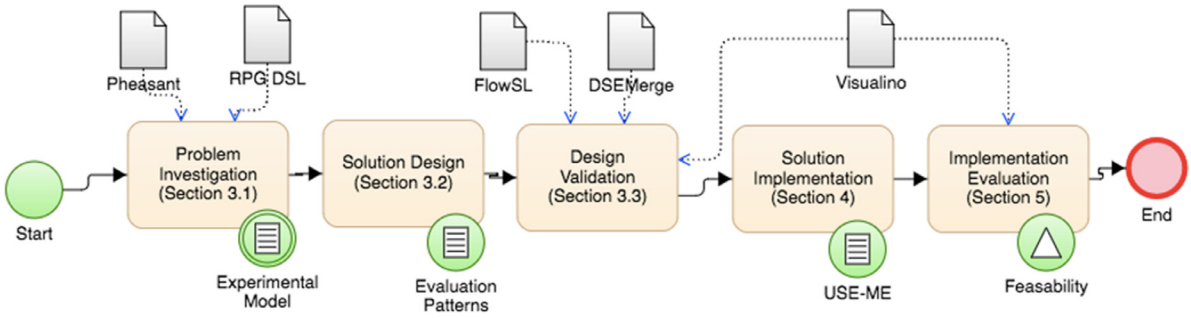

Fig. 2. Research process overview.

knowledge (i.e. research) problems that are decomposed in sub-problems. Knowledge problem is defined as a 'difference between the way the world is experienced by the stakeholders and the way they would like it to be'. A research problem is the 'difference between the current knowledge of stakeholders about the world and what they would like to know'.

The motivation of this work is to *provide a systematic methodological approach to evaluate the usability of domain-specific languages during its development for application domains that target large end-user groups* (see Section 1). Following this above-mentioned methodology, we characterise our problem as an engineering (i.e. practical) problem and argue that the goal of any DSL that is meant to be used by humans is to improve the productivity of its users. To achieve this goal, it is mandatory to increase the usability of the language (Section 2).

We address following research questions:

Q1: *How can we promote usability concerns since an early stage of development of the DSL?*
Q2: *How are we able to trace and validate language goals and usability evaluation coverage?*

The regulative cycle follows the five tasks: problem investigation, solution design, design validation, solution implementation and implementation evaluation (see Fig. 2). The first step is to analyse the problem in detail during the 'Problem investigation', namely to identify the stakeholders, their goals, problematic phenomena, causes, impacts and solution criteria. The following step is to analyse available solutions and design new ones during 'Solution design'. The next step is to perform 'Design validation' during which we validate solution properties and satisfaction of criteria. We observe goals, trade-offs and sensitivity of approach. The details about the first three phases were subject of previous work [8,21,56], and are briefly discussed in the following sections. Further, we discussed the 'Solution implementation' in Section 4, by presenting the concepts and activities which are mandatory for the application of the approach. Finally, 'Implementation Evaluation' is discussed in Section 5.2.

3.1. Problem investigation

The first step was to analyse the problem in detail. We started by focusing our attention onto stakeholders who have the need for a DSL, i.e. Domain Users, and specify their goals regarding a development of the DSL in Section 2. To understand better the problem, they are facing and their causes, we have analysed the existing DSL development techniques and artefacts produced during a regulative development cycle. The development of different DSLs followed in the context of the FCT/UNL faculty's DSL courses and DSL summer school.[1] These activities gave us a practical experience in using different modelling workbenches (EMF, TextEdit, Eugenia, MetaEdit, Kaos, ATL). We reported our experience during the development of a Role-Playing Game (RPG) DSL following the regulative development cycle [73]. Besides that, we have performed an initial evaluation on the DSL named Pheasant for high energy physics domain [22]. This study helped us understand the impact of the problem in the context where the DSL is meant to be used by non-programmers in a sensitive context. Finally, we analysed existing evaluation examples of DSLs and specified a generic experimental model [21]. The experimental model helped us understand the criteria for stakeholders to consider the problem as solved.

3.1.1. RPG DSL

In this case study, an RPG DSL for product lines was developed, which was completely built using MDD software development techniques [73]. We have shown several benefits of applying MDD regarding prototyping of cross-platform games, and their evaluation by means of static and dynamic verification techniques of the game's logic properties.

In the Domain Analysis phase, we worked on two different levels: at the problem level (i.e., expressing the concepts and logics of RPG Games), and at the solution level (i.e., how those concepts can be realised in a computational platform). At the level of the problem of RPGs' design, we tried to express and define what would be the common characteristics of all RPG games, regardless of what are the requirements of their implementation on an underlying computation infrastructure. At the

[1] dsm-tp.org (accessed May 4, 2017).

solution (computational) level, we had to choose platforms to deploy the generated games and to analyse them. We chose a game developing platform and built our framework on top of it. The criteria to select the target framework from the existing game engines were: fast development; provided abstraction level relatively to system calls and hardware dependencies (e.g., graphical primitives, input modalities, etc.); and, need to previous knowledge in the area of game engines.

We chose Corona SDK 2 from a set of three promising candidate frameworks because it supports the possibility to compile the game for different mobile platforms. The game is programmed using the Lua language: a scripting language, which is preferred for rapid development. The creation of an API over the framework allowed a model to model transformation that was easily produced between RPG and Framework meta-models, which consists mostly of 1 to 1 relationships. This strategy of bottom-up modelling in the framework allowed the focusing on the RPG entities. The mapping of the RPG meta-model to APN is too complex to be performed in just one step. Therefore we introduced an intermediate meta-model RPG, that allowed a simpler mapping between both.

3.1.2. Pheasant

The DSL on this case study published in [22], called Pheasant (Physicist's Easy Analysis Tool), is assessed in contrast with a pre-existing GPL baseline. The comparison combines quantitative and qualitative data, collected with users from a real-world setting, and was performed at the end of the development process. The goal of Pheasant's development was to improve the efficiency, reduce the error rate and have a less steep learning curve than the existing GPL. The target users of the Pheasant language are specialists in High Energy Physics, with varying experience in software development. The evaluation was performed according to the ISO 9241-11 usability definition, which is an essential part of the achieved Quality in Use.

The assessment includes Physicists with programming experience with two profiles: the ones with no experience with the previous framework used and the experienced ones. The goal was to analyse the performance of Pheasant programmers compared to the baseline alternative with respect to the efficiency, effectiveness and confidence in defining queries in Pheasant. The assessment was done from the point of view of a researcher trying to assess the Pheasant DSL, in the context of a case study on selected queries.

Introducing one language to the whole group of participants and only afterwards the other language would bias the evaluation, as the knowledge acquired while learning the first language could be partially reused while using the second language. To mitigate this threat to the validity of the results we had to split the group in two. This way, it reduced the influence of the first language while presenting the second. Mixing the two groups might lead to new variables in the evaluation that are hard to track. Therefore, it was necessary to organise four sessions, with each group taking part in two sessions (one for each language). Following the scientific method, the participants' performance in the query writing was evaluated. Every participant had four queries, specified in English, to be rewritten in the previously learned language.

Using Pheasant, the users increased effectiveness during their query specification. The DSL was less error-prone than the alternative; it is allowed non-programmers to define their queries correctly. The evaluation also showed a considerable speedup in the query definition by all the groups of users that were using Pheasant. In general, the feedback obtained from the users was that it is more comfortable to use Pheasant with the alternative. The preliminary pilot study was fundamental to ensure that the subject's time was well spent. This work's contribution illustrates how an experimental process (Section 3.1.3) can be used in the context of a DSL evaluation, with respect to its impact on Quality in Use characteristics (Section 2.2.1), namely effectiveness and efficiency. The valuable feedback from users concerning the tool support for the language, as well as their fears concerning language expressiveness, support the idea of an iterative evaluation process where improvements to the language and its tool support are to be performed and then assessed in a new round of evaluation.

3.1.3. Experimental model

In the context of Software Engineering, a controlled experiment can be defined as 'a randomised experiment or quasi-experiment in which individuals or teams (the experimental units) conduct one or more Software Engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages or tools (the treatments)' [74]. For our purposes, this can be instantiated with developers typically conducting software construction, or evolution tasks, for the sake of comparing different languages including the DSL under evaluation and any existing baseline alternatives to that DSL. This complex challenge was covered by a general model for DSL experimental evaluation presented in [21], that served as a set of instantiations that were proof of concept of the proposed model.

When conducting language evaluation experiments, one should clearly define the experiments' objectives. Building upon Basili's earlier work [75], Wohlin et al. proposed a framework to guide the experiment definition [76]. The framework is to be mapped into a template with the following: the object of study under analysis, the purpose of the experiment, its quality focus, the perspective from which the experiment results are being interpreted, and the context under which the experiment is run.

While the experiment definition expresses something about why a particular language evaluation is performed, the experiment planning expresses something about how it can be performed. Before starting the experiment, decisions have to be made concerning the context of the experiment, the hypotheses under study, the set of independent and dependent

variables that will be used to evaluate the hypotheses, the selection of subjects participating in the experiment, the experiment's design and instrumentation, and also an evaluation of the experiment's validity. Only after all these details are sorted out should the experiment be performed. The outcome of planning is the experimental language evaluation design, which should encompass enough details to be independently replicable.

3.2. Solution design

After the problem has been characterised, it was necessary to research the domain to justify that none of the existing solutions solve the problem [13]. As no satisfactory solution has been found, there was room to propose a new one. As a consequence, a new solution is designed with the aim to solve the problem [8]. The usability of a language needs to be evaluated through controlled experiments involving the language's end users. To be able to identify potential quality problems that will lead to user interaction and experience problems, a suitable approach is to apply UCD practices during design and development of the language. Nevertheless, it is found hard to control budget and plan time and responsibilities accordingly. An incremental, iterative process should be applied to enable tracking of design changes and validation of usability metrics.

3.2.1. Patterns for DSL evaluation

In [56] we proposed an iterative user-centred approach for DSL evaluation that is to be used during DSLs' life-cycle. This proposal illustrates correspondences between the DSL development process (Section 2.1.2) and the agile UCD approach (Section 2.2.2). The goal of our approach is to support DSL usability evaluations since the early stages of its development to prevent user-interaction mistakes, hence achieving a usable DSL by construction. The initial vision is detailed by a pattern language for evaluating the usability of DSLs that places the intended users of a language as the focal aspect of its design and conception by establishing formal correspondences for all stages between the DSL development process and the usability evaluation process. This iterative approach allows us to track the usability requirements and impact of recommendations, with a well-prepared evaluation process, allowing us to control the budget and the scope of the language's evaluation. This set of practices reflecting high-level guidelines are:

(i) Agile development process including patterns devoted to project management and engineering of a DSL. Through organisation and planning of language development, this practice enables the control of evaluation activities, their scope (i.e. the context of use), the budget and tracking the success of the DSL.
(ii) Iterative UCD is providing patterns for the engagement of the Expert Evaluator into the development process, with the intention of collecting relevant information concerning the Language Engineer perception of the problem solution and Domain Users' interpretation. Depending on the cognitive model instances to be evaluated, the DSL implementation technique or usability investigation technique, each design vary a lot.
(iii) Experimental evaluation design supporting the definition of experiment execution (e.g. hypothesis, tests, metrics, samples and statements) that were given by evaluation model [21]. This model supports the formal execution specification for any usability investigation assessment that includes involvement of the Domain Users.

To perform a DSL evaluation, a first step is to understand and specify its context of use. This activity is complementary to the domain analysis phase of DSL's construction. Further, a set of evaluation goals can be identified during the requirements' elicitation process. Interviews or questionnaires with the DSL's intended users can be designed to capture information about the working environment and by the baseline approach to solving problems. During DSL Design, it is necessary to validate if the design decisions conform to the context of use (e.g. chosen technical environment is integrable with the users' environment, or that the designed DSL constructs are understandable). Unclear assumptions can be improved through different iterations, but already with the first designs, the Expert Evaluator can define questions that could answer evaluation goals. During the Implementation, the evaluation model can be specified. The stakeholders that should be involved in the evaluation could already be profiled and prepared for the execution. In certain cases, just use of proactive approach can be sufficient to evaluate the iteration objectives. Finally, in the Evaluation phase the Expert Evaluator executes experiments and analyses the results, while the Language Engineers may still perform tests or prepare the product for Deployment.

The proposed iterative UCD evaluation approach can be merged into the development cycles of already existing and evolving languages. It enables us to intervene at any point of the DSL development. Very often the developers only become concerned with Usability issues in later phases of the DSL life-cycle.

3.3. Design validation

Once this design is completed, it is necessary to validate the solution (Design Validation) before its realisation. For that matter, the solution properties are assessed according to the criteria defined in the problem investigation, characterising the context of the target application and the coverage of the solution. If the solution has the desired effect for stakeholders in their context, the solution can be finally implemented. For that purpose, we have applied the approach in the two industrial cases of DSL development, namely FlowSL and Visualino. FlowSL served an instantiation of usability evaluation into DSL development guided by agile management during three iteration cycles [77]. The second one, Visualino, is an ongoing case study that we are performing, currently in the third iteration of the language's development and evaluation. This case study is a good representation of the DSL where the usability evaluation is mandatory as the users are children, and the programed

behaviour is expected to run on an Arduino robot. Finally, we applied our approach in the case of DSE Merge language [78], which is to be used by the programmers. This experience showed how the end users who are programmers could also benefit from the usability approach. Also, in this familiar environment was even easier to set up a more complex experiment and explore several possibilities for reusing it a virtual evaluation environment. However, we shortly report on this experiences in the following sections.

3.3.1. FlowSL

We applied action research to the development of a DSL, named FlowSL, designed to support managers when specifying and controlling the business processes supporting humanitarian campaigns [77]. FlowSL is targeted to non-programmers. Their ability to use this language was identified as one of the highest concerns, so discovering usability issues in early development iterations facilitated the achievement of an acceptable usability, while tracking the design decisions and their impact. Usability has two complementary roles in design: as an attribute that must be designed into the product, and as the highest level quality objective which should be the overall objective of design.

FlowSL is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organisation and is integrated into Movercado (MVC).[2] This project consists of a mobile-based messaging platform at the core of an ecosystem that enables real-time and a more efficient impact, by facilitating interactions among beneficiaries, health workers and facilities, e-money and mobile operators. A first version of the system (MVC1) was developed as a proof-of-concept to validate the key underlying principles. The second version of the system (MVC2) was developed in the form of a platform easily customisable by managers and extensible by developers of the organisation's team. An important goal was to develop a language, FlowSL, to empower the Campaign Managers to define new kinds of campaign flows taking advantage of their domain knowledge.

To balance the development effort with effective reusability (e.g. while envisioning new marketing solutions), MVC2 was developed in a fast-paced way, iteratively, along six two-weeks sprints, following an agile development process based on Scrum[3] and best practices of evolving reusable software systems. In the process of development, the Domain Experts were part of the Product Owners team, while the Language Engineers were part of the Scrum Team. The DSL evaluation process was guided by the FlowSL development stages, as different effort was estimated in each sprint for its development. The problem analysis was performed by mutual interaction and brainstorming between Domain Experts and Language Engineers in each sprint planning. We had the role of observing and guiding the analysis outputs, while preparing the evaluation plan, without being directly involved in the language specification.

In this case study, we integrated top-down usability engineering practices into a bottom-up agile development of a DSL from its beginning. While playing the role of Evaluation Expert that is expected to be introduced into DSL development, we experienced that small iterations involving other project stakeholders then Language Engineers, namely Domain Experts, Product Owners and End Users, can help us to clarify the meaning and the definition of the relevant language concepts. This enables an early identification of possible language usability shortcomings and helps to reshape the DSL accordingly. Early evaluations can be executed with a relatively low-cost thanks to model-driven tools (i.e. language workbenches) that support the production of rapid prototypes and presenting the idea. These evaluations support well-informed trade-offs among the strategy and design of the DSL under development, and its technical implementation, by improving communication. Besides, they improve the traceability of decisions, and of the solution progress. These iterations also help to capture and clarify contractual details of the most relevant language aspects that need to be considered during DSL development and are an essential element to improve the End Users experience while working with FlowSL.

3.3.2. Visualino

Visualino is a visual DSL with the objective to support children while programming a low-cost Arduino robots.[4] It is developed iterative reusing user-centred design methods. Programming technologies of rover robots for children still are excluding the children as target audience as it is hard for them to program in a textual language with a complex syntax full of technical concepts. To design an appropriate DSL for children is far from being a trivial task. There is no unique profile, and several factors related mostly to age, like the maturity level, that can influence widely in the design.

A visual programming language allows the user to implement programs, through the manipulation of visual elements or objects. This manipulation is deduced in a visual way, allowing the user to understand programming mechanisms quickly, increasing their accessibility to new systems. Users with no programming background may implement programs in a simpler form. Visual languages introduce programming concepts to children, while robots perform the developed code in the real world. Children have the opportunity to observe their own developed program, running on a physical robot.

The focus of the evaluated work was to build a dedicated DSL that removes the programming details, to control a rover robot, and at the same time allows children to easily get acquainted with it and preserve the possibility to program complex behaviour. The development of this language, named Visualino, counted with a robot consisting of common components and functions in the robotic area. The work was developed under the collaboration between the group ASE NOVA/LINCS and Artica,[5] a company that specialises in the development of robotic and audio-visual solutions.

[2] https://movercado.wordpress.com/ (accessed May 4, 2017).
[3] http://www.scrum.org/ (accessed May 4, 2017).
[4] http://www.arduino.cc/ (accessed May 4, 2017).
[5] http://artica.cc/ (accessed May 4, 2017).

The chosen meta-modelling workbench for the DSL implementation was Eugenia for Eclipse. Its first development iteration was a part of the master theses, where its abstract syntax and initial concrete syntax was proposed [79]. The DSL implementation started with domain analysis that brought concepts and details, so it was possible to develop the DSL meta-model. Among different behaviour paradigms for visual language, after extended analysis, it was chosen to represent a language through behaviour trees. This last paradigm is an alternative to the state machines and comprehends a hierarchy of behaviours, with an objective to fulfil. Each node may have a specification that determines how the actions of its children will be executed, which may be in parallel or sequentially.

The first evaluation session was with children between 7–10 years old, and with adults. The evaluation with adults helped to validate the language's functions, while evaluation with children provided ideas how to fill in the gap of approaching this language to the youngest. After the second development cycle, we applied our approach to compare a Visualino with the well-known Lego[6] commercial DSL which paradigm is based on the building blocks composition. We illustrated the evaluation model from this iteration in Section 5.2. This evaluation helped to understand which features are missing or can be improved to make Visualino competitive to the best product in the market with the same purpose. Finally, the developers addressed the recommendations during the third development cycle and renamed a language to Gyro. We further compared the Gyro to the existing low-cost alternative MBlock,[7] and has shown to be more usable when applied in the same context (e.g. with a same type of robots).[8] We also manage to show that the usability of the Gyro significantly improved in terms of efficiency and satisfaction when compared to earlier version of language (Visualino).

The preliminary evaluations helped in timely detecting crucial usability problems of Visualino, as well as detecting the audience that can comprehend the given paradigm. The provided feedback helped in improving the language to support the intended user's needs. A fundamental difference in this work is that the language evaluation was introduced early in the DSL development process so that problems can be found on-time and fixed at a fraction of the cost it would take to fix them if detected only in the deployment phase.

3.3.3. DSE merge

As a final case study reported in [78], we evaluated the tool support developed within this project at Budapest University of Technology and Economics, named DSE Merge, which presents a novel search-based automated model merge [80]. The framework built on top of off-the-shelf tools for model comparison, uses guided rule-based design space exploration (rule-based DSE) for merging models. In general, rule-based DSE aims to search and identify various design candidates to fill certain structural and numeric constraints. The exploration starts with initial models and systematically traverses paths according to specific operations properly identified by operators chosen by the end user. In this context, the results of model comparison will be the initial model, while a target design candidates will represent the conflict-free merged models'.

While many existing model merge approaches detect conflicts statically in a preprocessed phase, DSE carries out conflict detection dynamically, during exploration time thanks to the highlight of conflicting rules and constraint violations. Then, multiple consistent resolutions of conflicts are presented to the domain experts. This technique allows incorporating domain-specific knowledge into the merge process by additional constraints, goals and operations to provide better solutions. The alternative, i.e. baseline support for the model merge problem that is suitable for experimental comparison was found to have two possibilities: Diff Merge[9] and EMF Compare.[10]

The general experimental process started with the Learning session, during which the subjects filled the Background Questionnaire. After this, they proceeded by solving the exercises during the Task session, that was video recorded. Finally, during the Feedback session, participants filled a final questionnaire, rating tools that they have used. During the Learning sessions, the subjects learned about the domain and the tool. The subjects were invited to ask questions and to ask for help only after the presentation. During the Task session, the subjects were not limited by time to solve this tasks. They were not allowed to ask for help, except if they were experiencing some technical or connection problem.

During the pilot session, the cognitive effort for each task was estimated to be similar, the TLX scale[11] is decided to be used just once for each tool that is being evaluated, at the end of Video Session. Based on the results and opinions of the participants during the Pilot Session, it was found that Diff Merge is rated as a more competitive alternative for DSE Merge. The experimental groups were divided in two. One group started the Tool Session by learning about DiffMerge and then DSE Merge, while the second group had learned the other way around.

The most valuable contribution that resulted from this evaluation is the experiment design, instrumentation and metrics that can be easily repeated and reused for similar evaluations of new techniques. This experiment design took a deeper analysis of the subject's profiles, technology, social and physical environment and targeted workflow scenarios, which are defined explicitly and incorporated in a data collection instruments and reflected in the hypothesis.

---

[6] https://www.lego.com/en-us/mindstorms/ (accessed May 4, 2017).
[7] http://www.mblock.cc/ (accessed May 4, 2017).
[8] https://sites.google.com/view/vl-empiricalstudy/home (accessed May 4, 2017).
[9] http://www.eclipse.org/diffmerge/ (accessed May 4, 2017).
[10] https://www.eclipse.org/emf/compare/ (accessed May 4, 2017).
[11] https://humansystems.arc.nasa.gov/groups/TLX/ (accessed May 4, 2017).

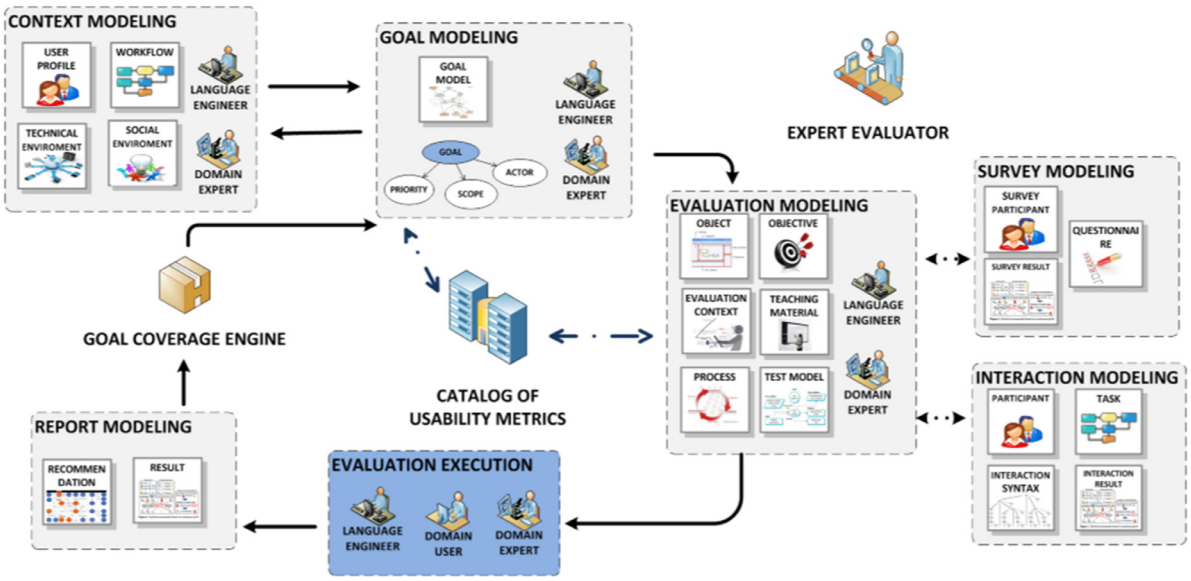

Fig. 3. USE-ME life-cycle.

4. Supporting usability evaluation methods during DSL development (solution implementation)

As already mentioned in Section 2.2, usability engineering aims to increase the awareness and acceptance of established usability methods among software practitioners. Knowledge of the basic usability methods is expected to enhance the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment. It is claimed to affect the user's performance and satisfaction. We introduce usability engineering methods into the DSL life-cycle (see Section 2.1.2) by following an MDD SLE process.

The main concepts supporting the modelling approach are introduced as UML class diagrams. The flow of activities is described by UML activity diagrams. While discussing the supporting conceptual framework, we will use the following symbols;

- [] – main concepts i.e. classes in diagrams,
- ⟨⟨⟩⟩ – attributes and relations of the introduced concepts that are part of the diagram presented in a section,
- ⟨*Italic*⟩ – attributes and relations that are part of diagrams from other sections,
- {} – instantiation of the concept i.e. instance object,
- " – activity in a diagram,
- *'Italic'* – decision in activity diagram.

4.1. The Usability Software Engineering Modelling Environment (USE-ME)

Similarly to other software qualities, usability evaluation should not be just added at the end of the development process. Following the discussion in Section 2, we argue that it has to be included in the development process from the beginning by accurately profiling the end-user and finding the right definition of the problem. The approach is to support usability evaluations, expressed as regular Expert Evaluator activities, in the SLE, backed by a conceptual framework that promotes the iterative UCD evaluation approach.

The Usability Software Engineering Modelling Environment (USE-ME) presents a solution implementation and consists of the following modelling activities that are expected to be performed by the Expert Evaluator (see Fig. 3):

- the **Context Modelling** to define the context of use for the DSL;
- the **Goal Modelling** that sets the objectives for the DSL;
- the **Evaluation Modelling** that instructs the usability experiments;
- the **Survey Modelling** that provides the support for survey/feedback collection;
- the **Interaction Modelling** that defines the interaction task under study;
- the **Report Modelling** that supports management of the collected data.

The goal specifications for the DSL under evaluation are underpinned by the **Catalogue of Usability Metrics** that describes usability goals and its possible measurements and treatments. This Catalogue helps to specify the relevant metrics and interpretation of the results. Finally, the **Goal Coverage Engine** versions the iterative evaluations and defines context

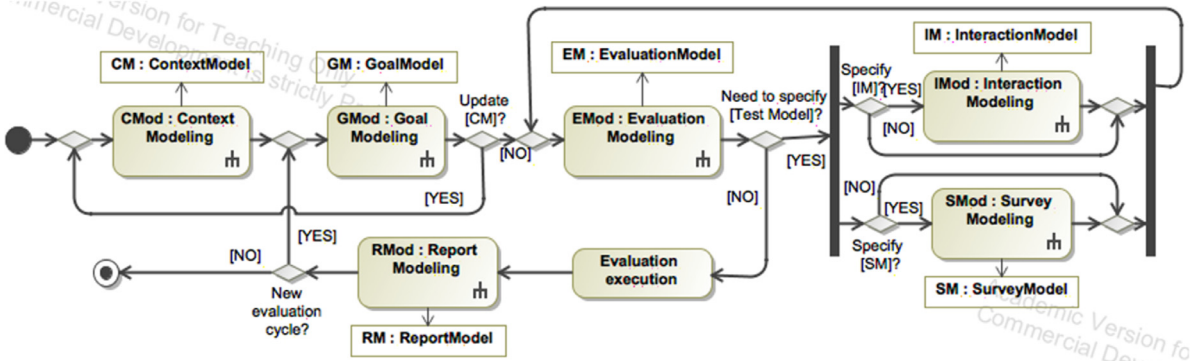

Fig. 4. USE-ME activity diagram.

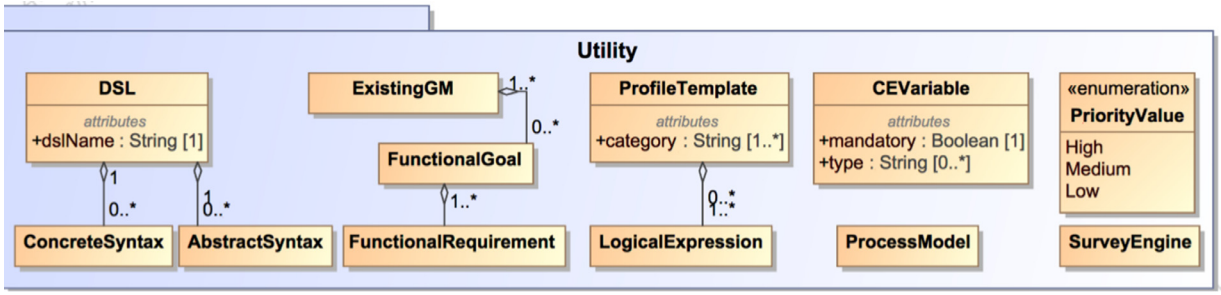

Fig. 5. Utility package.

coverage for the validated goals. The Expert Evaluator in certain modelling activities is supported by artefacts and feedback provided by other DSL stakeholders.

We present the process of usability evaluation using a USE-ME framework as an activity diagram (see Fig. 4). First, it is necessary to produce a [Context Model (CM)], which supports the description of language context, based on which is possible to describe the usability goals as a [Goal Model (GM)]. The Language Engineer and the Domain Expert are involved into 'Context Modelling' and 'Goal Modelling', as they are expected to contribute to the interpretation of development artefacts (i.e. domain model, feature model, abstract syntax, etc.) and review of produced specifications. While specifying the goals and their scope, it is likely that the new context elements are found which are relevant for the use of the DSL. In this case, it is necessary to *'update a CM'* and proceeds with specifying a [GM] till we have at least one usability goal for which only actor representing Expert Evaluator is responsible. Further, it is possible to define an [Evaluation Model (EM)], which highlights evaluation goals and their corresponding evaluation steps. The Language Engineer provides the developed artefacts (i.e. DSL, documentation, validation tests) and helps to prepare the evaluation. There is a *'need to specify a [Test Model]'* or reuse existing one. The [Test Model] is crucial for the assessment process and can be defined as an [Interaction Model (IM)] or/and a [Survey Model (SM)]. These two modelling activities depend on the same [EM] and should be performed in parallel to complement each other. However, for certain types of evaluations, it is not necessary to develop both models. For instance, when performing a heuristic evaluation, a checklist implemented as an [SM] can be sufficient. When the [EM] is ready, we can proceed with the 'Evaluation execution' in which Domain Users are included as subjects, while the Language Engineers and Domain Experts help in execution. The next step is to analyse the stored results of the test models and to create the [Report Model (RM)], that recommends a [GM] extensions and calculates a success factor of the evaluated usability goal. Finally, it is up to all DSL stakeholders to decide to continue to *'new evaluation cycle'* or finalise the assessment period. Ideally, this decision will eventually indicate the end of the development cycle.

4.1.1. Utility

In this section, we introduce the Utility package, which contains artefacts that are reused from the existing development process of DSL. It includes the following concepts (see Fig. 5):

- *DSL* – represents a language artefact, which can have associated to it an *Abstract Syntax* and a *Concrete Syntax*. This can be seen as a reference object to DSL under development and its progressive design implementations.
- *Existing GM* – goal model is a standard development artefact in MDD. If the DSL is evolving from the previous state, or usability analysis is performed in later phases of development cycle like implementation or testing, we already could have an existing GM from which we should reuse the knowledge. We will not focus during this analysis on *Functional*

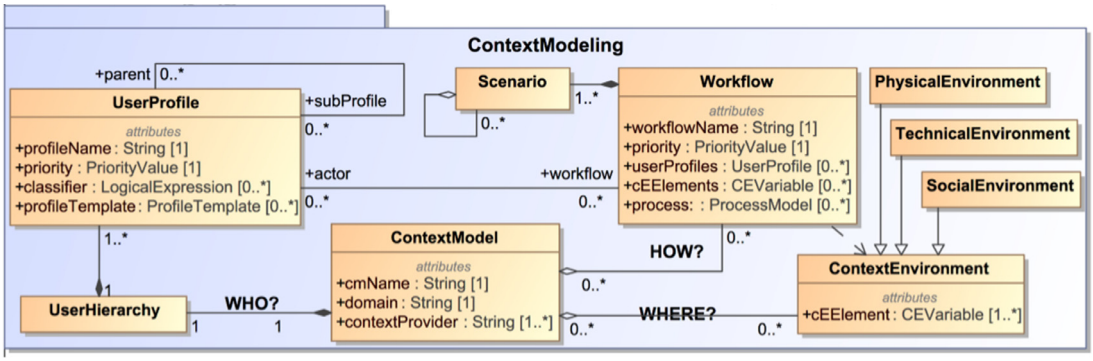

Fig. 6. Context model class diagram.

and *non-functional* goals and requirements. They are seen as a part of an existing goal model. Instead, we are addressing usability goals and requirements which are often dependent on the satisfaction of existing ones.

- *Profile Template* – is a classification artefact for a user profile. It is used for categorisation of user profiles and specifying their features as *Logical Expression* (e.g. regex, ocl expression, boolean). These expressions contain a list of concrete (e.g. 'age'⟩7) or abstract (e.g. 'age'=int) specifications. Often we may find sources which can be used to import these values which describe potential users, for example, client human resources.
- *CEVariable* – is a variable describing the environment of the language under development. Their elements are parts of the architectural design (e.g. feature diagrams), or the specifications of language supporting equipment and dependent tools (e.g. sensors, operating systems, interaction equipment).
- *Process Model* – is an artefact which refers to business process models designed during DSL development. Additionally, specifies the experiment processes designed as a part of this approach.
- *Survey Engine* – represents an artefact which is connected to existing survey platform (e.g. Google Forms, Survey Monkey, etc.).
- *Priority Value* – represents predefined priority values, which are set to be 3-scaled with values [High], [Medium] and [Low].

4.1.2. Context modelling

As pointed in the previous chapter, the evaluations are context dependent. In consequence, the DSL's intended Context of Use should be specified when answering 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?

We argue that it is sufficient to describe the **Context Model (CM)** with the following concepts (see Fig. 6):

- *User Profile* – helps us to define *who* will use the DSL. The user modelling activity [81] customises and adapts the language to the user's specific needs. Each profile can be instantiated as a subcategory of the main stakeholders during the DSL evaluation, namely {Language Engineer}, {Domain Expert}, {Domain User} and {Expert Evaluator}. Also, it stores the ⟨⟨priority⟩⟩ regarding the current evaluation needs (in the ongoing development cycle) expressed as an enumerated ⟨*Priority Value*⟩. Any additional [User Profile] is justified by a ⟨⟨classifier⟩⟩, presenting any ⟨*Logical Expression*⟩, which justifies new ⟨⟨subProfiles⟩⟩. Usually, during this classification, we categorise new profiles by particular ⟨⟨parent⟩⟩ characteristics into at least two distinct sets.
- *User Hierarchy* – is the prioritised categorisation of the [User Profile] presented by the diagram in which each ⟨⟨subProfile⟩⟩ inherits the properties of its ⟨⟨parent⟩⟩. These properties are documented using ⟨*Profile Templates*⟩, and they describe characteristics such as the expected background information (e.g. demographic data, education, special needs/disabilities) and relevant experience with computing and domain activities (e.g. expected knowledge sets, ontology). The [User Hierarchy] represents the user-centred meta-knowledge which categorises profiles and identifies shared semantics.
- *Context Environment* – describes *where* the DSL is to be used. Namely, it can be defined as:
- *Technical environment* – specifies the information about the user's system usage. In other words, software, hardware and network environment that describe the users': i) working equipment (e.g. interacting device specification); ii) DSL operating equipment (e.g. storage, calculation libraries); iii) operation systems; iv) supported platforms; v) frameworks; vi) dependent software tools; vii) and, technical usage conditions (such as display capabilities, connection bandwidth, etc.).

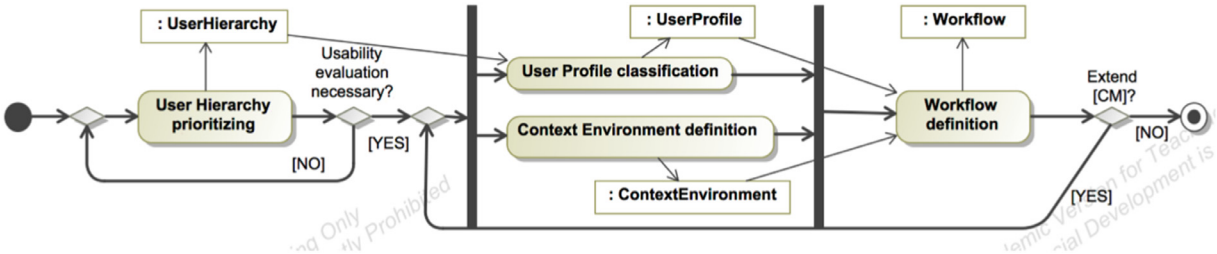

Fig. 7. Context modelling activity diagram.

- *Social environment* – expresses the relation of the DSL to the users' working environment concerning the social context i.e. situational environment [82]. The referred situations are used to model a conceptual framework for representing a given the social context of the semiotic environment in which the users exchange meaning. This model recognises the fact that technology is not developed in isolation but as a part of the wider organisational environment.
- *Physical environment* – describes the working equipment and organisation of the physical environment in which the user interacts with the system, as well as the models of the physical systems and their natural environment that are being affected by the use of the DSL.
  The specification of the technical, social or physical elements is stored as a ⟨⟨cEElement⟩⟩ which is represented as an environmental variable ⟨*CEVariable*⟩. The definition of the [Context Environment] instances can be supported by the integration of existing requirement engineering approaches that support traceability of the environmental variables during the software development [83].
- *Workflow* – describes *how* the DSL is to be used by documenting prioritised user workflows, which reflect a group of tasks relevant for specific ⟨⟨userProfiles⟩⟩ or environment elements i.e, ⟨⟨cElements⟩⟩. The [Workflow] can be defined by using the existing tools for modelling processes (e.g. BPMN [84], UML diagrams [85], or by user requirements notation ITU standard [86]). These models are referred by ⟨⟨process⟩⟩ attribute, which reflects ⟨*ProcessModel*⟩
- *Scenario* – represents a concrete task, i.e. use case, that produce the possible paths and outcomes. Each [Scenario] is describing the user interaction activities, executed by the specific [User Profile] or ⟨⟨cEElement⟩⟩ from [Context Environment]. The [Scenario] can be further decomposed in a more basic functional actions inside of the system that can be connected to a DSLs domain model, preferably to its concrete or abstract syntax features.

The Context Modelling activity, specified by the diagram in Fig. 7, engages all stakeholders included in the development. It starts with 'User Hierarchy prioritising', for which initial [User Hierarchy] is represented by DSL stakeholders, if not specified differently. This prioritisation helps stakeholders to decide if *'the usability evaluation is necessary?'*, and also to indicate from which stakeholder's perspective. Usually, the high priority for the End User profile justifies the investment in reactive experimental approach during the evaluation. Furthermore, it is necessary to perform 'User Profile classification' during which the relevant ⟨⟨classifiers⟩⟩ are identified, resulting a new [User Profiles] in the hierarchy. The parallel activity to this user identification is a 'Context Environment definition', during which the physical, social and technical environmental elements, i.e. ⟨⟨cEElements⟩⟩, are specified. Finally, after knowing who will use the DSL and where it is going to be used, the expert evaluator can describe why the DSL will be utilised by [Workflows] created during the 'Workflow definition'. It is likely that the evaluation expert will identify at this point missing context element specifications, or the need for a new [User Profile] classification. In this case, when the process reaches the decision point *'Need to extend CM?'* it is possible to return and extend associated models. Finally, when there are no new insights after the Workflow specification, the Expert Evaluator is ready to finalise this activity.

4.1.3. Goal modelling

The *Goal Model (GM)* specifies objectives that a user may have while using the DSL. This model describes '*why* is the new language being developed?', and this makes it a central part of the USE-ME framework through which we are expected to model context dependent goals and trace the success of the DSL under development (see Fig. 8).

Goals capture, at different levels of abstraction, the various objectives the system under consideration should achieve [87]. A goal model is a well-known element of requirements engineering that is also widely used in business analysis. They have been used in SLE to model non-functional requirements [88] and early requirements for software systems. There are already several approaches for goal modelling; namely, KAOS [89], i* [90] and Tropos method [91]. These approaches already support the core concepts of goal modelling, i.e., goal hierarchy and quantitative and qualitative relations. However, they should be specialised and, in some way extended to serve our research objectives. Usability is found in these approaches as a non-functional requirement or a soft goal. In our approach usability is meant to be highest level objective (i.e. goal) for developed DSL.

The [GM] of USE-ME can represent extensions to ⟨*Existing GM*⟩ of the language, or can be built from scratch. It is described by **Usability Goal** using a GOMS (Goal, Operators, Methods, Selection) [50] and/or GQM (Goal, Question, Metric)

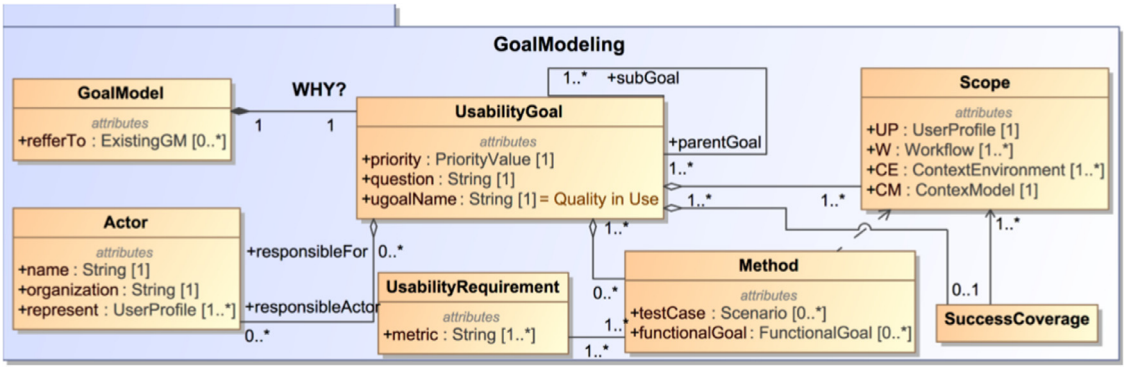

Fig. 8. Goal model class diagram.

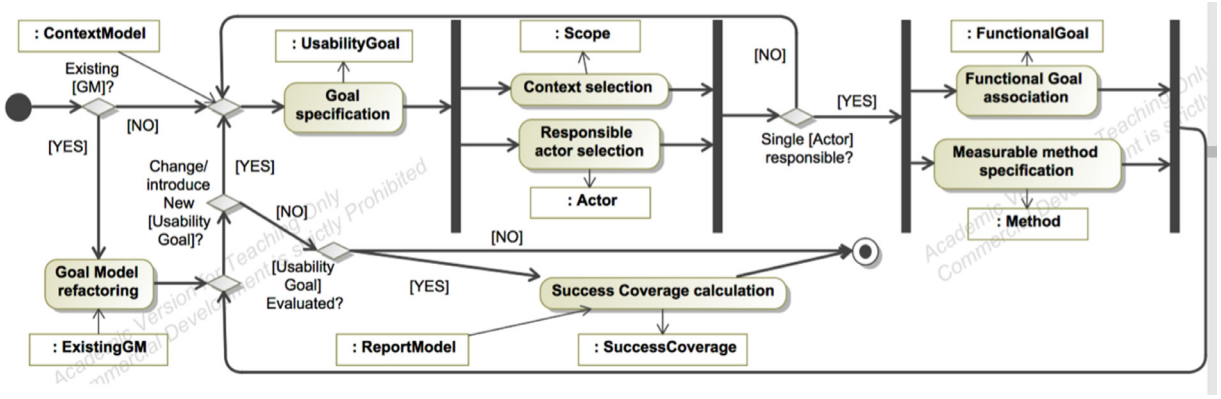

Fig. 9. Goal modelling activity diagram.

analysis [92]. The root [Usability Goal] of the [GM] is the Quality in Use, and represents ⟨⟨parentGoal⟩⟩ of any newly defined ⟨⟨subGoal⟩⟩. Goals are characterised by:

- *Actor* – is a specialisation of DSL stakeholder, namely an instantiation of a [User Profile] from the [CM], to which are assigned the responsibilities (⟨⟨responsible For⟩⟩) for the associated [Usability Goal]. The resulting responsibility model clearly distinguishes stakeholders to which a [Usability Goal] is related (i.e. [Scope]) from the ones that are responsible for achieving it (i.e. [Actor], ⟨⟨responsibleActor⟩⟩).
- *Scope* – that is described by the instance of [CM] for which [Usability Goal] applies. By default, each goal of [GM] applies for the complete [CM], if it is not specified differently (e.g. specific ⟨*User Profile*⟩, *Workflow*⟩ or *Context Environment*⟩). The priority level of [Usability Goal] can be inherited from the [Scope] to which it is related to.
- *Method* – defines the measurable requirements (i.e. *Usability Requirements*) that contribute to the achievement of the goal. It consist ⟨⟨testCase⟩⟩s taken as individual ⟨*Scenario*⟩s from [Scope]). These ⟨⟨testCase⟩⟩s can be used to evaluate the requirement. A [Method] is often dependent on the development stage of a DSL and the context of the planned evaluation. For instance, during early evaluations without a functional prototype, we can evaluate the readability and understandability of the design; when assessing early ideas, the focus can be to evaluate the feasibility of the language construction or usage process, and finally when having functional prototypes we can perform experiments with using a DSL. A [Method] is associated with ⟨⟨functionalGoal⟩⟩ that should be tested by Language Engineer. The ⟨*Functional Goal*⟩ represents the functionalities that are to be provided to support the execution of a ⟨⟨testCase⟩⟩.

Success Coverage – reflects the evaluated context coverage and can be represented by the percentage of the [CM] (i.e. [Scope]) to which each success factor apply. As we are evaluating usability, we find that the metric representing the score for the evaluated scope can also rate on the scale −1 to 1, where the (1) indicates that measured experience was positive, while (−1) indicates a negative experience while using the given DSL. Further, the representation of success can be represented by the use of Kiviat diagrams [13] reflecting different requirements that were taken into consideration, or by single project dependent indicator predefined by stakeholder [93].

The Goal Modelling activity, specified by the diagram in Fig. 9, starts with identifying if there is an *'Existing [GM]?'*. In case there is a [GM], previously created within USE-ME or another context, the Expert Evaluator performs '[GM refactoring]' regarding the evaluation Priority and a structure and identifies if there is a need to 'Change or introduce a new [Usability Goal]'. Further, the [Usability Goal] is changed/introduced during 'Goal specification' activity during which Expert Evaluator consults Usability Catalogue (Section 6.2.). In parallel, a [Scope] is associated with the goals during 'Context selection' and an

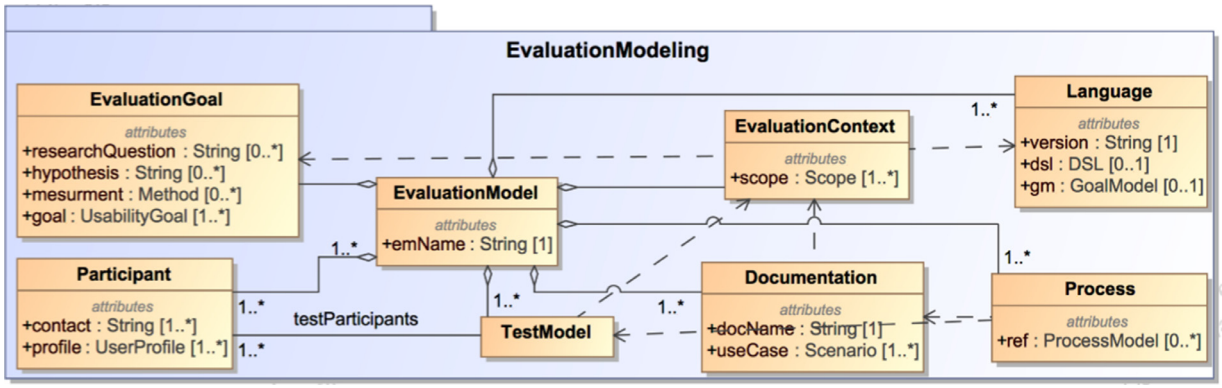

Fig. 10. Evaluation model class diagram.

[Actor] during 'Responsible actor selection'. Next, it is necessary to verify if the *'Single [Actor] is responsible?'*, meaning that the goal is decomposed into a ⟨⟨subGoals⟩⟩ for which only one [Actor] is ⟨⟨responsibleFor⟩⟩, and by this making it ready for evaluation. This is followed by 'Functional Goal association' during which all the functional prerequisites that need to be verified for evaluation are provided by the Language Engineer and associated with [Method] justifying [Usability Goal] evaluation. In parallel, the Expert Evaluator defines an evaluation [Method] during 'Measurable method specification', which aggregates [Usability Requirements] and ⟨⟨testCases⟩⟩. This activity requires consultation of the Usability Catalogue, to reuse the existing metrics if possible. Finally, if the *'[Usability Goal] is evaluated'*, then it is feasible to perform a 'Success Coverage calculation' that indicates the scope for which the goal was evaluated, as well as the evaluation results.

4.1.4. Evaluation modelling

Although DSLs' development process is not the same as the one of the UIs, typically DSLs have an explicit underlying model (thanks to meta-models or grammars), while UIs models are usually implicit in their implementation. In general, there is no distinction between DSLs and UIs from the end users point of view, so their evaluation should essentially validate human–computer interaction (HCI) [22].

Therefore, we can reuse the common techniques used for UI usability evaluation; e.g. UCA (Usability Context analysis) [94], MUSiC (Metrics for Usability Standards in Computing) [95] and MAGICA [96] methods and tools. UCA ensures that the user-based evaluation produces valid results, by specifying how important factors are to be handled in the assessment. MUSiC and MAGICA provide modules that can be used for measuring user satisfaction, user performance, cognitive workload, task completion time and analytic measurement.

The evaluation modelling activity is expected to support the application of techniques mentioned above that are represented by the **Evaluation Model [EM]** (see Fig. 10). [EM] expresses the purpose of evaluating a certain objective (an instance of [GM]) for a DSL in the particular context (an instance of [CM]). Therefore, the prerequisites for the evaluation modelling are to have a [GM] and [CM] for a DSL under evaluation. This activity is supported by modelling of:

- *Evaluation Goal* – defines the experimental ⟨⟨hypothesis⟩⟩ and ⟨⟨researchQuestions⟩⟩. It is related to ⟨⟨goal⟩⟩s specified in [GM] and inherits its ⟨*Methods*⟩ which can be introduced as ⟨⟨measurements⟩⟩.
- *Language* of experimental study – is a ⟨*DSL*⟩ under evaluation. The experimental modelling is supporting more than one [Language] for a case of comparative evaluations (e.g. with the baseline that can be an alternative system or previous DSL version).
- *Evaluation Context* – specialisation of the [CM] that describes the ⟨*User Profiles*⟩, ⟨*Workflows*⟩, and ⟨*Context Environments*⟩ taken into consideration during execution of the experiment. This model preferably should reflect the intersection of the [CMs] that specify [Languages] of the evaluation study, in which each [CM] reflects the ⟨*Scope*⟩ of [Evaluation Goals].
- *Participant* – refers to the actual participants of the study, which are expected to match the [User Profile] included in the [Evaluation Context]. It also stores contact information of experiment subject.
- *Documentation* – presents teaching materials for the [Language] under study, e.g. videos, guided examples with image annotations, presentations. These materials can be stored in a shared document repository with version support (such as Wiki).
- *Test Model* – describes the usability testing activities that are not learning treatments, i.e. questionnaires, interviews, and observations. The creation and execution of these models are supported by USE-ME sub-activities of survey modelling and interaction modelling, described in the following sections.
- *Process* – defines the concrete design for evaluation by modelling the activities that should be performed with specific [User Profile] that is described by the [Evaluation Context]. The choice of appropriate treatment is guided by its probability to evaluate the [Evaluation Goal]. These activities are represented by [Documentation] and [Test Models]. They specify the flow of learning treatment activities that are modelled in the evaluation [Process].

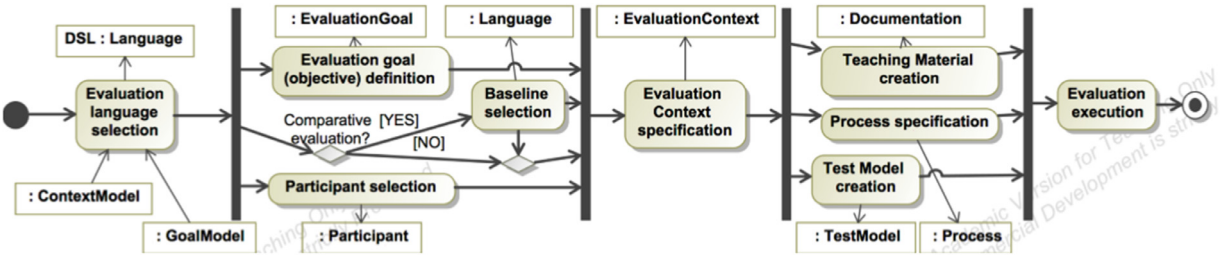

Fig. 11. Evaluation modelling activity diagram.

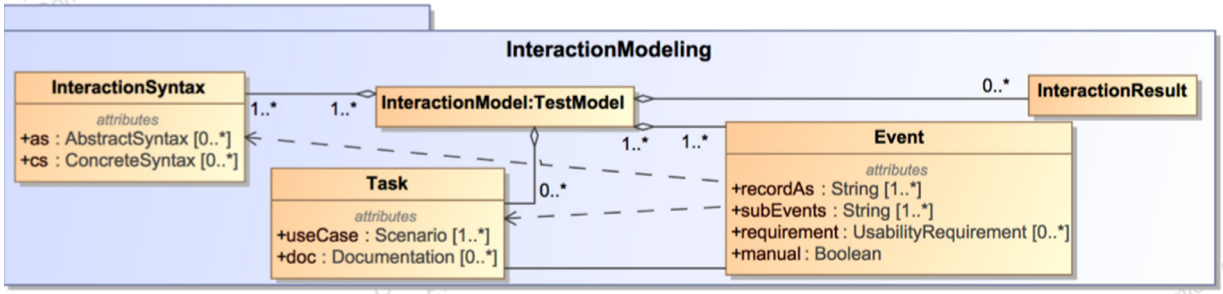

Fig. 12. Interaction model class diagram.

When preparing an evaluation (see Fig. 11), the evaluation expert performs an 'Evaluation language selection' where the DSL under development is chosen to be an evaluation object. Next, he defines the experimental objective and participants during 'Evaluation Goal definition' and 'Participant selection' activities. He also decides if he will perform a 'Comparative evaluation?' and, if so, which another language he will use ('Baseline selection'). The 'Evaluation Context specification' activity defines the context that is considered during evaluation, taking into consideration ⟨⟨profile⟩⟩ of selected [Participants] and the ⟨⟨scope⟩⟩ of the [Evaluation Goal]. This activity can be extended by the context modelling activity if the existing [CM] is missing relevant information. Before the 'Evaluation execution', it is necessary to produce [Documentation] during the 'Teaching Material creation', define evaluation [Process] by 'Process specification' and prepare the [Test Model] during 'Test Model creation'. The 'Test Model creation' activity selects a previously created [Test Model] or calls the Interaction Modelling or Survey Modelling to produce the new or refined [Test Models].

4.1.5. Interaction modelling

The [Test models] that are defined as **Interaction model [IM]** (see Fig. 12) encapsulate summative methods for measuring usability over concrete tasks that involve interaction with at least one [Language] (i.e. DSL under evaluation or its alternative). The [IM] supports capturing of the predefined events and providing statistics about their occurrences. It is described by:

- *Interaction Syntax* – reflects the interaction elements from the version of the [Language] of experiment study (for instance DSLs abstract and/or concrete syntax model, or feature model).
- *Task* – ⟨⟨useCase⟩⟩ taken from the ⟨⟨scope⟩⟩ of [Evaluation Context]. It is documented in [Documentation] and represents a concrete task for which the interaction will be analysed.
- *Event* – accounts for the type of data (⟨⟨subEvents⟩⟩) that will be captured from different interaction devices. It describes how it will be recorded and can refer to single events (e.g. eye movement, mouse click, biosignal, gesture), or to the particular sequence of interaction and/or the sequence of reactions (themselves or as a consequence of interaction sequence). Also, events can be associated with the ⟨⟨requirement⟩⟩.
- *Interaction Result* – includes the statistical analysis and results of the executed interaction model, where participants use the [Language] of experiment study to perform a [Task].

Interaction Modelling (see Fig. 13) starts with 'Interaction Task definition', that is interdependent on 'Interaction Syntax analysis', as the task is to be solved by the use of the analysed syntax, following the [Workflow] that is included in the [Evaluation Context]. Based on the [Task], the evaluation expert is ready to perform 'Events specification' during which [Event]s are defined, as well as the way to capture them. Further, he performs the 'Interaction Participant assessment' activity where the experiment [Participants] are assigned to the [IM]. In the same time, the scheme for [Interaction Result] is prepared by 'Interaction Result formatting' activity. Finally, when the [SM] is complete, it is sent to 'Interaction execution' during which results are automatically stored in the [Interaction Result] model.

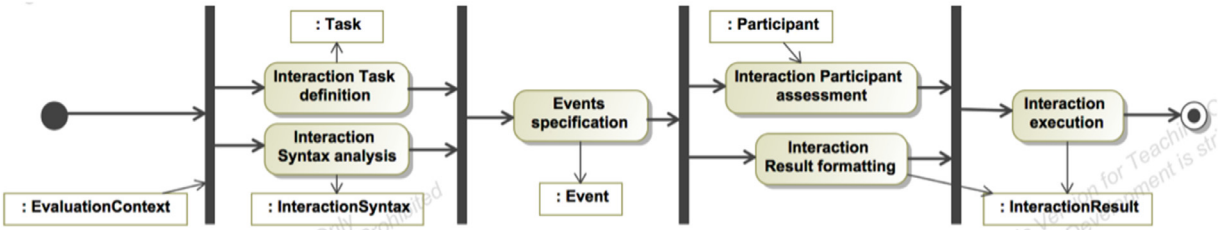

Fig. 13. Interaction modelling activity diagram.

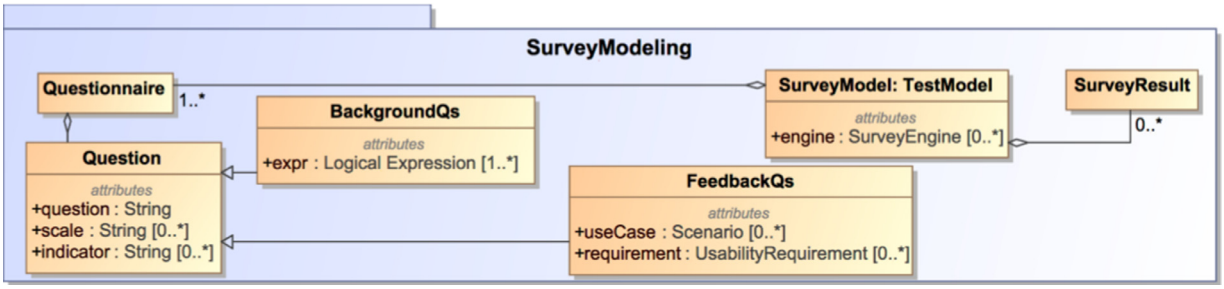

Fig. 14. Survey model class diagram.

4.1.6. Survey modelling

The survey methodology is a field of applied statistics that studies the sampling of individual units from a population and the associated survey data collection techniques, such as questionnaire construction and methods for improving the number and accuracy of the response to the inquiry. A survey is not just the instrument (the questionnaire or checklist) for gathering information. It is a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes, and behaviour [97].

In the case of USE-ME, the Survey Modelling activity supports formative methods for measuring usability. The *Survey Model [SM]* (see Fig. 14) can correlate to any existing [Survey Engine] that automates response collection (e.g. Google Forms, Survey Monkey, mySurveyLab, etc.). It is described by the following concepts:

- *Questionnaire* – defines a particular set of **Question**s (inquiries) to the survey part that can be provided in different forms; on-line, by phone or personal interview, pen and paper, and in advanced interaction environment (a testing environment that supports additional interaction equipment that can capture eye movements, gesture, biosignals). Generally, [Question] has an associated concrete ⟨⟨scale⟩⟩, represented as a list of String values. It can also have a defined ⟨⟨indicator⟩⟩ that helps to get fine-grained variables that can help during the results analysis. The [Question] can be defined as:
- *Background Qs* – designed to collect the information about the participant (e.g. demographic data, education, special needs/disabilities). Each question is related to at least one ⟨*Logical Expression* of participant's ⟨*User Profile*⟩.
- *Feedback Qs* – collects the opinions and reactions about what is being tested i.e. the DSL, its baseline or alternative. In our case it would be expected to support existing appropriate methods such as inspections (usability heuristics [17], quality requirements for DSLs [98], Framework for Qualitative assessment of DSL [20]) and notation assessments (Cognitive dimensions of notations framework [18], Physics of Notations [16]). Each question, or a checkpoint, can refer to concrete ⟨⟨useCase⟩⟩ and/or ⟨⟨requirement⟩⟩.
- *Survey Result* – includes the statistical analysis and results of a survey, that can be generated automatically if ⟨*Survey Engine*⟩ was used, and further customised (e.g. merged results from different questionnaires, formatted, statistically analysed, etc.).

The creation of [SM] (see Fig. 15) starts with asking if *'New questions?'* are needed to be defined for the survey. In the case they are needed, the evaluation expert decides if the questions to be defined are '[Background]', leading to 'Background question definition', or '[Feedback]', leading to'Feedback question definition'. When there are no *'New questions?'*, the expert evaluator is performing 'Survey participant assessment' where the experiment [Participants] are assigned to the [SM]. In the same time, the scheme for [Survey Result] is prepared in the 'Survey result formatting' activity. Finally, when the [SM] is complete, it is sent to 'Survey execution' during which results are automatically stored in the [Survey Result] model.

4.1.7. Report modelling

The report modelling activity helps on the construction of final reports for experimental assessment specified by the [EM] and encapsulate the results and improvement suggestions into the *Report Model (RM)* (see Fig. 16). It consists of:

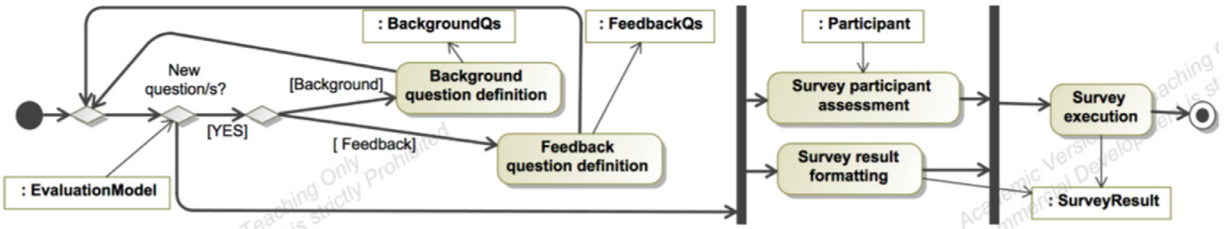

Fig. 15. Survey modelling activity diagram.

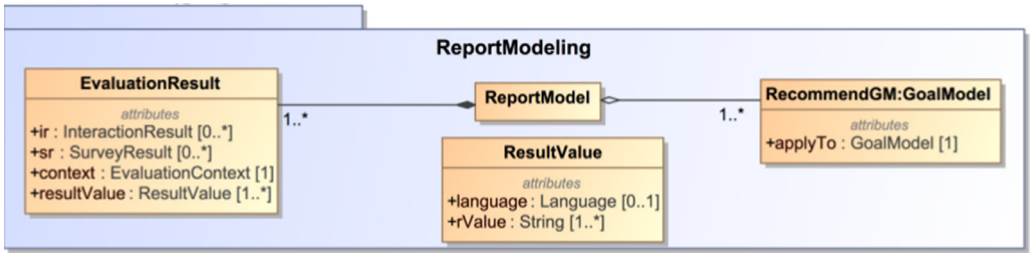

Fig. 16. Report model class diagram.

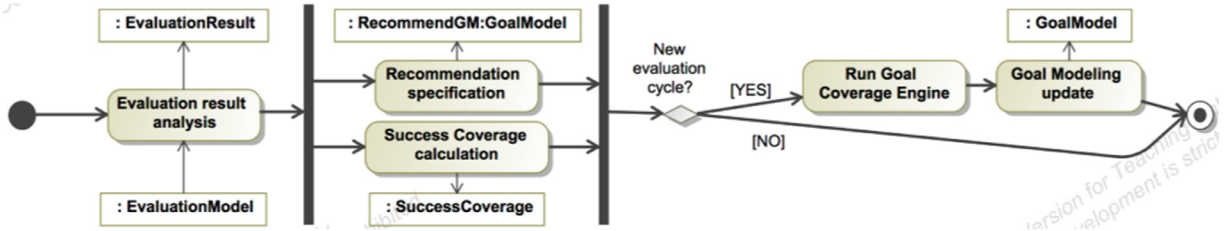

Fig. 17. Report modelling activity diagram.

Evaluation Result – is created based on analysis of the result models from different ⟨*Test Models*⟩ (i.e. ⟨*Survey Result*⟩ and ⟨*Interaction Result*⟩). It represents the interpretation of results over a predefined ⟨⟨context⟩⟩ that is related to the evaluated goal.

RecommendGM – is a recommended [GM], which include updates (changes over or new goals or context elements) to previous [GM] of the evaluated DSL, referred with ⟨⟨applyTo⟩⟩.

In Fig. 17 we describe the report modelling activities. The first activity is to perform the 'Evaluation result analysis'. The first activity includes reasoning about the correlations between different factors accessed by experimental instruments (namely, [SM] and [IM]). Based on these results, the expert evaluator performs a 'Recommendation specification', by designing a [RecommendGM]. At the same time, the evaluator can calculate a [Success Coverage] for the recommended [GM], which helps in making decisions how to redesign goals. Finally, when recommendation is ready if the plan is to enter *'New evaluation cycle?'*, the expert evaluator 'Run Goal Coverage Engine' which validate that all rules are preserved while integrating a recommended [GM] with the initial one. Finally, during 'Goal modelling update' recommendations are accepted or rejected.

4.1.8. Catalogue of usability metrics

This catalogue is seen as a structured knowledge base about the DSLs usability evaluations. It has the purpose to help during two crucial activities of the USE-ME framework:

- *Goal modelling*, where it is used to find existing specifications of the usability goals and requirements. These specifications can be based on standards [51], or existing frameworks which address the evaluations in general [16,19,20]. On the other hand, they can provide examples of which were reported on existing DSL evaluations.
- *Evaluation modelling*, where it is expected to support the choice of treatments for evaluation process activities based on a diversity of goals and the number of available participants, technology, etc. Further, it is meant to register the instantiation of quality in use metrics [99] for selected evaluation objectives and enhance a metric reuse and improvement.

To obtain the first structure of this knowledge base, we are performing an extensive Systematic Literature Review (SLR) whose protocol can be found in [100]. The SLR is extended from previously performed work of Gabriel [12]. The selected publications should report on the human–machine DSLs (excluding the machine-machine languages). The SLR should report on the involvement of domain users in the DSL development process and the evaluation of DSLs usability. The goal is to

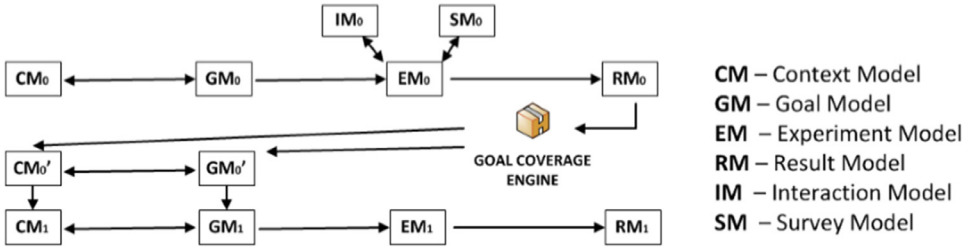

Fig. 18. Model to model transformations supported by Goal Coverage Engine.

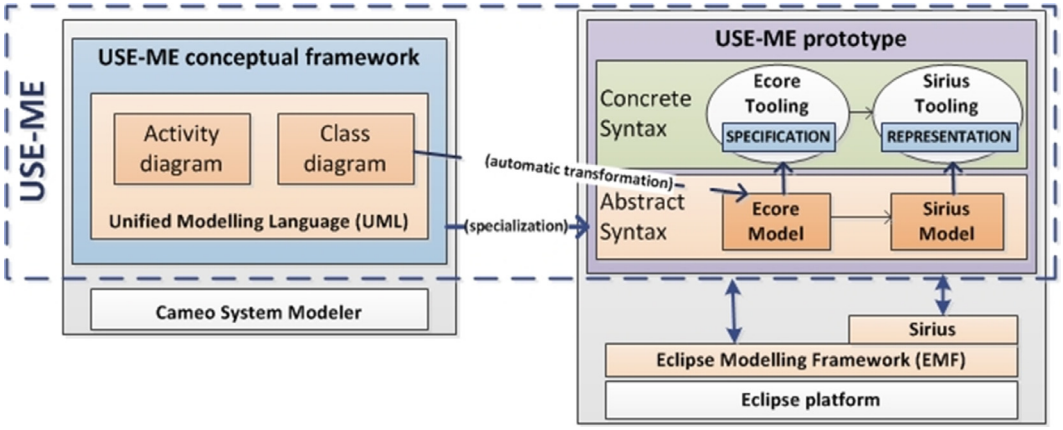

Fig. 19. USE-ME architecture.

identify domain analysis and/or evaluation techniques for developed DSLs. The next step is to summarise the state-of-the-art research trends, as well as to categorise the proposed approaches, techniques, tools and methods for domain analysis and evaluation phases. We are concerned about how selected reports identify the DSL objectives, and how they are evaluated.

4.1.9. Goal Coverage Engine

This engine should provide version support of the USE-ME project models and support their updates by using model merge approaches (see Section 3.3.3), while checking the predefined rules. The Goal Coverage Engine supports all DSL stakeholders to trace different states of models for different versions of the DSL or their assessments, as well as a knowledge base. For each new iteration cycle of the DSL development, the stakeholders need to confirm that the context and goals are still the same, or update them if they have changed. This engine should also check the validity of specific rules for removing/merging/updating USE-ME models. Finally, this engine can be seen as a transformation engine that performs the updates of GMs based on the recommendations of the Report Model and finds the scope coverage for which the goals are validated based on a context instance (see Fig. 18).

5. USE-ME feasibility study (implementation evaluation)

In a previous section, we described the implementation of our approach, for which we present a feasibility study in this section. First, we present implementation architecture which we used for prototype implementation. Further, we illustrate an instantiation of prototype models on an industrial case study. Finally, we perform a pilot empirical evaluation of the implemented tool on four DSL development projects. We have shown that it is feasible to capture all information which is relevant for the evaluation and its use in the result analysis.

5.1. Implementation architecture

The USE-ME prototype [101] was developed as specialisation of conceptual framework (see Fig. 19). The USE-ME conceptual framework was formally specified on Cameo Systems Modeler,[12] a cross-platform collaborative model-based systems engineering environment. This platform enabled us to produce Unified Modeling Language (UML)[13] compliant models and diagrams. Namely, we presented the main concepts as class diagrams and a process as activity diagrams. This diagrams

[12] https://www.nomagic.com/products/cameo-systems-modeler (accessed May 4, 2017).
[13] http://www.uml.org/ (accessed May 4, 2017).

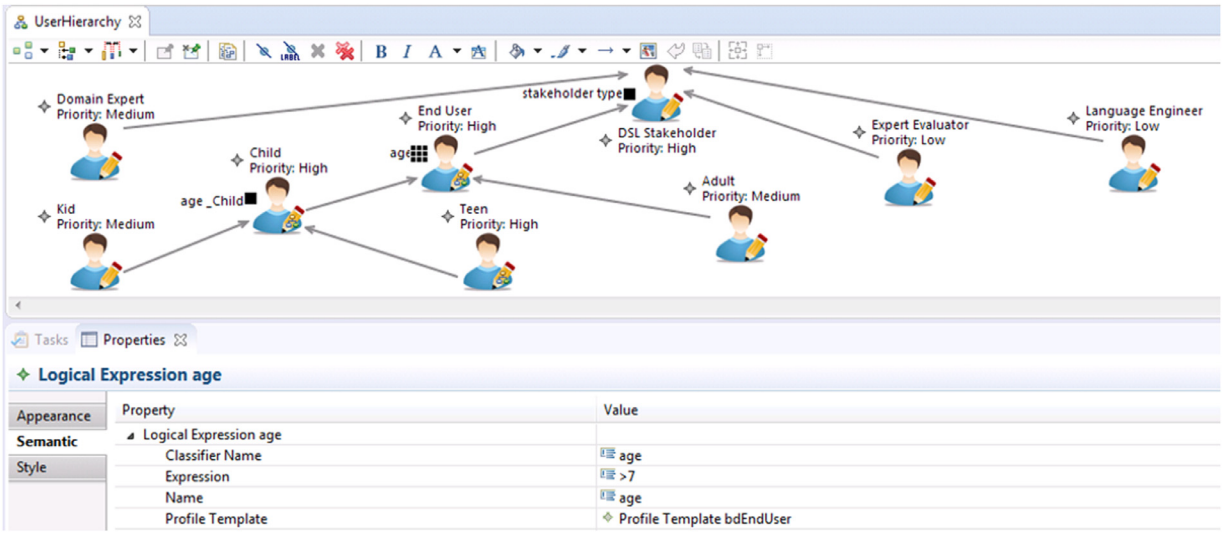

Fig. 20. User hierarchy.

served as a general specification of abstract syntax for the USE-ME approach. They were reviewed by a research group of NOVA-LINCS during a presentation session and trough individual questionnaires.

The USE-ME prototype[14] was developed using Eclipse Modeling Framework (EMF),[15] which is an Eclipse-based modelling framework for building tools and other applications based on a structured data model. From an XMI model specification, EMF provides tools and runtime support to produce a set of Java classes for the model. The EMF framework includes a meta model (Ecore) for describing models and runtime support for the models. The class diagram specified in Cameo System Modeler was transformed to an Ecore model. Due to constraints in the target meta-modelling tool, we had to adapt the model given in pure UML in order not to lose information from the original contents and to address restrictions provided by Ecore (e.g. necessity of containment relationships).

Further, we used Sirius,[16] a platform for developing and using graphical model editors, which is also based on the Eclipse Platform, and in particular the modelling stack based on EMF. The Sirius platform is domain-agnostic, in that it can be used by modellers for any business domain as long as they can describe it using EMF. In our prototype, we used Sirius to declare the visual representation of models instantiated in Ecore. The USE-ME architecture as described in this document supports modellers to design the USE-ME instances in an EMF generated editor with Ecore tooling. It is also possible to redefine and preview the implemented representations by Sirius.

5.2. Visualino case study

We dedicate this Section to illustrate the USE-ME approach with a case study about a free web-based programming language, named *Visualino* (see Section 3.3.2), for rover-like robots. The following description refers to the design of empirical study conducted during the second development iteration of Visualino,[17] where the USE-ME approach was applied systematically.

During the description we use following symbols;

- [] – instantiation of the concept i.e. instance object;
- ⟨⟨⟩⟩ – property of the instance object;
- {} – property value of the instance object;

5.2.1. Context model instantiation

The Context modelling starts with prioritising the initial user hierarchy, which is represented by the DSL's stakeholders. Therefore, we are having a [DSL Stakeholder] as a 'root' element of the hierarchy, and its subProfiles defined as [Domain Expert], [End User], [Expert Evaluator] and [Language Engineer] (see Fig. 20).

For the case of Visualino, the target users are children, which usually are not experienced with programming. As such, the usability evaluation of the language with its end users is highly important. We set up the priority of its [End User]

[14] https://github.com/akki55/useme (accessed May 4, 2017).
[15] http://www.eclipse.org/modeling/emf/ (accessed May 4, 2017).
[16] https://eclipse.org/sirius/ (accessed May 4, 2017).
[17] https://sites.google.com/view/vl-empiricalstudy/home (accessed May 4, 2017).

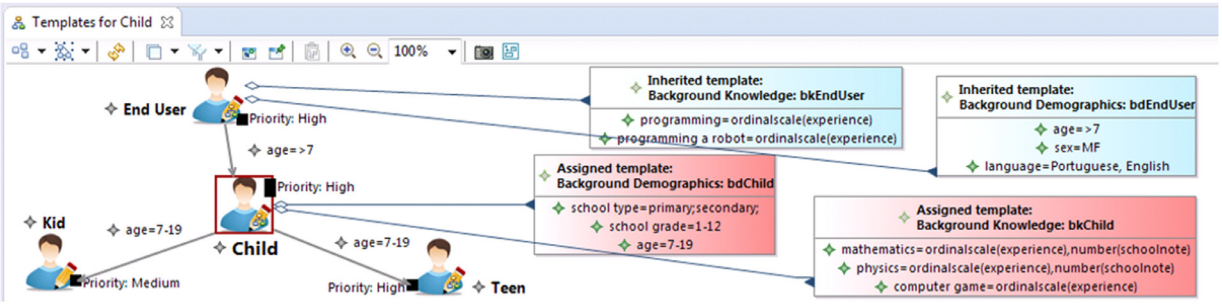

Fig. 21. User profile template.

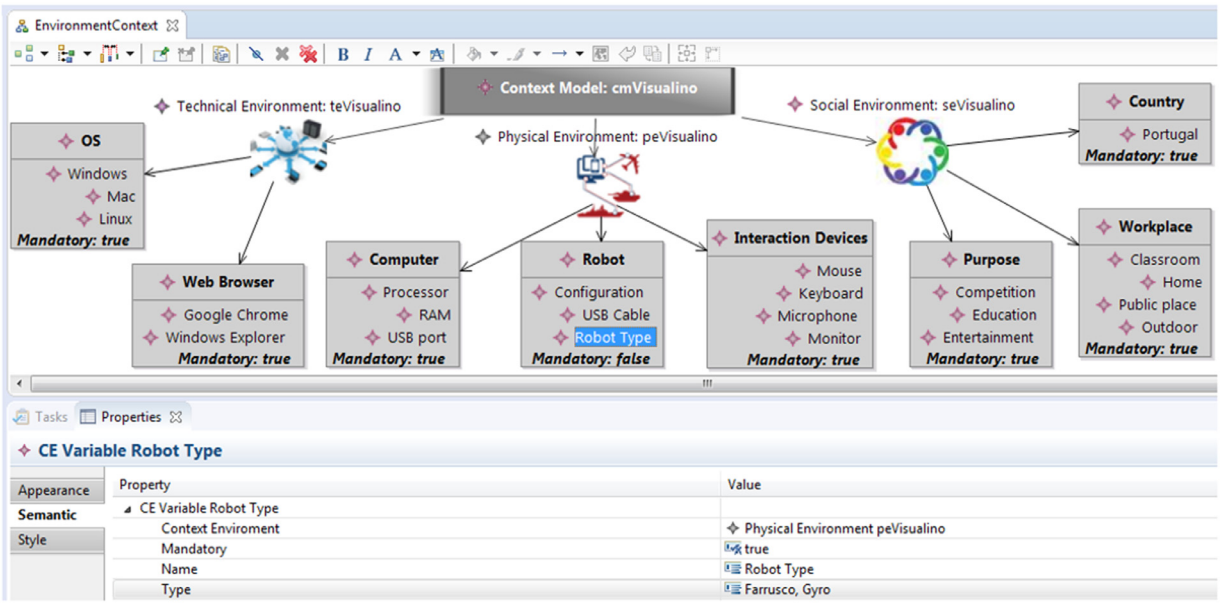

Fig. 22. Environment context.

to be {High}. Having, the lower priority value associated with other stakeholders indicates that the investment in the usability evaluation of these profiles is not as important during the current development cycle. However, as the language develops further, the investment into the evaluation with other user profiles may become more important. For instance, the development iteration may focus on the language extension that supports the [Domain Expert] to specify the environment configurations for different robots. Then, the priority for the [Domain Expert] profile will be updated accordingly. In this case, the priority of the [End User] should stay the same. Setting up a 'lower' priority is reasonable only if the initial purpose of the language has changed. For example, this may occur if the language 'fails' to be adopted by the initial audience, and the opportunity for further development presented itself in a different context.

After the initial prioritisation, we classify user profiles and define profile templates (see Fig. 21). Profile templates are categorised as a {Background Demographics} and {Background Experience}. They are captured with logical expressions. These expressions reflect profile characteristics and their values, concrete or as measurable scales. For instance, the expression ⟨⟨language⟩⟩ indicates that [End User] is expected to speak {Portuguese} or {English}. On the other hand, ⟨⟨programming⟩⟩ is a characteristic that can be measured by {ordinalscale(experience)} which can be later specified with concrete scales.

Logical expressions can be used further as classifiers. For instance, an expression ⟨⟨age⟩⟩ is defined as a condition {age ⟩ 7} for [End User] profile. When creating the 'child' profiles, we use this expression as a classifier, meaning it will be restricted in two or more distinct sets (e.g. {age = 7–19}) for [Child] and {age ⟩ 19}) for [Adult]). It further specialises a [Child] into [Kid] and [Teen] profiles. Each subProfile inherits profile templates and logical expressions that are assigned to their parents.

This kind of structural analysis of domain users for a DSL answers to the question 'Who will use the DSL?'. The next step is to respond the question 'Where will the DSL be used?' by specifying the context environment.

The context environment is described by environmental elements (i.e. CEVariable) such as the social, physical and technical environment (see Fig. 22). These elements can be reused from architectural descriptions, feature diagrams or other

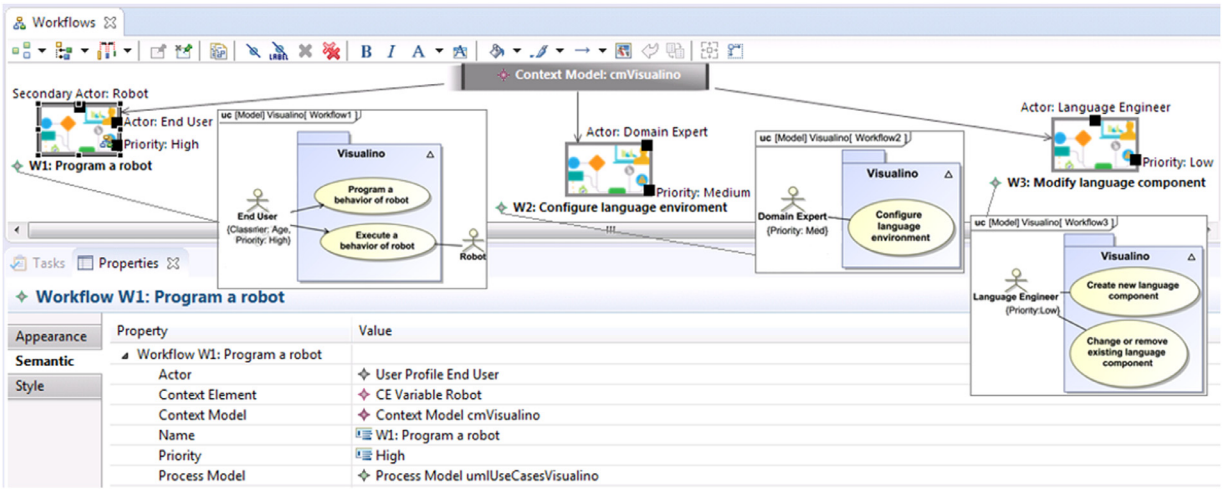

Fig. 23. Workflows.

specification files of the language. For each environment element, it is necessary to specify if it is {Mandatory} or {Optional}. Further, it is possible to define contained environment elements and specify their types as a list of values that can be prioritised.

For example, the social environment of Visualino is characterised by [WorkPlace] which is described by ⟨⟨Classroom⟩⟩ ⟨⟨Home⟩⟩, ⟨⟨Public place⟩⟩, ⟨⟨Outdoor⟩⟩. Further, the [Purpose] can be ⟨⟨Competition⟩⟩, ⟨⟨Education⟩⟩ or ⟨⟨Entertainment⟩⟩. Finally, the [Country] is specialised only to ⟨⟨Portugal⟩⟩, as Visualino is being prepared to be distributed only in this country, for now. The physical environment expects from the user to have a [Computer], that is specified by ⟨⟨Processor⟩⟩, ⟨⟨RAM⟩⟩, and ⟨⟨USB Port⟩⟩, being all mandatory. Also, the physical environment should contain [Interaction Devices], from which ⟨⟨Mouse⟩⟩ and ⟨⟨Keyboard⟩⟩ are mandatory. On the other hand, it is optional but probably convenient to have a [Robot] during the use of the Visualino DSL. Each [Robot] can have the specific ⟨⟨Configuration⟩⟩, reflecting the parts of a robot that are configured to be used. A ⟨⟨Robot type⟩⟩ indicates the version of the Arduino robots (i.e. {Farrusco} and {Gyro}), and a ⟨⟨USB Cable⟩⟩. Finally, the technical environment is specified by the [OS] and the [Web Browser], which are both mandatory for running Visualino. The supported [OS] should be ⟨⟨Windows⟩⟩, ⟨⟨Mac⟩⟩ and ⟨⟨Linux⟩⟩. Their concrete versions are stored in a list. Further, the [Web Browser] indicates a ⟨⟨Google Chrome⟩⟩ and ⟨⟨Windows Explorer⟩⟩.

The answer to the question 'How is the DSL expected to be used?' can build on scenario-based approaches, such as use case descriptions, which define scenarios and workflows. (see Fig. 23). In the case of Visualino, use case specifications are designed with SysUML language. These specifications are referred in the ⟨⟨Process Model⟩⟩ instance. The actors who make part of use case description are represented by different user profiles (users) or environment elements (other systems) that initialise a use case.

A workflow [W1] poses a scenario in which the {End User} (specified as an actor) wants to program a behaviour of a robot and, eventually, execute that behaviour on a physical {Robot} defined in the context environment. In most of the cases, the priority of each workflow can be easily inherited based on the involved user profiles priorities. Following that rule, this case is has a {High} priority. In the workflow [W2] the {Domain Expert} wants to configure the language environment, which inherits the {Medium} priority. Finally, the workflow [W3] represents the case of a {Language Engineer} who wants to create a new language component or change or remove the existing one, in this instance, inheriting the {low} priority.

Several scenarios are associated with a workflow [W1] (see Fig. 24). The first scenario is to program the robot to *move forward* and then *move back*. The second scenario is to program the robot to move along a path similar to a '5'. This scenario includes the following sequence of instructions: *move forward, first turn left, second turn left, first turn right, second turn right* and *stop*. In each turn, the robot needs to execute the move operation using the same amount of time and then to make a 90° angle turn. The third scenario is to program a robot to *move forward* until it *bumps* into some object, then it would *move back* and *stop*. The fourth scenario aggregates the sequences from the previous scenarios. It describes how to program the robot to make a shape of '5', as in the second exercise, but the robot will only turn when it hits the bumper.

5.2.2. Goal model instantiation

In this section, we show how the collected knowledge from the context model can be used in goal analysis (see Fig. 25). We start with an empty model, as there is no existing goal model from the previous development iteration of Visualino. The 'root' goal represents the highest objective of our analysis, that is to achieve [Quality in Use]. It is prioritised as high, indicating that usability evaluation is highly necessary for this development project. Its ⟨⟨scope⟩⟩ is set to be applicable on {all} CM, designed in the previous section. The ⟨⟨actor⟩⟩ responsible for achievement of the highest usability goal reflects all stakeholders included in the development. The {Children robotics expert} and {Visualino development} represent the

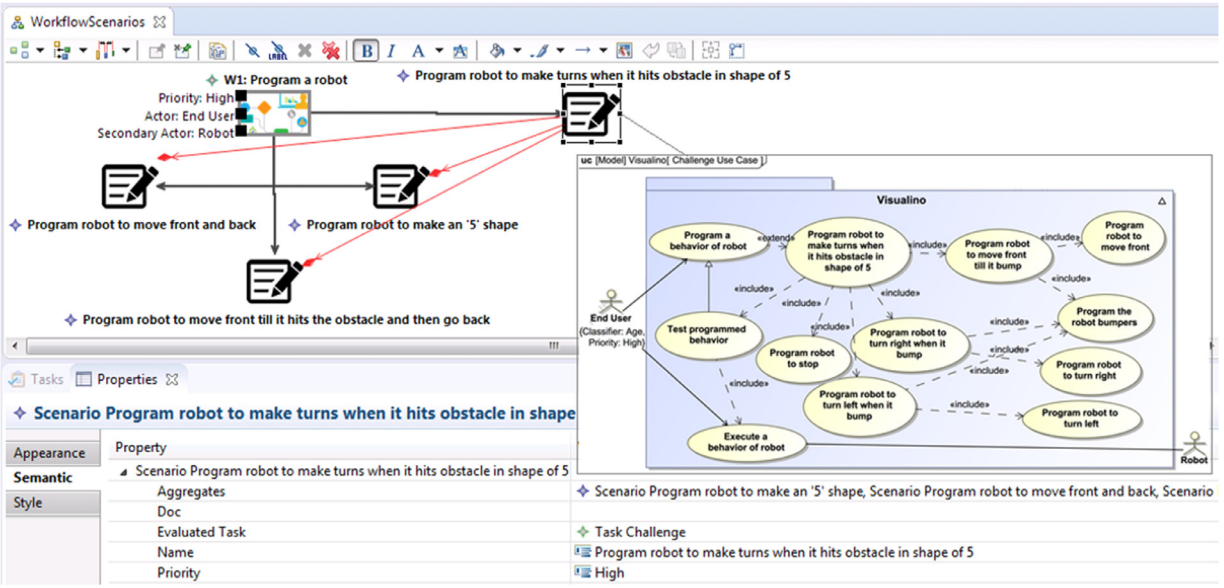

Fig. 24. Scenarios.

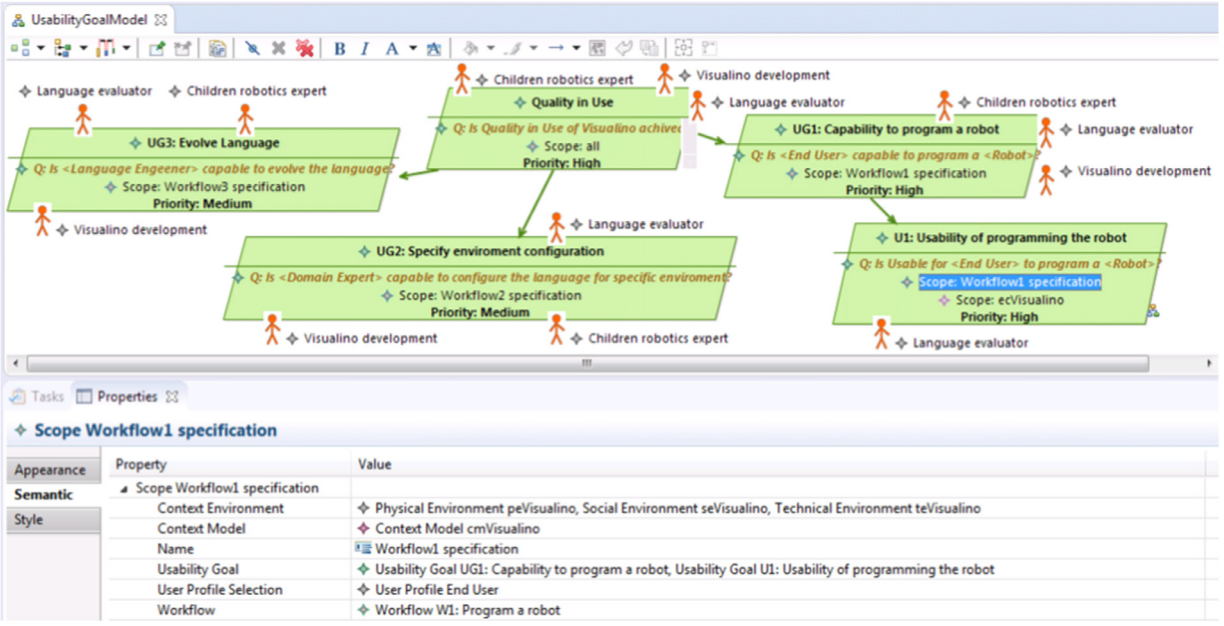

Fig. 25. Usability goal model.

domain experts and language engineers from Artica, while {Language Evaluator} represents the authors that were included as expert evaluators.

The [Quality in Use] is divided into three ⟨⟨subGoals⟩⟩, namely [UG1], [UG2] and [UG3], which reflect different workflows from Fig. 23. The ⟨⟨scope⟩⟩ objects, associated with these goals, restrict a CM in a way that they reflect parts for which each workflow apples. They are only associated to a sub-branch of the user hierarchy model that consider a user profile of the workflow's actor. A goal [UG1] is given the high ⟨⟨priority⟩⟩ as its ⟨⟨scope⟩⟩ reflects the {End User} profile and {Workflow1} workflow. As it is suggested by goal modelling approaches, such as Kaos [102], we further divide this goal into the subgoal [U1] for which the evaluation the responsible is just one ⟨⟨actor⟩⟩, {Language evaluator}.

A usability goal [U1] specifies that the ⟨⟨actor⟩⟩ {Language evaluator} evaluates the [Usability to program a robot] and represents a quality goal usually measured by non-functional requirements (see Fig. 26). It answers the ⟨⟨question⟩⟩ {Is it usable for an [End User] to program a robot?}. Further, a measurable method is specified as [Programming a ⟨Use Case⟩

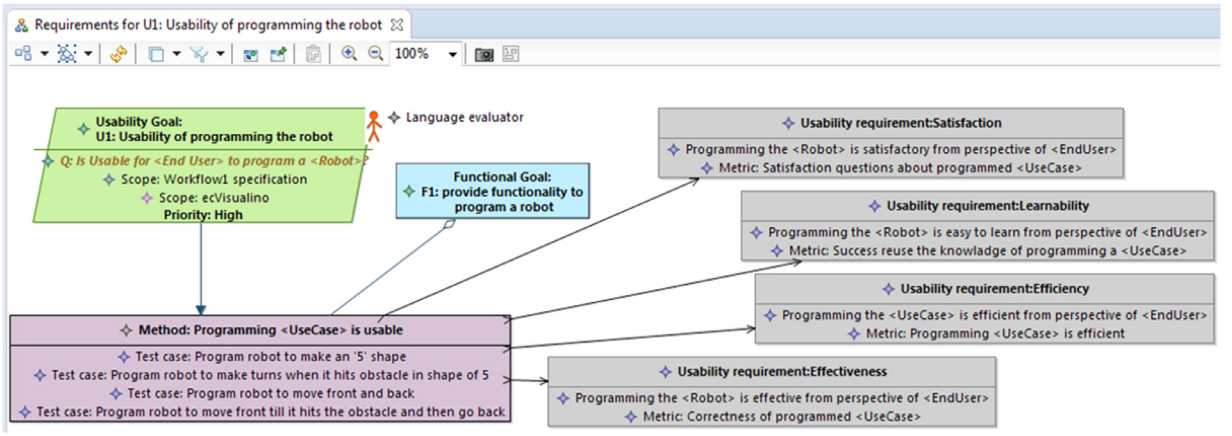

Fig. 26. Usability requirements for usability goal [U1].

is usable] and is related to particular usability requirements. This method is related to a functional goal [F1]. A functional goal specifies that the {Visualino development} verifies that it is possible to program a robot. It answers the ⟨⟨question⟩⟩ {Which scenarios are verified to be programmable using Visualino?} by testing requirements which describe the necessary functionalities to execute the selected ⟨⟨Test case⟩⟩, {Workflow1}.

Further, usability requirements are specified in a way they can be measured in the current phase of development. In the case of Visualino, we focus on the evaluation activity which was planned to be executed after the implementation. In this phase, the following measurable requirements are found to impact the goal [U1]: [Effectiveness], measured by a {Correctness of programmed ⟨Use Case⟩.}; [Satisfaction], measured by {Satisfaction questions related to ⟨Use Case⟩.}; [Efficiency], measured by {Time to program the ⟨Use Case⟩.}; and, [Learnability], measured by {Success to program the learned Use Case.}. Note that other measures could be appropriate in different development stages: for the design phase without working prototype, readability and understandability [103], or for a proof of concept, feasibility and integrability. The catalogue of usability metrics is expected to support this choice and document the metrics from existing experiences.

5.2.3. Evaluation model instantiation

As mentioned previously, language usability can be promoted by combining usability engineering techniques with empirical software engineering. In this section, we present an evaluation model for the empirical study of our target DSL, Visualino, which is specified based on its [CM] and [GM]. The first thing to do is to model the evaluation objectives (see Fig. 27). The primary language of the presented study is a [Visualino]. It was found useful to compare its early prototype to other widely used language for programming robots, namely [Lego], representing a second language in [EM]. The participants were expected to match the {Teen} ⟨⟨profile⟩⟩, as the experiment was performed with secondary school subjects during the ExpoFCT event at Universidade Nova de Lisboa in Portugal. The goal was to evaluate the ⟨⟨usability goal⟩⟩ {U1 – Usability of programming the robot}, specified by GM. Namely, each evaluation goal was to validate ⟨⟨usability requirement⟩⟩: first addressing the {Effectiveness}, and second by {Satisfaction}. For each, we have specified the research question in the form of a ⟨⟨problem⟩⟩ using the GQM template, from which we have derived the ⟨⟨hypothesis⟩⟩.

We analysed the Lego context by giving attention to the workflows and environment that are comparable to the Visualino's, and the ones that are in the scope of the ⟨⟨usability goal⟩⟩ which we are evaluating. During this analysis, it was possible to identify the context in which the evaluation study was appropriate to be performed, and it is described by evaluation context (see Fig. 28). The evaluation context is defined by: ⟨⟨user profile⟩⟩ of [ExpoFCTParticipant], which is {Teen}; the ⟨⟨context environment⟩⟩ that represents where the experiment will take place; and, by the ⟨⟨workflow⟩⟩ that will be used. For instance, for the social environment, the ⟨⟨country⟩⟩ was set to {Portugal}, and the ⟨⟨workPlace⟩⟩ was selected to be a {Classroom}, while the ⟨⟨purpose⟩⟩ was chosen to be a {Competition} to create a motivating environment for the participants. The technical environment was instantiated by ⟨⟨OS⟩⟩ {Windows 7} and the ⟨⟨browser⟩⟩ {Google Chrome}. The physical environment details the information about the ⟨⟨computers⟩⟩, ⟨⟨interaction devices⟩⟩ and ⟨⟨robots⟩⟩ that will be used. The preferred ⟨⟨configuration⟩⟩ for both robots (Lego and Farrusco) is specified, to assure that the robots will have the same functionality. We can note how the environmental elements of Visualino's CM are initialised to concrete values, which are saved to value lists associated with this objects. Finally, the observed ⟨workflow⟩ is the {Workflow1}, which includes all scenarios defined previously, as they are executable in this specific context.

During the execution of the experiment, it was decided to use a 'between groups' design where participants were randomly assigned to either programming a robot using [Visualino] or [Lego], but not both. The evaluation process [emVisualinoProcess] was designed to start with a 30 min learning session, during which the participants should learn the language's concepts and how to solve three basic exercises that reflect the first three scenarios specified in {Workflow1}. This three activities and the tool help were prepared as printable documents, presentations and video demos with the version of the

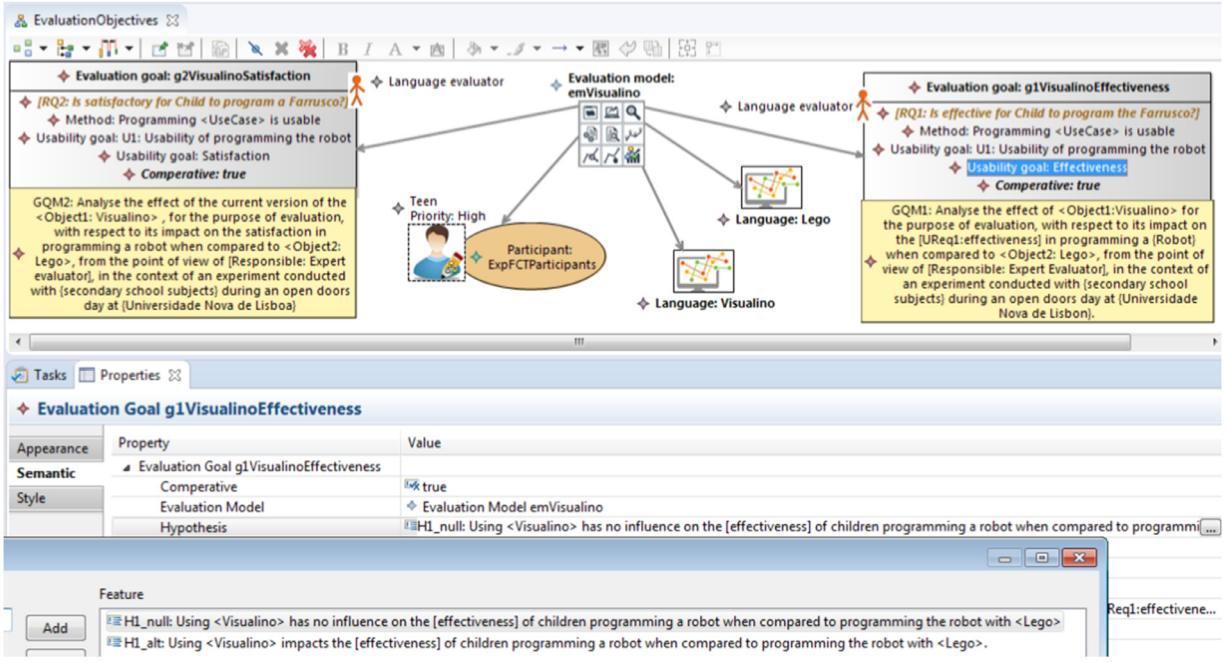

Fig. 27. Evaluation objectives.

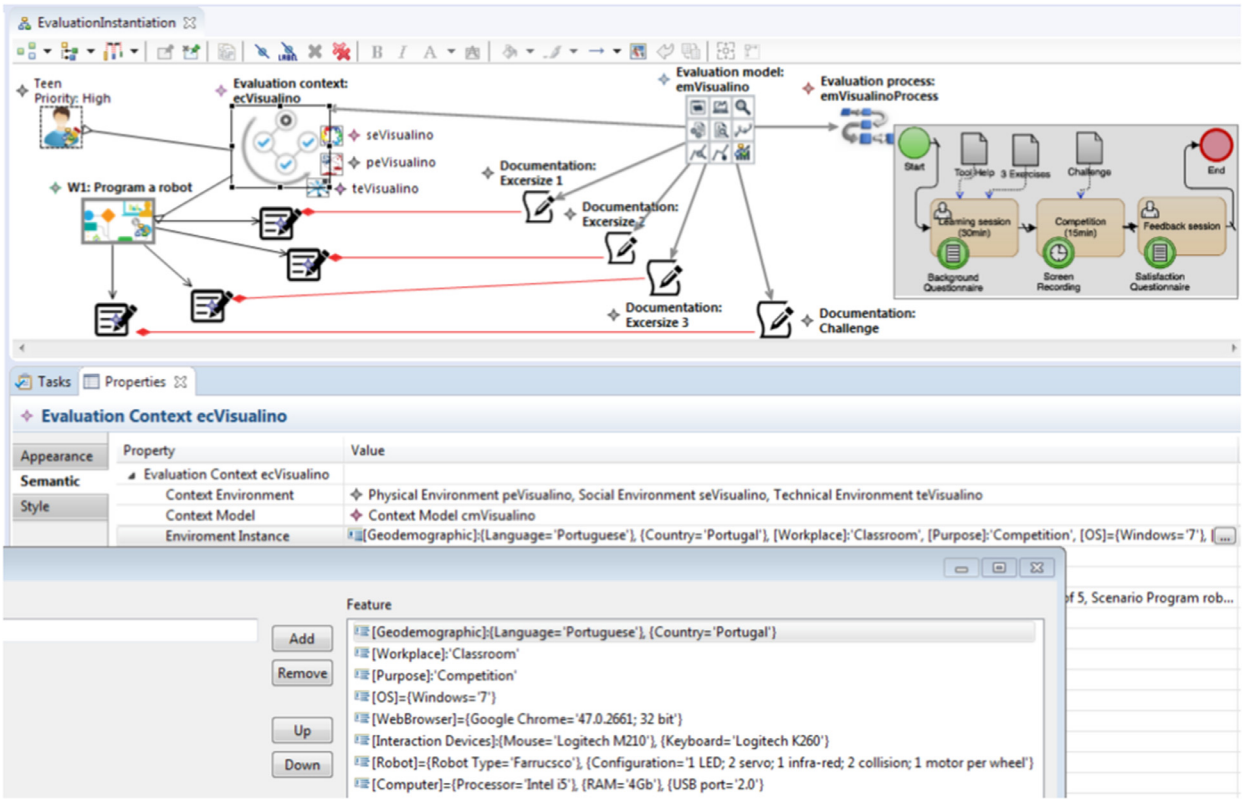

Fig. 28. Evaluation instantiation.

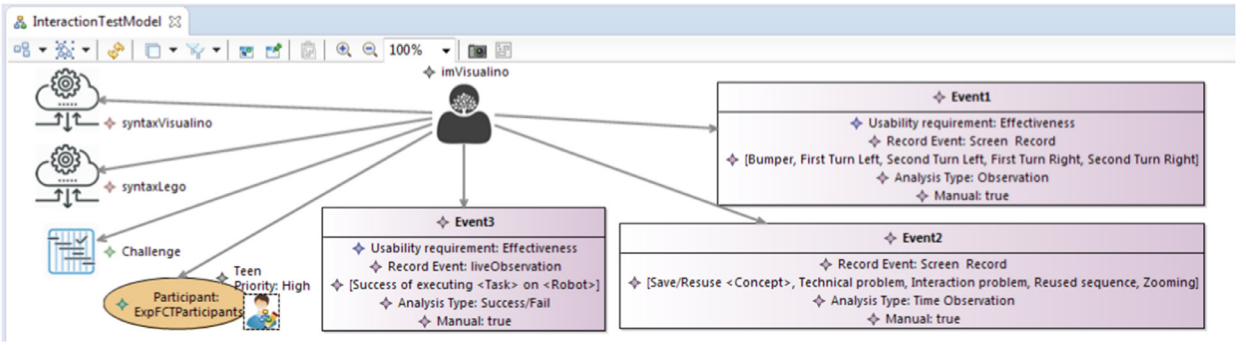

Fig. 29. Interaction test model.

language under evaluation. The materials were saved as documentation objects. Further, it is necessary to define a ⟨⟨test model⟩⟩ through survey modelling for {Background Questionnaire}, which was filled in during the evaluation session. The evaluation continued with 15 min of a competition session during which the participants tried to solve a programming {Challenge} reflecting the fourth scenario, which was aggregating the sequences from the previous scenarios. The ⟨⟨test model⟩⟩ for this session was specified with the interaction model, as we wanted to capture relevant events by recording the contents of the computer screen during the session. Finally, there was a feedback session, taking up to 5 min, to collect the children's subjective opinions about using the language they were assigned to, by answering to the {Satisfaction Questionnaire}, which was modelled by the survey modelling activity, but dependent on the specification of the interaction model.

While preparing this evaluation, from the teaching materials and the events analysis, we were also creating the documentation of Visualino and its application scenarios that can be shipped along with the software product, as they were reviewed and improved for the purposes of the experiment. This material, referring to a specific scenario can be documented within the CM. Further, we illustrate how the test models are designed using the interaction modelling and the survey modelling activities.

5.2.4. Interaction model instantiation

In this section, we instantiate the interaction test model (see Fig. 29), which is used during the competition session of evaluation. This model encapsulates the ⟨⟨interaction syntax⟩⟩ from both, Visualino and Lego. Further, it is specified for the ⟨⟨task⟩⟩ which is associated with the concrete scenario from the evaluation context, named {Challenge}. A correct solution of the problem was provided by the Artica developers for Visualino, and by a language engineer who had experience with Lego.

By analysing the given task and the evaluation model, three events are defined to be captured. To measure ⟨⟨usability requirement⟩⟩ {Effectiveness}, described as correctness of solving ⟨Use Case:Challenge⟩, [Event1] and [Event3] are specified. [Event1] captures Success (S) or Fail (F) of modelling the following concepts: Bumper, First Turn Left, Second Turn Left, First Turn Right, and Second Turn Right. It is captured manually observing {Screen Record}, e.g. videos of the participant's interaction with the language while trying to provide a correct solution. Further, [Event3] was marked with Success (S) if the robot successfully performed the challenge in the arena; Fail (F) if the team tried until the time limit, but did not succeed to program the robot correctly; and Quit (Q), if the team gave up. Finally, in the [Event3], we assessed additional useful information during the video analysis, like if the team: (i) saved or reused the previous exercises; (ii) experienced technical problems or functional errors; (iii) had interaction difficulties (e.g. using copy/paste for visual objects or connecting concepts in sequence); (iv) reused previously constructed sequences (within the same exercise); or (v) used any other additional language features (e.g. zooming). [Event3] registers the time of occurrence of certain ⟨captureEvents⟩, measured from the beginning of the competition session and can serve as a source for ideas on improvement of certain language features for language engineers.

The interaction results are analysed by the expert evaluator, who prepared the data collection sheets and has put forward the analysis. The correlation analysis was supported by SPSS tool [104], and the graphs were generated in Excel.

5.2.5. Survey model instantiation

By performing survey modelling, we instantiated the test models (see Fig. 30) for the background questionnaire used during the learning session, and for the feedback questionnaire used during the feedback session. The survey forms were composed of Smileyometers, which are found to be appropriate for children questionnaires [105]. While answering to forms, children were assisted by an adult (one of the experiment assistants). This was done to ensure that there were no misinterpretations of questions and answers, and to confirm that participants did not experience reading problems. As participants were grouped into teams, the participants' individual answers to questionnaires were merged, and the mean rate was computed within each team, for each answer.

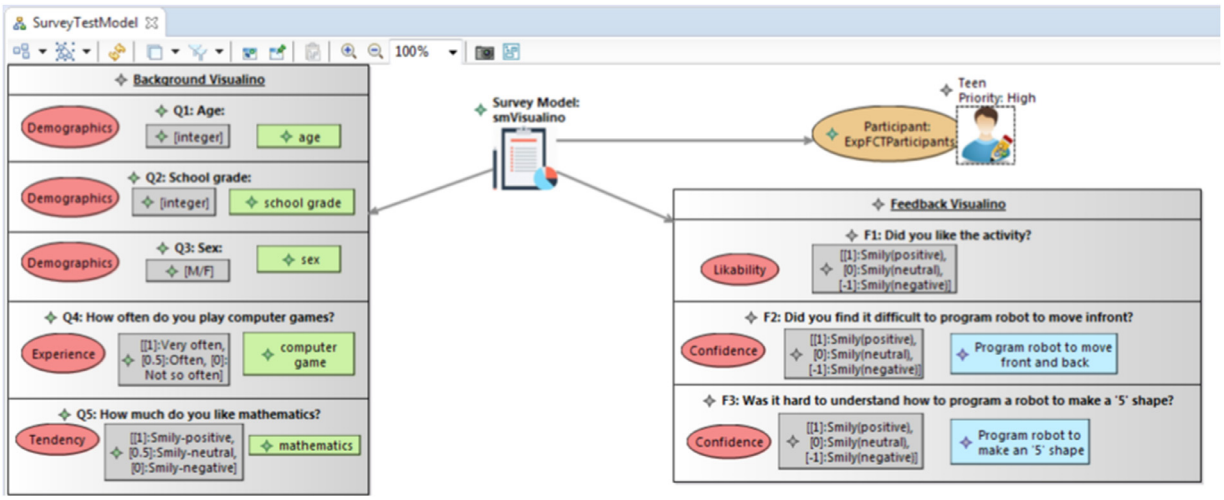

Fig. 30. Survey test model.

The background questionnaire [BackgroundVisualino] is related with the {Teen} ⟨⟨profile⟩⟩. Each question in this questionnaire is related to one of the logical expressions associated with evaluated profile in CM. For instance, for [Q1:Age], the ⟨⟨logical expression⟩⟩ {age} was associated. In this case, the age is defined as an actual value ranging from 13–19. As so, the expected answer is defined as an {integer} ranging from 13 to 19. If another value occurs, the participant's data is discarded from further analysis. The collected data includes ⟨⟨indicators⟩⟩ defined as {Demographics}. These help to verify if there are other factors, such as age or gender, influencing the results. On the other hand, certain ⟨⟨indicators⟩⟩ like /Experience/ were calculated as a merged value of this group of questions to serve for further impact analysis. Also, the abstract scale, for instance of ⟨⟨logical expression⟩⟩ {Computer game} was instantiated as 3-point Likert scale {[1]: Very often, [0.5] Often, [0] Not so often} with concrete weights which are used for later calculations.

The feedback survey was designed to measure the ⟨⟨requirement⟩⟩ {Satisfaction}, which was characterised by the following ⟨⟨indicators⟩⟩: {Confidence}, reflecting how confident children were about their solution, {Likability}, reflecting how interesting and enjoyable they found the challenge itself; and {Learnability}, reflecting how useful they found what was taught during the learning session to help them facing the final challenge. Each question can be related to a concrete ⟨⟨scenario⟩⟩ included in EM. For instance, the question [F2: Did you find it difficult to program the robot to move in front?] indicates {Confidence} of the participant regarding the learned ⟨⟨scenario⟩⟩ {Program robot to move front and back}. The ⟨⟨scale⟩⟩ for all satisfaction questions reflects positive or negative experiences and is defined as {[1]: Smiley[positive], [0]: Smiley[neutral], [−1]: Smiley[negative]}.

5.2.6. Result model instantiation

To evaluate the usability of {Visualino} and {Lego} we reused techniques of UI evaluation that imply the involvement of real users as subjects of controlled experiments. A comparison criterion was based on the correctness of the problem solution, the time to solve it and personal preference. To get correct interpretations of our results, it was mandatory to profile the users. The criteria that were observed were their age, gender, and programming background. Further, we collected the participant's feedback and success rate regarding their experience with a language. During the report modelling activity, we declared all of these results in a Report Model [expo2015result] (see Fig. 31). The results specified by the interaction and survey models are documented for each event or questionnaire. They are correlated to evaluation results, whose objective was to analyse the {Effectiveness} and {Satisfaction} ⟨⟨requirements⟩⟩ for each language. The statistical analysis documents are assigned as ⟨⟨outsideRef⟩⟩ properties.

The observed evaluation results of the second iteration of the design of the DSL showed convergence and highlighted new possible improvements of Visualino. The children subjects were still having problems in making a right solution, having a small {Effectiveness} score of 0.37 for Visualino. The subjects were expressing likability toward Lego, while toward a Visualino they were indifferent. Based on these results, we created the recommended GM [Recommendations] consisting of new requirements impacting the usability goal for [U1] and functional goal [F1].

5.3. Pilot evaluation study

In this section, we report on the pilot assessment of the feasibility of the USE-ME tool with master students in computer science which were involved in a DSL course. In total, four groups were consisting of two or three participants which were developing the following DSLs:

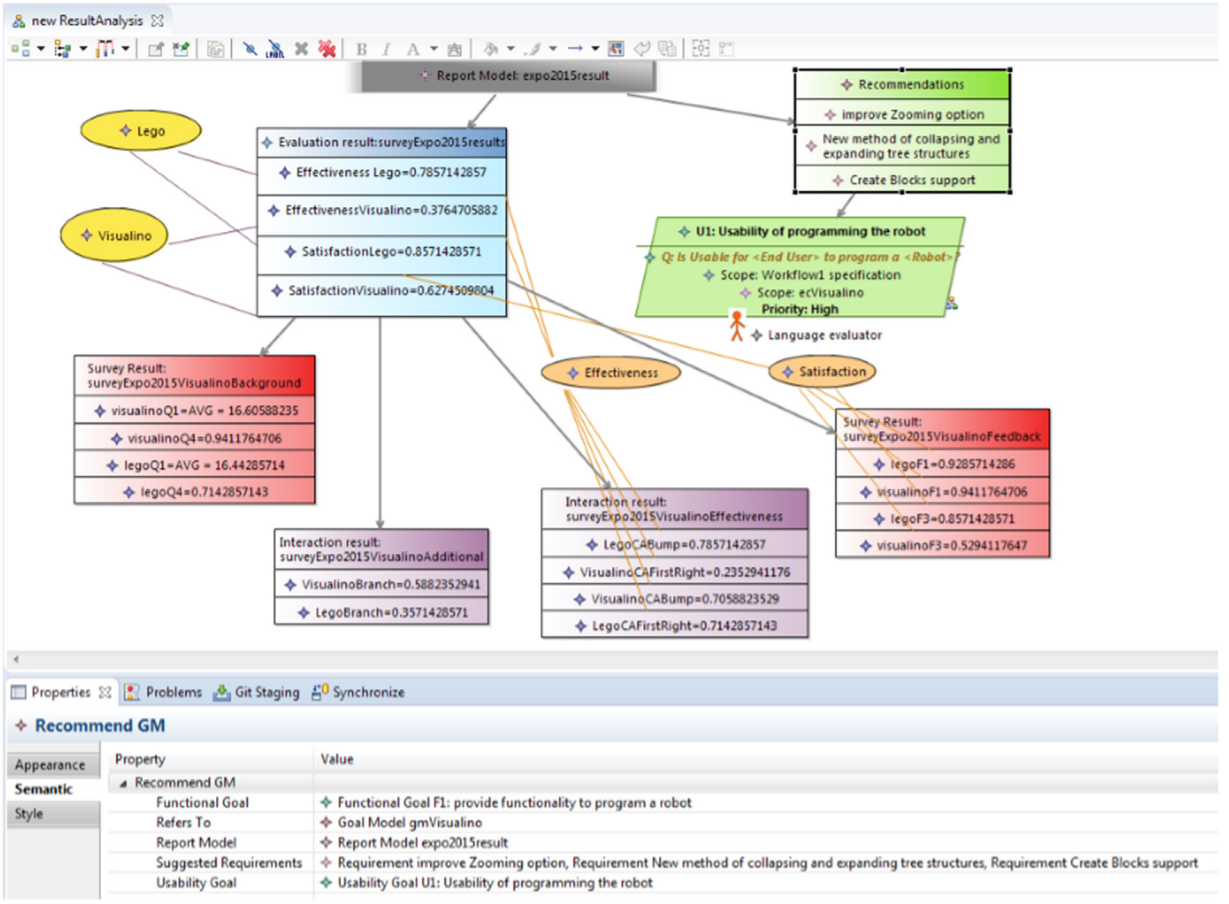

Fig. 31. Report model.

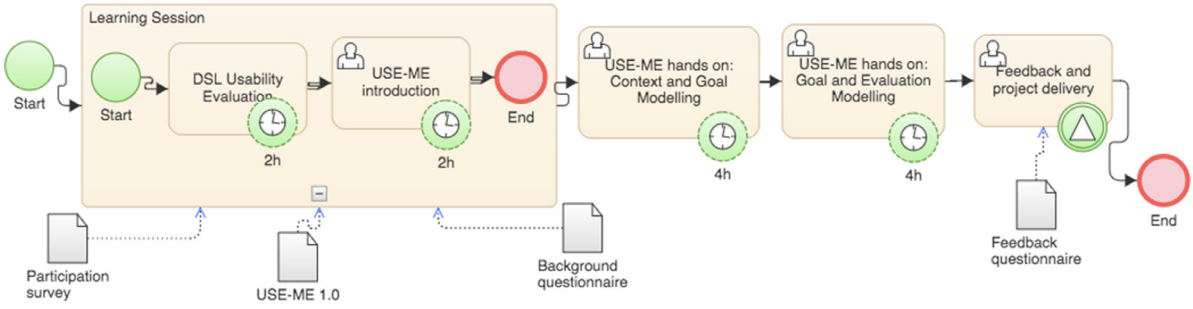

Fig. 32. Pilot session process model.

- *DSL spreadsheets* – a DSL that transforms an activity graph into a Gantt chart. The target users of this DSL are project managers.
- *Gestures kinect* – a DSL which supports specification of communication between navy users using a Kinect device. The target users of this DSL are navy operators.
- *Peddy paper* – a DSL which creates several instances of personalised rally paper scripts. The target users are rally paper organisers.
- *Smart house* – a DSL which supports a design of the elements and operations for a smart home. It is meant to be used by house owners.

The evaluation sessions were prepared as shown in Fig. 32. The first learning session took place after four weeks of the DSL development. Students were introduced to the usability evaluation during a 2 h theoretical lecture. For the next two hours, the students received a tutorial on the USE-ME tool and were guided to perform installation and set up working

Table 1
Validation table of USE-ME models produced for each DSL.

DSL	Spread sheets	Gestures kinect	Peddy paper	Smart house	AVG
Participation time	6h	12h	12h	12h	
Context modeling	***0.36***	***0.72***	***0.56***	***0.76***	**0.6**
User hierarchy	0.6	0.8	0.8	1	0.8
User profileTemplate	0.4	0.6	0.4	0.6	0.5
Environment context	0.2	0.8	0.6	0.8	0.6
Workflows	0.2	0.8	0.2	0.8	0.5
Scenarios	0.4	0.6	0.8	0.6	0.6
Goal modeling	***0***	***1***	***0.4***	***0.5***	**0.475**
Usability goal model	0	1	0.4	0.8	0.55
Usability requirements	0	1	0.4	0.2	0.4
Evaluation modeling	***0.2***	***0.6***	***0.7***	***0.7***	**0.55**
Evaluation objectives	0.2	0.8	0.6	0.6	0.55
Evaluation instantiation	0.2	0.4	0.8	0.6	0.5
Interaction test model	0.2	0.4	0.8	0.8	0.55
Survey test model	0.2	0.8	0.6	0.8	0.6

environment. Also, the students were given a participation questionnaire to fill in and describe a purpose of their DSL. At the end of the session, they were given the background questionnaire to fill in. The following two weekly four-hour sessions consisted of USE-ME hands on labs. The students were introduced to the modelling activities followed by Visualino example. For each activity, the same student from the group was using the tool to create USE-ME models for their DSL, while other students from the group were helping in deciding what would be a correct specification. Finally, students were asked to try to finish the learned models and deliver them as a part of the DSL course. After delivery, the students who were using the tool to model were asked to fill in a feedback questionnaire.

The evaluation deliveries (project reports, USE-ME models), background and feedback questionnaire results and evaluation results can be found at [106]. Here we shortly describe the obtained results:

The participants were master students with intermediate knowledge of modelling techniques, namely having a background knowledge of UML (class diagrams, use cases, activity diagrams and interaction diagrams). Also, participants knowledge related to the working environment was advanced. However, the participants had little or no knowledge regarding HCI, especially empirical experiments or the usability testing. As the objectives of the USE-ME framework are to support Language Engineers in specifying the usability evaluations, as the Expert Evaluators are often found to be too expensive to be included, we find that participants are adequate surrogates for the target end user of {Language Engineer} profile which is to be supported by USE-ME approach.

Table 1 presents the results of the USE-ME model validation for each reported DSL. We have analysed all diagrams which should be provided for Context Modelling, Goal Modelling and Evaluation Modelling activities. The diagrams were graded regarding their completeness and correctness as follows: (0) – no delivered model, (0.2) – very low, (0.4) – low , (0.6) – satisfactory, (0.8) – good, (1) – very good. We can note that all groups expect the Spreadsheet DSL manage at least satisfactory specify their models. However, Spreadsheet group participated only for a two hours of a hands-on session and this impacted the low score of their reports. In each case, we can observe that delivered models were satisfactory and by this indicating the understandability of the approach in specifying the usability evaluation within a small amount of time. However, it seems the biggest challenge for participants was to figure out how to specify usability goal model and associated requirements. In its current version, the prototype does not yet support the usability catalogue. We believe that this extension will help users without much knowledge about requirement engineering and quality evaluation to define properly usability objectives for their case.

The participants from the groups which were having a bit higher understanding of HCI were naturally selected to be responsible for modelling with the USE-ME framework. That is why just these participants were asked to provide a feedback about the tool. Participants authorised the publication of project results to Zenodo library. Although they found usability evaluations necessary for a DSL deployment in practice, they did not find the modelling activity interesting. One of the main reason was that the modelled evaluation was not to be executed, making it less interesting for their project purpose. The Gesture Kinect group found that it was easy to understand the USE-ME approach. The Peddy Paper group found it on another hand not to be so easy, pointing out there were too many steps to follow. The Smart House and Spreadsheets groups found approach more or less easy, pointing the lack of a guided cycle and highlighting a usefulness of the Visualino example for understanding an approach. All of them reported to be able to discover environmental elements which they did not consider before in development, and found easy to create a user hierarchy and more or less easy to specify other context model elements. However, they found the modelling of usability goals they found a bit harder but still manageable. Two groups found useful to reuse context model elements in goal specifications, or during the evaluation modelling. They found it to be easy to creating test model specifications for evaluation. In general, they did not feel very confident while using a USE-ME tool, however, they find a tool more or less expressive enough for purpose of specifying usability evaluation, and also suitable for another kind of software products. However, none of the participants is familiar with any other tool which supports usability evaluation.

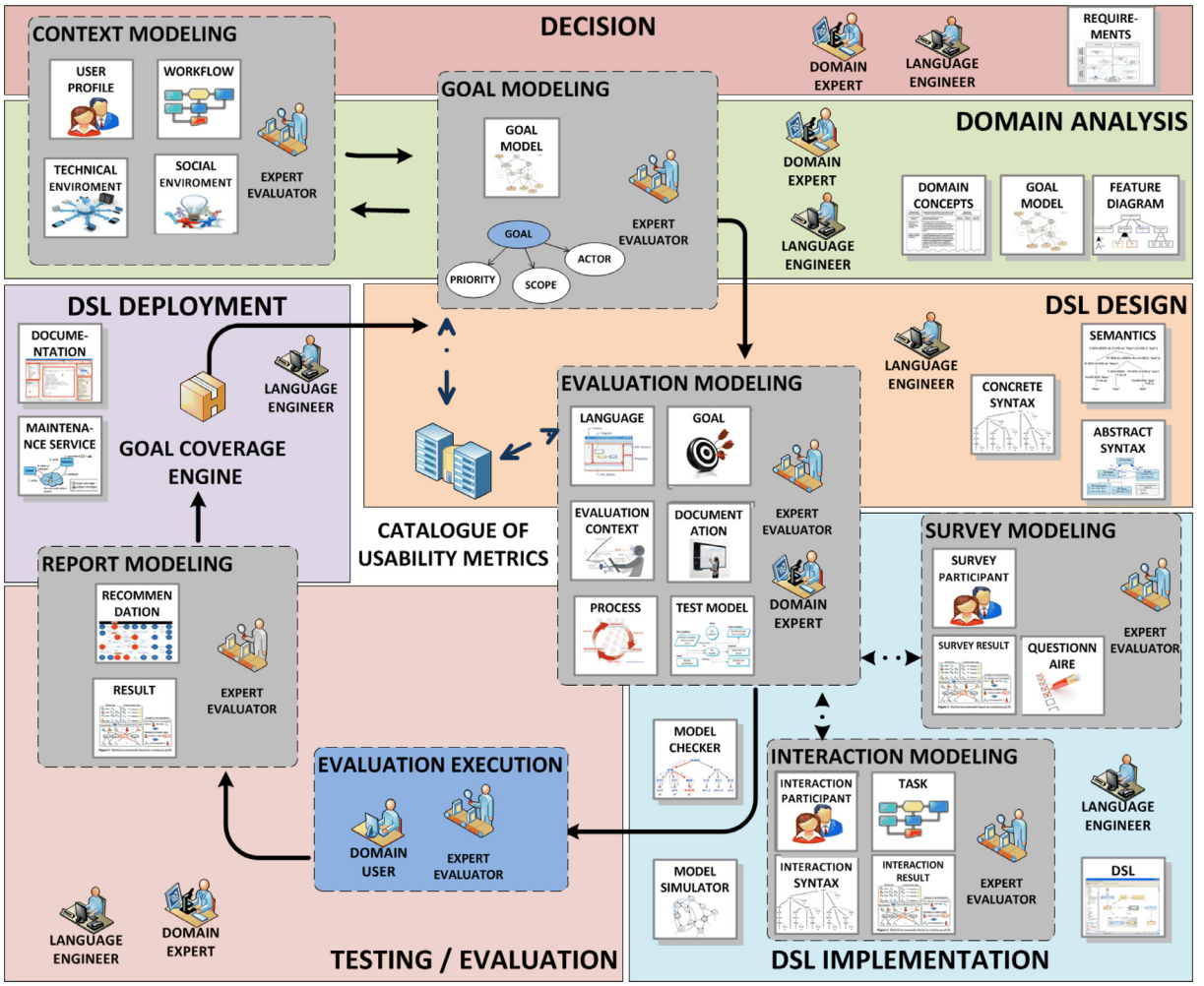

Fig. 33. Integration of USE-ME approach with DSL development phases.

6. Applicability of USE-ME approach

In this section, we discuss a correlation of the USE-ME process we proposed in Section 4 with a DSL development cycle presented in Section 2.1.2.

6.1. Integration of USE-ME approach with DSL development phases

In Fig. 33 we present an 'idealistic' integrated development process where one cycle of USE-ME process is performed during one development iteration. We correlate the USE-ME activities with each DSL development phase. During the Domain Analysis and Decision phase of the DSL development, we perform the Context Modelling and Goal Modelling activity. Here we can share information which is being specified and discussed within the context of DSL artefacts from this phase. For instance, the feature diagram can bring the information about environmental elements of the Context Model and the Goal Model itself can serve to discover usability goals and requirements, and correlate them with the existing goal model. Further, during the DSL Design phase Goal Modelling should be concluded and the Evaluation Modelling phase can start. The evaluation is dependent on the scope of the DSL artefacts which are designed and are to be implemented in the current iteration. For instance, already early on, it is possible to define the objective of the study and specify its process and context or to discuss if the comparative evaluation should take place and which are alternatives to be considered. During the implementation phase, test models are created for the evaluation trough Survey Modelling and Interaction Modelling activities. Further, in the testing/evaluation phase, the evaluation is executed. However, often its execution can depend on the success of the tests. Usability goal measures can depend on the provided functionalities of the DSLs. If these functionalities are tested in this phase, the evaluation process should reflect the dependencies on the tests which should be performed before its execution. Finally, the Report Modelling activity can be performed and, if necessary, extended to the DSL deployment

phase. Finally, the Recommendation Model is used to extend/change the previous Goal Model and to make a decision about the next development cycle.

However, sometimes the complete USE-ME cycle can be applied during a single DSL development phase. For instance, during the Domain Analysis phase in which the requirements and objectives are not clear, we can model the survey which will help to clarify the objectives from the different stakeholders. In this case, the resulting goal or context model is formally justified. On the other hand, during the Design phase, we might want to experiment with different syntax designs, and it is useful to create an assessment to evaluate their readability and understandability. Further, during an implementation phase, the various prototypes can be implemented and evaluated in order to continue with final implementation. Finally, during the testing/evaluation phase, we might want to perform several evaluation executions which are interdependent, e.g. previous ones impacting the extension of the CM and GM.

6.2. Application of USE-ME to incremental iterative DSL development

The USE-ME framework reuses the specifications along different iterations so that it is not necessary to start again from scratch with the analysis and design in each iteration. When introducing small extensions or changes to the DSLs' syntax or semantics, it will be necessary to extend the existing USE-ME models with 'new' context instances (e.g. new/extended user type, environment or user story). The updated evaluation goals should take into consideration the new features and/or expected usability improvements over a previous version of the DSL. It is useful to perform the comparative evaluation with a previous version of the DSL, to observe if an improvement happened. For instance, in the case of Visualino (see Section 3.3.2), we reused the experimental design, evaluation process and their instruments (surveys and an interaction test model) from the previous iteration. This gave us means to comparatively analyse the results from the first and second iteration. To support the organisation and planning of the iterative development, and increase the transparency of the development tasks, we can benefit from existing management tools support used for agile approaches (for instance we applied Scrum and Pivotal Tracker during the FlowSL case study (see Section 3.3.1).

To trace changes in a DSL, we can benefit from existing incremental language development approaches such as LISA [107]. In this work, when the language is extended with new features, this corresponds to a new language containing specifications of the change together with the description of the previous language. For example, Mernik and Zumer [107] extend (in the sense of object-oriented extension) a simple language describing a robot movement with a new language which also calculates when the robot will reach the final position. For performing this extension, there should be a concrete motivation behind. For instance, in this case, new scenarios should be added to the previous workflow in a Context Model. This change does not imply the addition of the new Usability goal, but rather an extension of the existing ones with a new scenario. The evaluation design can be kept and applied to a refined scenario. On another case, also taken from [107], the authors perform an extension which supports the robot for cleaning. This change refines the previous end user profile (the user wanted 'to use a robot'), to a new profile where the end user wants 'to use a robot *for cleaning*'. Also, the environment context is specified more in depth reflecting the robots for cleaning and environment where they are used. Finally, the new user stories may have different actors and will be documented under the new workflow. The evaluation should redefine the usability goals, and probably create new ones which take into consideration the new context.

Early evaluations and extensions without a significant change of context or usability goals, can be performed with domain experts, while other users should be involved in evaluations later on. By performing early assessments with more available users the evaluation design and its metrics are being validated, and a risk of not obtaining a return of the investment in extensive evaluations is lowered. Also, as mentioned previously, bigger investment into usability evaluations should be applied to the DSLs which target a wide scope of the end users, especially those who are not exactly reflected by the domain experts profile which is included in the development. For the 'in house', or small DSLs developed for a very small set of users, especially those included in its development process, an application of this approach as a whole might be too expensive, but the analysis procedure can still be found useful.

6.3. Applicability of the approach outside of the scope of model-based DSLs under development

6.3.1. Previously released DSLs

We applied the USE-ME approach on two industrial DSLs. In both case studies, we were not directly involved in the development of the DSLs but participated as evaluation experts. These DSLs were developed from scratch: an internal DSL based on Ruby [103]; an external DSL described in Section 5.2. We chose these case studies as we did not find many examples of usability evaluations involving end users early in the DSLs' development cycle (see Section 7). If we think of DSLs already existing in the market, the USE-ME process will be similar to the process used with DSLs which are under development. However, with previously released DSLs, it should already be clear *Who* are the users, *What* are the user stories and *Where* the DSL is being used. It is expected that it will be possible to reuse most of this knowledge from already existing artefacts and specify the usability goals and metrics in an easier fashion. During the evaluation modelling, the assessments can be planned to be performed automatically (e.g. with automated usage data collection) and remotely. With a large number of regular users, more data can be obtained.

On the other hand, if there is still information missing to create a complete context or goal model, it is still possible to plan the evaluation, by sticking to the information that exists. For instance, if there is no formal specification/documentation

of the workflows, this information can be omitted. However, the evaluation model will be designed accordingly, e.g. to evaluate the satisfaction with a language or the readability of design concepts. It is also possible to create an interaction model which captures just 'random' roll-back cycles (semantic errors) during a common use of a product (e.g. placing and deleting certain commands repetitively in a certain sequence can help identifying misinterpretation issues of a concrete syntax element).

6.3.2. Grammar-based DSLs

Both of our case studies were developed by using metamodelling tools (MetaEdit and Eclipse EMF, respectively). Our process is not restricted to model-based DSLs and could, in principle, be applicable also to grammar-based DSLs or even to GPLs. However, this would require adapting our prototype to cope with their architecture. This would be feasible but is beyond the scope of this paper. It would be necessary to apply modelling techniques (e.g. reverse engineering), to create certain artefacts (e.g. goal model, UML diagrams, etc.).

6.4. The users of the USE-ME approach – taking the role of expert evaluator during the DSL development cycle

The modelling activities in our framework are presented as activities of an Expert Evaluator which, in practice, is not typically included in the DSL development process. We mentioned in Section 2.2 that this HCI expert usually implements complex experimental usability evaluation studies. The HCI expert profile includes comprehension of user profiling, experimental approaches and usability evaluation methods. However, when applying the proposed approach, these kinds of experts may lack the knowledge about model-driven methods and requirements engineering, and it may be not trivial for them to grasp the modelling concepts and tools. Therefore, these experts should be introduced to abstraction modelling and trained to use the modelling support with a focus on mastering the modelling concepts present at the USE-ME approach (e.g. goal modelling, process modelling, UML modelling).

Moreover, we found that in case there is no person included in the development team with evaluation expertise, the knowledge of these experts can be transmitted to a typical DSL Engineer through the USE-ME approach. The approach was validated by researchers from NOVA-LINCS research centre, who are experts in MDD and DSL development and were not involved in the framework development. Some of them are also knowledgeable about the experimental software engineering and requirements engineering. These experts provided valuable feedback and improvement suggestions over the framework. However, people with high level of experience might also not be available during the DSL development process to take the role of Expert Evaluators. Inexperienced engineers can still be suitable for the Expert Evaluator role in the DSL development project. As a proof of concept, we performed a preliminary pilot evaluation of the framework prototype with master students in informatics who had knowledge about MDD and DSL development [106]. However, these engineers should be trained in user profiling, usability evaluation methods, experimental software engineering and requirements engineering practices.

7. Related work

The related studies show that the scientific community is still not making use of the current solutions offered by the software engineering community [2]. The requirements discovery process is complicated because DSLs are exploring new domain-related problems. Therefore, in most of the cases requirements cannot be known in advance; they change as knowledge changes [108]. These constraints affect the applicability of many of the traditional software engineering practices and partially explain why scientists tend not to use them [109].

7.1. General purpose usability evaluation approaches

In the context of GPLs, comparing the productivity impact of using different languages during the software development process has some tradition [110]. Some of the common techniques are the use of a popularity index [111], the cognitive dimensions framework [19], or heuristics based on the studies of cognitive effectiveness for visual syntax [16]. These methods can be reused when these techniques are identified as relevant for the usability of a DSL (e.g. Moody's work on cognitive effectiveness can be reused if the visual concrete syntax is given to the target DSL). When usability problems are identified too late, a common approach to mitigate them is to build a tool support that minimises their effect on the user's productivity [112].

GPL users are typically technically-oriented programmers, with good understanding, skills, and experience in computer science and technology. However, the understanding of domain concepts is also necessary to develop programming solutions. On the other hand, DSLs are meant to reduce the use of computation concepts by putting the focus on the domain concepts. As such, DSLs are expected to be used by a diverse target population (e.g. experts from physics, chemistry, finance, management, etc.). The evaluation criteria for DSLs need to be appropriate for their target users, as well as, technical, social and physical environment.

We can build on existing methodologies and tools to assess the usability of UIs [13] and adopt them for programming languages as we discussed in Section 2.2. Existing UI practices, due to the wide spectrum of the context of use that they target, have a low level of external validity, making it hard to interpret what the collected information means. DSLs are built for a more confined context of use, and they capture one particular set of domain concepts. When evaluating them, the population of users is smaller, and the external validity of the result is expected to be higher.

7.2. The state of practice in DSL evaluation

Some studies address specific quality characteristics for DSLs by performing the experimental evaluation after implementing the language (see Sec. 2.1.2). Haugen and Mohagheghi [113] present a structured questionnaire based on three dimensions of a DSL; expressiveness, transparency, and formalisation. Merilinna and Parssinen [114] investigate the benefits of using DSLs by making comparisons between the DSL approach and traditional methods experimentally. Kosar et al. [115] independently evaluated several DSLs and are mostly concerned with program comprehension, correctness and efficiency while using the DSLs when compared with using GPLs. A detailed analysis of their data could be used to identify opportunities for improving the tested DSLs. Kieburtz et al.'s [23] experiment address DSL comparative evaluation as part of the concern with flexibility. Murray's experiment [25] explicitly looks for opportunities for improving the respective DSLs under scrutiny by taking learnability, understandability, usability, user satisfaction and language evolution as improvement goals. Kärnä et al. [116] evaluate the DSL solution in an industrial case focusing on the productivity and usability. They first determine the objectives for the creation of the DSL and then collect data via controlled laboratory studies. In general, existing assessments are performed with a final version of a DSL when potential problems are expensive to fix. Our research aims to introduce language evaluation concerns early in the DSL development process so that problems can be found 'on-time' and fixed at a fraction of the cost it would take to fix them if detected only after the implementation.

Gabriel et al. [12] present a systematic review emphasising the reduced concern on the evaluation of DSLs. This work highlights the state of practice and increases awareness of the shortcomings of research regarding this problem. Kosar et al. [11] performed a systematic mapping study of DSLs in the period from 2006 till 2013. They concluded that 'DSL community focuses more on the development of new techniques/methods rather than investigating the integrations of DSLs with other software engineering processes or measuring the effectiveness of DSL approaches'. This work explicitly presents a clear lack of evaluation research, in particular, controlled experiments and a need for integration and measuring effectiveness. Also, it is pointed that DSLs had rarely been validated (e.g., by end-users) assuming that the developed DSLs were perfectly tailored for domains, as well as fitting end-users requirements. However, this is far from true. DSL under development should have been validated by empirical studies, the involvement of end-users, or by the psychology of programming research.

Some authors, however, did tackle above mentioned problems. Kolovos et al. [98] list the core quality requirements for a DSL. Hermans et al. [37] identify success factors for designing DSLs by performing an empirical study. Wu et al. [117] present an approach to determine the effort while using DSLs during application development, contributing to the classification of the effort and proposing of related metrics. Kelly and Pohjonen [6] discuss worst practices for creating DSLs which developers should avoid, based on industrial case studies. McKean and Sprinkle [118] present criteria that will help in selecting a DSL or any other approach to be used in system development. Nishino [119] was to solve the usability problems of DSLs caused by a bad design. He proposes to identify usability problems by analysing a set of cognitive dimensions proposed by Green and Petre [19]. However, their approach does not cover how to proceed to apply changes in a potential existing analysis model, how to evolve the design models or the implementation infrastructure. This approach addresses only partially the testing of usability problems by usability expert, and it doesn't include end-users into the evaluation. Kahraman and Bilgen [20] propose a Framework for Qualitative Assessment of DSLs that adapts and integrates the ISO/IEC 25010 standard, maturity level evaluation approach and the scaling approach into a perspective-based model. This framework supports the choice of quality goals from the different stakeholders' perspective but does not to include the Domain User concerns. Finally, Häser et al. [120] provides an integrated end-to-end tool environment to perform controlled experiments in DSL engineering. The environment supports language design and steps of experimentation, i.e., planning, operation, analysis and interpretation, as well as presentation and package. Controlled experiments have the potential to provide appropriate, data-driven decision support for language engineers to compare different language features with evidence-based feedback. However, this kind of usability assessments is not always cost effective in early phases of language development.

7.3. Applying UCD methodology into design of visual languages

Assessments of the usability of a visual language recently are getting more often seen in practice by successfully involving the UCD methodology (see Section 2.2.2) into a design of visual DSLs. Neumann et al. [121] investigates rule-based, end-user strategy programming by introducing a domain-specific, end-user programming environment to allow football coaches to create animated football scenarios and by associating strategy information with virtual football players. This study contributes further by putting into evidence the usefulness of Natural Programming design process, which applies principles of UCD [122].

Aghee and Pautasso [123] iteratively evaluated a DSL for developing of Mashups using formative evaluations. Angelini et al. [124] presents an innovative visualisation environment, which eases and makes more effective the experimental assessment process. Ardito et al. [125] offers an approach where a composition platform enables the extraction of content from heterogeneous services and its integration. Bauelo et al. [126] presents a visual query system that has been designed and implemented following the UCD approach. Danado and Patern, [127] creates a visual mobile end user development framework which allows end users without programming background to create, modify and execute applications, and provides support for interaction with smart devices, phone functions, and web services. Fogli and Provenza [128] describes a meta-design approach to transfer the development of government-to-citizen services from professional software developers to administrative employees, without forcing employees to acquire any programming skills.

We find that these examples are valuable, although they lack systematic experimental assessment in most of the cases. However, evaluation practice discovered in these works could be made model-driven, systematic and extended to DSLs in general.

7.4. Collaborative attempts of introducing an end-user into DSL development

Finally, we present attempts to solve the usability problems by taking into account end-users needs during DSL development. However, these approaches are in most of the cases just applicable in the domain analysis and language design phase, namely supporting collaborative development of the abstract and concrete syntax with end-users. However, these examples do not adequately represent the complete domain model explicitly (scope, terminology, concepts, and commonalities and visibilities). Also, the profile of the users which are included, or knowledge sets which are necessary, are primarily domain experts.

Perez et al. [129] claims that end-users are taken into account because a visual DSL is selected by applying best practices of end-user development during the design of the concrete syntax, but without a systematic evaluation or customisation support. In fact, the proposal always creates visual DSLs to improve usability, which may not be the best approach for some end-users. The goal of Wuest et al. [130] was to facilitate the meta-modelling activity to non-experts by creating FlexiSketch, an environment for modellers and end-users to design together with the examples of the domain using sketches. These examples are used for the creation of the DSL syntax, both the abstract syntax and the concrete syntax. Similarly, Cho et al. [131], provide a friendly solution for end-users to describe domain examples, the creation of the DSL syntax, and the semantic restrictions. Kuhrman et al. [132] want to bring together DSL developers and domain experts when developing DSLs in complex application domains. This work provides a DSL, named 'PDE language', which provides a visualisation model editor with different views to facilitate the participation of domain experts in design of concrete syntax. Sanchez-Cuadrado et al. [133] wanted to support the use of informal drawing tools as a friendly interface that facilitates the meta-modelling task. The difference of this work in respect to the others is the approach used to generate the meta-model from the domain examples sketches. This meta-model is obtained iteratively, one example at a time, in which developers can assess the evolution of the meta-model and the specific effects of each domain example over the meta-model. If some changes have been applied, a procedure checks for possible mismatches between the final meta-model and the domain examples. Nevertheless, none of the above approaches makes explicit the domain model details, nor addresses evaluation activities of DSL development.

The motivation of Izquierdo et al. [31,134,135] was to highlight the importance of the end-users role in the definition of DSLs and provide means to enable the collaboration between end-users and developers in the context of DSL development. With this aim, they propose a community-driven DSL Collaboro, to encourage end-user participation in the definition of DSLs. Collaboro describes the elements of the collaborative activity (comment, vote, solution, etc.) and the elements of the abstract and concrete syntax of a DSL (entity, attribute, relationship, textual notation, etc.). These elements are used to track the evolution of both the abstract and concrete syntax. The collaborative development starts after requirements gathering when developers design a preliminary abstract and concrete syntax. End-users can comment, propose solutions, and vote other participants opinions and proposals. This interaction continues until the syntax after all the changes proposed and applied. In summary, the main contribution of this work as a whole, in contrast to previous works, is the use of friendly mechanisms, such as the use of an informal panel and the use of examples, as a way to reason about the domain and the abstract and concrete syntax. Villanova et al. [136] made a significant contribution in introducing agile methodologies in DSL development. Methods like customer reviews and demonstrations that are used in Scrum are not often validated by real end-users, but mostly with managers or 'buyers' of the software. The practice of introducing a definition of *Done* brought by the agile practices promotes demonstration to the customers and collecting feedback. However, demonstration and questionnaire are not sufficient to test product usability with the customers. Indeed, without using a product, end users will have a hard time identifying where they are likely to make mistakes and use the product inappropriately. While they can provide some feedback on their satisfaction regarding a language construction, they will not be able to give feedback on their efficiency and effectiveness while using it.

The mentioned works do not address how to prioritise user recommendations, for the next iterations of the DSL development. We could leverage user profile to support prioritisation of requirements for the next iterations, which would lead to more timely and usable improvements in the DSL. This means providing the set of the most needed functionalities in the application area that are possible to be developed in the following cycle when considering the technical restrictions (e.g. lack of the informational database to support the functionality, need to develop previous modules to support the functionality). The domains for which DSLs are developed are constantly changing, so prioritising evaluation can help providing timely solutions that are usable by end-users. However, the specific requirements and needs of the target end-users are not evaluated by including them in systematic evaluations during the DSL development process.

8. Conclusions and future work

This paper presents and evaluates a framework, called USE-ME, to capture essential concepts relevant for usability evaluation during an iterative and incremental DSL development life-cycle. It aims to cope with the need for systematic approaches for usability evaluations of DLSs, including supporting empirical studies and controlled experiments with end users, that

takes into consideration the several development stages. USE-ME covers the whole DSL development process, contrasting with other related approaches which tend to focus on domain analysis and design. Further, they tend to use domain experts as subjects, while we involve the actual end-users in our evaluations.

This work identifies all the mandatory concepts and activities and aggregates them into a formal meta-model. With this approach, we highlight the complexity of the information that should be traced to streamline and automate (as much as possible) the user-centred development process. The framework supports internal and external quality (e.g. syntactic and semantic models) and quality in use (e.g. interaction and survey models). Our approach helps the Language Engineers to explicitly model the evaluation process. This procedure allows the DSL developers to monitor the impact of language evolution to the efficiency and effectiveness of practitioners using the language (and its companion toolset). Additionally, it supports specifying experimental assessments and tracing the impact of usability improvements since an early stage of development of DSLs. In the long run, this approach is expected to improve the productivity of the DSL developers and the language users.

The methodological contributions are supported by a modelling tool. A prototype tool was integrated with an existing Eclipse-based IDE to support the development of DSLs. This prototype enables language engineers to take the role of Expert Evaluators and to prepare the evaluation models. Also, it supports tracing the evaluation goals, reusing experiment designs and providing better documentation and reasoning.

This approach, besides documenting the evaluation process, enables reasoning about the models and setting a clear workflow to its users. As part of the future work, we are improving the tool support to guide the users through the modelling process and validate the models. It is being implemented as a part of Goal Coverage engine, which will contain specification and verification of different syntactic and semantic rules depending in which step of a USE-ME process is a user. This engine will also support verification of rules regarded to transformations between the models in different development stages. Finally, the USE-ME engine will be extended with a tool for (semi)automatically determine the success coverage of the specified usability goals by the DSL developer.

Finally, another follow-up to the presented work is to study the integration of USE-ME with already existing solutions and modelling workbenches for the development of the DSLs. We believe that this contribution is a step towards the automation of the DSL development process tackling a specific concern that is typically neglected, Quality in use.

Acknowledgements

The authors would like to thank FCT/MEC NOVA LINCS; PEst UID/ CEC/04516/ 2013 and DSML4MA TUBITAK/0008/2014 Projects, as well as to COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) for the partial support to this work.

References

[1] Mernik M, Heering J, Sloane AM. When and how to develop domain-specific languages. ACM Comput Surv 2005;vol. 37:316–44.
[2] Kelly S, Tolvanen J-P. Domain-specific modeling: enabling full code generation. Wiley-Interscience; 2008. ISBN 0470249250
[3] Azadi Marand E, Azadi Marand E, Challenger M. DSML4CP: a domain-specific modeling language for concurrent programming. Comput Lang Syst Struct 2015;vol. 44:319–41. doi:10.1016/j.cl.2015.09.002.
[4] Völter M, Dietrich C, Engelmann B, Helander M, Kats L, Visser E, et al. DSL engineering: designing, implementing and using domain-specific languages. CreateSpace Independent Publishing Platform; 2013a. ISBN 978–1481218580.
[5] Gray J, Fisher K, Consel C, Karsai G, Mernik M, Tolvanen J-P. DSLs: the good, the bad, and the ugly. In: Proceedings of the companion to the 23rd ACM SIGPLAN conference on object-oriented programming systems languages and applications. ACM; 2008. p. 791–4.
[6] Kelly S, Pohjonen R. Worst practices for domain-specific modeling. IEEE Soft 2009;vol. 26:22–9.
[7] Völter M. Best practices for DSLs and model-driven development. J. Object Technol 2009;vol. 8:79–102.
[8] Barišić A, Amaral V, Goulão M, Barroca B. How to reach a usable DSL? Moving toward a systematic evaluation. In: Proceedings of the 5th international workshop on electronic communications of the EASST: multi-paradigm modeling (MPM 2011), vol. 50; 2011a. p. 13. doi:10.14279/tuj.eceasst.50.741.
[9] Marcus A. The ROI of usability. In: Bias, Mayhew, editors. Cost-justifying usability. North- Holland, Elsevier; 2004.
[10] Bias RG, Mayhew DJ. Cost-justifying usability: an update for the Internet age. Elsevier; 2005.
[11] Kosar T, Bohra S, Mernik M. Domain-specific languages: a systematic mapping study. Inf Softw Technol 2016;vol. 71:77–91.
[12] Gabriel P, Goulão M, Amaral V. Do software languages engineers evaluate their languages?. In: Franch X, Gimenes I, Carvallo J-P, editors. Proceedings of the XIII congreso iberoamericano en 'software engineering" (CIbSE'2010). Cuenca, Ecuador: Universidad del Azuay; 2010. p. 149–62. ISBN: 978-9978-325-10-0
[13] Barišić A, Amaral V, Goulão M, Barroca B. Quality in use of DSLs: current evaluation methods. In: Proceedings of the 3rd INForum – simpósio de informática (INForum2011). Coimbra, Portugal: University of Coimbra.
[14] Basili VR. The role of controlled experiments in software engineering research. In: Basili VR, Rombach D, Schneider K, Kitchenham B, Pfahl D, Selby R, editors. Empirical software engineering issues. critical assessment and future directions. Heidelberg: Springer; 2007. p. 33–7. LNCS
[15] Nielsen J. Usability engineering. Academic Press; 1993.
[16] Moody D, van Hillegersberg J. Evaluating the visual syntax of UML: an analysis of the cognitive effectiveness of the UML family of diagrams. In: Software language engineering. Springer; 2009. p. 16–34.
[17] Nielsen J, Molich R. Heuristic evaluation of user interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems: empowering people; 1990. p. 249–56. ISBN 0201509326
[18] Blackwell A, Green T. Notational systems-the cognitive dimensions of notations framework. In: HCI models, theories, and frameworks: toward an interdisciplinary science. Morgan Kaufmann; 2003.
[19] Green TRG, Petre M. Usability analysis of visual programming environments: a cognitive dimensions framework. Vis Lang Comput 1996;vol. 7:131–74.
[20] Kahraman G, Bilgen S. A framework for qualitative assessment of domain-specific languages. Softw Syst Model 2013;vol. 14(4):1505–26.
[21] Barišić A, Amaral V, Goulão M, Barroca B. Evaluating the usability of domain-specific languages. In: Mernik M, editor. Formal and practical aspects of domain-specific languages: recent developments. IGI Global; 2012a. p. 386–407. doi:10.4018/978-1-4666-2092-6. ISBN 9781466620926.

[22] Barišić A, Amaral V, Goulão M, Barroca B. Quality in use of domain-specific languages: a case study. In: Proceedings of the 3rd ACM SIGPLAN workshop on evaluation and usability of programming languages and tools (PLATEAU) at SPLASH, PLATEAU '11. ACM; 2011c. p. 65–72. doi:10.1145/2089155.2089170. ISBN 978-1-4503-1024-6.
[23] Kieburtz RB, McKinney L, Bell JM, Hook J, Kotov A, Lewis J, et al. A software engineering experiment in software component generation. In: Proceedings of the 18th international conference on software engineering (ICSE'1996). IEEE Computer Society; 1996. p. 552. ISBN 0818672463
[24] Kosar T, Mernik M, Carver J. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empir Softw Eng 2011;17(3):276–304. doi:10.1109/MS.2003.1231149.
[25] Murray NS, Paton NW, Goble CA, Bryce J. Kaleidoquery: a flow-based visual language and its evaluation. Vis Lang Comput 2000;vol. 11:151–89.
[26] Erdweg S, van der Storm T, Völter M, Tratt L, Bosman R, Cook WR, et al. Evaluating and comparing language workbenches: existing results and benchmarks for the future. Comput Lang Syst Struct 2015;vol. 44:24–47. doi:10.1016/j.cl.2015.08.007.
[27] Häser F, Felderer M, Breu R. Is business domain language support beneficial for creating test case specifications: a controlled experiment. Inf Softw Technol 2016a;vol. 79:52–62. doi:10.1016/j.infsof.2016.07.001.
[28] Johanson AN, Hasselbring W. Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. Empir Softw Eng 2016:1–31. doi:10.1007/s10664-016-9483-z.
[29] Albuquerque D, Cafeo B, Garcia A, Barbosa S, Abrahão S, Ribeiro A. Quantifying usability of domain-specific languages: an empirical study on software maintenance. J Syst Softw 2015;101:245–59. doi:10.1016/j.jss.2014.11.051.
[30] Meliá S, Cachero C, Hermida JM, Aparicio E. Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study. Softw Qual J 2016;vol. 24:709–35. doi:10.1007/s11219-015-9299-x.
[31] Izquierdo JLC, Cabot J, López-Fernández JJ, Cuadrado JS, Guerra E, de Lara J. Engaging end-users in the collaborative development of domain-specific modelling languages. In: Cooperative design, visualization, and engineering. Springer; 2013. p. 101–10.
[32] Brooks F. The mythical man-month. Addison-Wesley; 1975. ISBN 0-201-00650-2
[33] Correal D, Casallas R. Using domain specific languages for software process modeling. In: Proceedings of OOPSLA workshop on domain-specific modeling. Montreal, Canada; 2007.
[34] Völter M, Stahl T, Bettin J, Haase A, Helsen S. Model-driven software development: technology, engineering, management. John Wiley & Sons; 2013b.
[35] Gray J, Tolvanen J-P, Kelly S, Gokhale A, Neema S, Sprinkle J. Domain-specific modeling. In: Handbook of dynamic system modeling. CRC Press; 2007. p. 1–7.
[36] Selic B. The pragmatics of model-driven development. IEEE Softw 2003;vol. 20:19–25.
[37] Hermans F, Pinzger M, Deursen AV. Domain-specific languages in practice: a user study on the success factors. In: Proceedings of the 12th international conference on model driven engineering languages and systems, 5795/2009. Denver, Colorado, USA; 2009. p. 423–37. Lecture Notes in Computer Science
[38] Weiss DM, Lai CTR. Software Product-line engineering: a family-based software development process. Addison Wesley Longman, Inc.; 1999.
[39] Benyon D, Green T, Bental D. Conceptual modeling for user interface development. Springer Science & Business Media; 2012. doi:10.1007/978-1-4471-0797-2. ISBN 978-1-85233-009-5
[40] Kühne T. Matters of (meta-) modeling. Softw Syst Model 2006;vol. 5:369–85.
[41] Atkinson C, Kühne T. Model driven development: a metamodeling foundation. IEEE Softw. 2003;vol. 20:36–41. doi:10.1109/MS.2003.1231149.
[42] Strembeck M, Zdun U. An approach for the systematic development of domain-specific languages. Softw Pract Exp 2009;vol. 39:1253–92.
[43] Kleppe AG. Software language engineering: creating domain-specific languages using metamodels. Addison-Wesley; 2009. ISBN 0321553454
[44] Thibault S. Domain-specific languages: conception, implementation and application. Universitè de Rennes; 1998. Ph.D. thesis.
[45] Visser E. WebDSL: a case study in domain-specific language engineering. In: Generative and transformational techniques in software engineering II, 5235. Berlin, Heidelberg: Springer-Verlag; 2007. p. 291–373. doi:10.1007/978-3-540-88643-3_7. Lecture Notes in Computer Science
[46] Czarnecki K. Overview of generative software development. In: Unconventional programming paradigms. Springer; 2005. p. 326–41.
[47] Kosar T, López PEM, Barrientos PA, Mernik M. A preliminary study on various implementation approaches of domain-specific language. Inf Softw Technol 2008;vol. 50:390–405.
[48] Čeh I, Črepinšek M, Kosar T, Mernik M. Ontology driven development of domain-specific languages. Comput Sci Inf Syst 2011;vol. 8:317–42. doi:10.2298/CSIS101231019C.
[49] Rajlich V, Wilde N. The role of concepts in program comprehension. In: Proceedings of the 10th international workshop on program comprehension, 2002. IEEE; 2002. p. 271–8.
[50] Dix A. Human computer interaction. Pearson Education; 2004.
[51] International Organization for Standardization. ISO/IEC 25010:2011 systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) - system and software quality models. International Organization for Standardization (ISO); 2011. Ph.D. thesis.
[52] Shackel B, Richardson SJ. Human factors for informatics usability. Cambridge University Press; 1991.
[53] International Organization for Standardization. ISO/IEC 9241–11 ergonomic requirements for office work with visual display terminals (VDTs) – part 11: guidance on usability. International Organization for Standardization (ISO); 2001. Ph.D. thesis.
[54] Standardization IOf. ISO/IEC 9126: information technology – software product evaluation – software quality characteristics and metrics. Technical Report; 2004.
[55] Petrie H, Bevan N. The evaluation of accessibility, usability and user experience. The universal access handbook Human factors and ergonomics. Stephanidis C, editor. CRC Press; 2009.
[56] Barišić A, Monteiro P, Amaral V, Goulão M, Monteiro MP. Patterns for evaluating usability of domain-specific languages. In: Proceedings of the 19th conference on pattern languages of programs (PLoP), SPLASH 2012. Tucson, Arizona: ACM. ISBN:978-1-4503-2786-2.
[57] Rubin J, Chisnell D. Handbook of usability testing: how to plan, design and conduct effective tests. Wiley-India; 2008.
[58] Paz F, Pow-Sang JA. A systematic mapping review of usability evaluation methods for software development process. Int J Softw Eng Appl 2016;vol. 10:165–78. doi:10.14257/ijseia.2016.10.1.16.
[59] Reisner P. Query languages. In: Handbook of human–computer interaction. Amsterdam, The Netherlands: North-Holland; 1988. p. 257–80.
[60] Nielson J. Usability engineering. AP Professional; 1993.
[61] AlRoobaea R, Al-Badi A, Mayhew P. Generating a domain specific inspection evaluation method through an adaptive framework: A comparative study on educational websites. Int J Hum Comput Interact 2013;vol. 4(2):88.
[62] Lárusdóttir M, Cajander Ã, Gulliksen J. Informal feedback rather than performance measurements: user-centred evaluation in scrum projects. Behaviour & information technology. Taylor & Francis; 2013. p. 1–18. Ahead-of-print
[63] Cockburn A, Highsmith J. Agile software development: the people factor. Computer 2001;vol. 34(11):131–3.
[64] Sy D. Adapting usability investigations for agile user-centered design. J Usability Stud 2007;vol. 2:112–32.
[65] Norman DA, Draper SW. User centered system design. NJ: Hillsdale; 1986.
[66] Vredenburg K, Mao J-Y, Smith PW, Carey T. A survey of user-centered design practice. Proceedings of the SIGCHI conference on human factors in computing systems. ACM; 2002. p. 471–8.
[67] Bevan N. Cost benefits framework and case studies. Cost-justifying usability: an update for the internet age. Morgan Kaufmann; 2005.
[68] Sgall P, Hajicová E, Panevová J. The meaning of the sentence in its semantic and pragmatic aspects. Springer Science & Business Media; 1986. ISBN 90-277-1838-5
[69] Von Alan RH, March ST, Park J, Ram S. Design science in information systems research. MIS Q 2004;vol. 28:75–105.
[70] Simon HA. The sciences of the artificial. MIT Press; 1996.
[71] Wieringa R. Empirical research methods for technology validation: scaling up to practice. Syst Softw 2014;vol. 95:19–31.

[72] Wieringa R. Design science as nested problem solving. In: Proceedings of the 4th international conference on design science research in information systems and technology; 2009. doi: 10.1145/1555619.1555630.
[73] Marques E, Balegas V, Barroca B, Barišić A, Amaral V. The RPG DSL: a case study of language engineering using MDD for generating RPG games for mobile phones. In: Proceedings of the 12th workshop on domain-specific modeling; 2012. p. 13–18.
[74] Sjøberg DIK, Hannay JE, Hansen O, Kampenes VB, Karahasanovic A, Liborg N-K, et al. A survey of controlled experiments in software engineering. IEEE Trans Softw Eng 2005;vol. 31:733–53.
[75] Basili VR. The role of experimentation in software engineering: past, current, and future. In: Proceedings of the 18th international conference on software engineering (ICSE'1996). Berlin, Germany: IEEE Computer Society; 1996. p. 442–9.
[76] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in software engineering: an introduction, vol. 6. Boston, EUA: Kluwer Academic Publishers; 1999.
[77] Barišić A. Evaluating the quality in use of domain-specific languages in an agile way. In: Proceedings of the doctoral symposium at the 16th international conference on model driven engineering languages and systems (MoDELS), CEUR-WS, 1071. Miami, Florida: CEUR-WS; 2013.
[78] Barišić A. STSM report: evaluating the efficiency in use of search-based automated model merge technique. In: Proceedings of multi-paradigm modelling for cyber-physical systems (MPM4CPS), COST action IC1404; 2016. doi: 10.5281/ZENODO.237420.
[79] Leonardo P. Child programming: an adequate domain specific language for programming specific robots. 2013.
[80] Kessentini M, Langer P, Wimmer M. Searching models, modeling search: on the synergies of SBSE and MDE. In: Proceedings of the 1st international workshop on combining modelling and search-based software engineering. IEEE Press; 2013. p. 51–4.
[81] Kobsa A. Generic user modeling systems. In: User modeling and user-adapted interaction, vol. 11. Springer; 2001. p. 49–63.
[82] Halliday MAK. Language as social semiotic. London Arnold; 1978.
[83] Lempia DL, Miller SP. Requirements engineering management handbook, vol. 1. National Technical Information Service; 2009.
[84] Allweyer T. BPMN 2.0: introduction to the standard for business process modeling. BoD-Books on Demand; 2010.
[85] Fowler M. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional; 2004. ISBN 0321193687
[86] Amyot D, Shamsaei A, Kealey J, Tremblay E, Miga A, Mussbacher G, et al. Towards advanced goal model analysis with jUCMNav. In: Proceedings of international conference on conceptual modeling. Springer; 2012. p. 201–10.
[87] Van Lamsweerde A. Requirements engineering in the year 00: a research perspective. In: Proceedings of the 22nd international conference on software engineering. ACM; 2000. p. 5–19.
[88] Chung L, Nixon BA, Yu E, Mylopoulos J. Non-functional requirements in software engineering, vol. 5. Springer Science & Business Media; 2012.
[89] Van Lamsweerde A. Goal-oriented requirements engineering: a guided tour. In: Proceedings of the fifth IEEE international symposium on requirements engineering, 2001. IEEE; 2001a. p. 249–62.
[90] Yu ESK. Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the third IEEE international symposium on requirements engineering, 1997. IEEE; 1997. p. 226–35.
[91] Giorgini P, Mylopoulos J, Sebastiani R. Goal-oriented requirements analysis and reasoning in the tropos methodology. Eng Appl Artif Intell 2005;vol. 18:159–71.
[92] Van Solingen R, Basili V, Caldiera G, Rombach HD. Goal question metric (GQM) approach. In: Encyclopedia of software engineering. Wiley Online Library; 2002.
[93] Vallecillo A, Morcillo C, Orue P. Expressing measurement uncertainty in software models. In: Proceedings of the 2016 10th international conference on the quality of information and communications technology (QUATIC). IEEE; 2016. p. 15–24. doi:10.1109/QUATIC.2016.013. ISBN 978-1-5090-3581-6.
[94] Bevan N. Quality and usability: a new framework. In: van Veenendaal E, McMullan J, editors. Achieving software product quality. Netherlands: Tutein Nolthenius; 1997.
[95] Binucci L. Software quality of use: evaluation by MUSiC. In: Objective software quality. Springer; 1995. p. 165–78.
[96] Alva M, Martínez P A, Cueva L J, Sagástegui Ch T, López P B. Comparison of methods and existing tools for the measurement of usability in the web. In: Web engineering. Springer; 2003. p. 386–9.
[97] Kitchenham BA, Pfleeger SL. Personal opinion surveys. In: Guide to advanced empirical software engineering. Springer; 2008. p. 63–92.
[98] Kolovos DS, Paige RF, Kelly T, Polack FAC. Requirements for domain-specific languages. In: Proceedings of ECOOP workshop on domain-specific program development (DSPD), 2006; 2006.
[99] International Organization for Standardization. ISO/IEC 25022:2016, systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) - Measurement of quality in use.
[100] Barišić A, Goulão M, Amaral V. Domain-specific language domain analysis and evaluation: a systematic literature review. Faculdade de Ciencias e Technologia, Universidade Nova da Lisboa; 2015. doi: 10.5281/ZENODO.265487.
[101] Barišić A, Amaral V, Goulão M. Usability software engineering – modeling environment (USE-ME 1.1). Faculdade de Ciências e Tecnologia, Universidade Nova da Lisboa. doi: 10.5281/ZENODO.345941.
[102] Van Lamsweerde A. Goal-oriented requirements engineering: a guided tour. In: Proceedings of fifth IEEE international symposium on requirements engineering. IEEE; 2001b. p. 249–62.
[103] Barišić A, Amaral V, Goulão M, Aguiar A. Introducing usability concerns early in the DSL development cycle: flowSL experience report. In: Proceedings of MD2P2 2014 model-driven development processes and practices workshop, CEUR-WP, vol. 1249; 2014. doi: 10.1.1.665.6015.
[104] Leech NL, Barrett KC, Morgan GA. SPSS for intermediate statistics: use and interpretation. Psychology Press; 2005. ISBN 1-4106-1142-6.
[105] Sim G, Horton M. Investigating children's opinions of games: fun toolkit vs. this or that. In: Proceedings of the 11th international conference on interaction design and children. ACM; 2012. p. 70–7.
[106] Barišić A, Amaral V, Goulão M. USE-ME empirical evaluation pilot study. Faculdade de Ciencias e Technologia, Universidade Nova da Lisboa; 2017b. doi:10.5281/ZENODO.398830.
[107] Mernik M, Žumer V. Incremental programming language development. Comput Lang Syst Struct 2005;vol. 31:1–16. doi:10.1016/j.cl.2004.02.001.
[108] Segal J, Morris C. Developing scientific software. IEEE Softw 2008;vol. 25:18–20. doi:10.1109/MS.2008.85.
[109] Kendall R, Carver JC, Fisher D, Henderson D, Mark A, Post D, et al. Development of a weather forecasting code: a case study. IEEE Softw 2008;vol. 25:59–65. doi:10.1109/MS.2008.86.
[110] Prechelt L. An empirical comparison of seven programming languages. IEEE Comput 2000;vol. 33:23–9.
[111] Cook S., Jones G., Kent S., Wills A.C.. Domain specific development. 2007.
[112] Bellamy R, John B, Richards J, Thomas J. Using cogtool to model programming tasks. In: Proceedings of evaluation and usability of programming languages and tools (PLATEAU 2010); 2010. p. 1.
[113] Haugen O, Mohagheghi P. A Multi-dimensional framework for characterizing domain specific languages. In: Proceedings of OOPSLA workshop on domain-specific modeling. Montréal, Canada; 2007.
[114] Merilinna J, Perssinen J. Comparison between different abstraction level programming: experiment definition and initial results. In: Proceedings of OOPSLA workshop on domain-specific modeling. Montreal, Canada; 2007.
[115] Kosar T, Oliveira N, Mernik M, Pereira MJV, Crepinšek M, Cruz D, et al. Comparing general-purpose and domain-specific languages: an empirical study. Comput Sci Inf Syst 2010;vol. 7:247–64.
[116] Kärnä J, Tolvanen J-P, Kelly S. Evaluating the use of domain-specific modeling in practice. In: Proceedings of the 9th OOPSLA workshop on domain-specific modeling; 2009.
[117] Wu Y, Hernandez F, Ortega F, Clarke PJ, France R. Measuring the effort for creating and using domain-specific models. In: Proceedings of the 10th workshop on domain-specific modeling. ACM; 2010. p. 14.

[118] McKean D, Sprinkle J. Heterogeneous multi-core systems: UML profiles vs. DSM approaches. In: Proceedings of the 2012 workshop on domain-specific modeling. ACM; 2012. p. 45–8.
[119] Nishino H. Misfits in abstractions: towards user-centered design in domain-specific languages for end-user programming. In: Proceedings of the ACM international conference companion on object oriented programming systems languages and applications companion. ACM; 2011. p. 215–16.
[120] Häser F, Felderer M, Breu R. An integrated tool environment for experimentation in domain specific language engineering. In: Proceedings of the 20th international conference on evaluation and assessment in software engineering. doi: 10.1145/2915970.2916010.
[121] Neumann C, Metoyer RA, Burnett M. End-user strategy programming. Vis Lang Comput 2009;vol. 20:16–29. doi:10.1016/j.jvlc.2008.04.005.
[122] Pane JF. A programming system for children that is designed for usability. Lewis University; 2002. Ph.D. thesis.
[123] Aghaee S, Pautasso C. End-user development of mashups with naturalmash. Vis Lang Comput 2014;vol. 25:414–32. doi:10.1016/j.jvlc.2013.12.004.
[124] Angelini M, Ferro N, Santucci G, Silvello G. VIRTUE: a visual tool for information retrieval performance evaluation and failure analysis. Vis Lang Comput 2014;vol. 25:394–413. doi:10.1016/j.jvlc.2013.12.003.
[125] Ardito C, Costabile MF, Desolda G, Lanzilotti R, Matera M, Piccinno A, et al. User-driven visual composition of service-based interactive spaces. Vis Lang Comput 2014;vol. 25:278–96. doi:10.1016/j.jvlc.2014.01.003.
[126] Bauleo E, Carnevale S, Catarci T, Kimani S, Leva M, Mecella M. Design, realization and user evaluation of the smartvortex visual query system for accessing data streams in industrial engineering applications. Vis Lang Comput 2014;vol. 25:577–601. doi:10.1016/j.jvlc.2014.08.002.
[127] Danado J, Paternò F. Puzzle: a mobile application development environment using a jigsaw metaphor. Vis Lang Comput 2014;vol. 25:297–315. doi:10.1016/j.jvlc.2014.03.005.
[128] Fogli D, Provenza LP. A meta-design approach to the development of e-government services. Vis Lang Comput 2012;vol. 23:47–62. doi:10.1016/j.jvlc.2011.11.003.
[129] Pérez F, Valderas P, Fons J. Towards the involvement of end-users within model-driven development. In: End-user development. Springer; 2011. p. 258–63.
[130] Wuest D, Seyff N, Glinz M. Semi-automatic generation of metamodels from model sketches. In: Proceedings of the 2013 IEEE/ACM 28th international conference on automated software engineering (ASE). IEEE; 2013. p. 664–9.
[131] Cho H, Gray J, Syriani E. Creating visual domain-specific modeling languages from end-user demonstration. In: Proceedings of the 4th international workshop on modeling in software engineering. IEEE Press; 2012. p. 22–8.
[132] Kuhrmann M, Kalus G, Knapp A. Rapid prototyping for domain-specific languages-from stakeholder analyses to modelling tools. Int J Enterp Model Inf Syst Architect 2013;vol. 8:62–74. doi:10.18417/emisa.8.1.4.
[133] Sánchez-Cuadrado J, De Lara J, Guerra E. Bottom-up meta-modelling: an interactive approach. In: France R, Kazmeier J, Breu R, Atkinson C, editors. Models. Lecture Notes in Computer Science(LNCS), vol. 7590. Berlin, Heidelberg: Springer; 2012. p. 3–19. doi:10.1007/978-3-642-33666-9_2. ISBN 9783642336652.
[134] Izquierdo JLC, Cabot J. Enabling the collaborative definition of DSMLs. In: Advanced information systems engineering. Springer; 2013. p. 272–87.
[135] Cánovas Izquierdo JL, Cabot J. Collaboro: a collaborative (meta) modeling tool. PeerJ Comput Sci 2016;vol. 2:e84. doi:10.7717/peerj-cs.84.
[136] Villanueva MJ, Valverde F, Pastor O. Involving end-users in the design of a domain-specific language for the genetic domain. Inf Syst Dev 2014:99–110.