A Requirements Engineering Approach for Usability-Driven DSL Development

Ankica Barišić
NOVA-LINCS, FCT/UNL
Campus de Caparica
Caparica, Portugal 2829-516
a.barisic@campus.fct.unl.pt

Dominique Blouin
LTCI Lab, Telecom ParisTech
Université Paris-Saclay
46 rue Barrault
Paris, France 75013
dominique.blouin@
telecom-paristech.fr

Vasco Amaral
Miguel Goulão
NOVA-LINCS, FCT/UNL
Campus de Caparica
Caparica, Portugal 2829-516
(vma,mgoul)@fct.unl.pt

ABSTRACT

There is currently a lack of Requirements Engineering (RE) approaches applied to, or supporting, the development of a Domain-Specific Language (DSL) and the environment in which it is to be used. We present a model-based RE approach to support DSL development with a focus on usability concerns. RDAL is a RE fragment language that can be complemented with other languages to support RE and design. USE-ME is a model driven approach for DSLs usability evaluation which is integrable with a DSL development approach. We combine RDAL and a new DSL, named DSSL, that we created for the specification of DSL-based systems. Integrated with this combination we add USE-ME to support usability evaluation. This combination of existing languages and tools provides a comprehensive RE approach for DSL development and an interesting case study of languages composition allowing the reuse of the assets of the existing languages. We illustrate the approach with the development of the Gyro DSL for programming robots.

CCS CONCEPTS

•Software and its engineering →Software usability; Domain specific languages; Requirements analysis;

KEYWORDS

Requirements engineering, Domain-Specific language, Usability evaluation

ACM Reference format:
Ankica Barišić, Dominique Blouin, Vasco Amaral, and Miguel Goulão. 2017. A Requirements Engineering Approach for Usability-Driven DSL Development. In *Proceedings of ACM Software Language Engineering conference, Canada, October 2017 (SLE2017),* 12 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Domain-Specific Languages (DSLs) offer expressiveness for modeling systems at the proper level of abstraction, before they are automatically deployed or even simulated, with notations close to the end user domain. 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). Bridging this gap is expected to increase productivity. However, engineers seem to underestimate the importance of aligning the languages, in particular, their notations and tools, with the skills of their end users [39]. Assessing the impact of introducing a DSL into a development process requires focusing on the productivity gains resulting from the extent to which the domain users are able to use the languages with their notations and tools [3]. Investment into this assessment, commonly called usability evaluation, is justified by the resulting reduction of development costs and increased revenues brought by improved effectiveness and efficiency of DSLs end users.

As opposed to software and systems products for which several model-based Requirements Engineering approaches have been developed showing several benefits, DSLs nowadays are, to our knowledge, mostly developed informally without such support. This situation may be due to the lack of consistent and computer-aided integration of two different and demanding complementary software processes: DSL development and usability engineering. Therefore, a more formal and iterative approach is required to develop DSLs and track all requirements, including usability requirements. As for other software products, the approach should include the context of use of the DSL in its environment, as well as the impact of recommendations with well-planned evaluation processes. Such approach can be supported by modeling all these aspects using appropriate languages and tools.

A first attempt of defining the required concepts for such approach has been made through the specification of the Usability Software Engineering Modeling Environment (USE-ME) [1, 2]. However, USE-ME focuses on usability and actually requires a complete RE process on which it can base its evaluation. The Requirements Definition and Analysis Language (RDAL) [6, 7] was developed as a fragmented language to be combined with other modeling languages in support of well known RE best practices such as those recommended by the Requirements Engineering Management Handbook (REMH) [21] and those of GORE (Goal-Oriented Requirements Engineering) [36]. RDAL was originally planned to become a standard annex of the Architecture Analysis and Design Language (AADL, SAE AS5506B standard) [1] for supporting requirements capture, analysis and verification. In the end, it led to the

SLE2017, Canada

DOI: 10.1145/nnnnnnn.nnnnnnn

[1] http://standards.sae.org/as5506b/ Accessed June 15 2017

development of the ReqSpec language and ALISA (Architecture-Led Incremental System Assurance) approach [11], which are however strongly coupled with the AADL. In contrast, the modularity of RDAL allows its reuse with other Architecture Description Languages not necessarily targeting the safety-critical embedded systems domain. Although the REMH has been written for the domain of safety-critical embedded systems, a large majority of its practices that are supported by RDAL are generic enough to be applicable to the development of many other types of systems. As a matter of fact, the concepts of the RDAL-REMH approach share many similarities with the concepts required by USE-ME.

Therefore, in this paper we present a complete framework for the RE of DSL development based on RDAL-REMH and combined with USE-ME, in order to provide a focus on the important concern of usability. This allows USE-ME to benefit from the REMH well-established practices and is also an opportunity to evaluate the planned reuse capability of RDAL for a very different domain. Furthermore, such combination of existing languages and tools constitutes an interesting case of languages composition and reuse.

In the next sections, first the RDAL-REMH and USE-ME approaches are introduced. Next, a study mapping the REMH best practices, where possible, to those of the RDAL and USE-ME approaches is presented in section 3. This study showed the need for a new DSL for the specification of DSL-based systems that we developed. This language is introduced in section 4. Next, the approach is illustrated by the modeling of the development of the Gyro DSL for programming robots.

2 BACKGROUND

2.1 RDAL-REMH

In 2008, the US Federal Aviation Administration (FAA) commanded a state-of-the-art of the RE research and a survey of the current industry practices [20]. The purpose was to select and adapt methods for the successful management, integration, verification and validation of requirements that may be developed by multiple entities. The results of that study lead to 11 best practices presented in the Requirements Engineering Management Handbook (REMH) [21]. Such practices were developed to support their incremental adoption, following a sequence tailored to each organization, in order to minimize the risk of disruption of the organization's development processes thus favoring a smoother adoption by industry of results from the RE research.

The first column of table 1 lists the 11 best practices of the REMH with a short description for each of them. Despite that the REMH was developed for the embedded systems domain, it can be observed from table 1 that a large majority of the recommended practices are not specific to this domain and can potentially be beneficial for other domains.

A combination of languages and tools was proposed to provide support of the REMH practices with models [6]. It involved the combination of the User Requirements Notation (URN, ITU recommendation Z.150 standard) [2] to model use cases, AADL for modeling safety-critical embedded systems architectures and the RDAL [7] to model and analyze requirements. This combination of languages was exercised for one of the example requirements specification provided by the REMH and illustrating its practices. The approach was very beneficial as it allowed improving significantly the quality of the specification [6]. The jUCMNav tool [3] was used for the URN language and the OSATE tool for AADL [4]. The RDAL language and its combination with URN and AADL was provided by the RDAL Tool Environment (RDALTE) [5].

While so far the fragment language RDAL has only been used with URN and AADL, it was explicitly designed to be reusable with other languages than AADL, not necessarily targeting the safety-critical embedded systems domain. Therefore, retargeting the approach for a different domain such as DSL development was of great interest in order to validate the adaptability of the RDAL-REMH approach, but also to provide model-based RE for DSL development.

2.2 USE-ME

USE-ME [1] promotes an iterative user-centered evaluation approach for DSLs. Usability evaluations are expressed as regular expert evaluator activities, in the software language engineering process. Usability engineering aims at increasing 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. Further, software engineering supports the systematic development and is concerned with all aspects of software production. However, the USE-ME framework is not intended to fully support a complete development cycle. Instead, it is supposed to be integrated with existing approaches which support DSL development. For instance, a requirements engineering process for which the objective is to provide a view on usability over a complete set of requirements models. The utility specification package enables mapping to the artifacts which are commonly developed during DSL development process, like a DSL architecture, the existing goal model and requirements, or other specification diagrams (e.g. use cases, business processes).

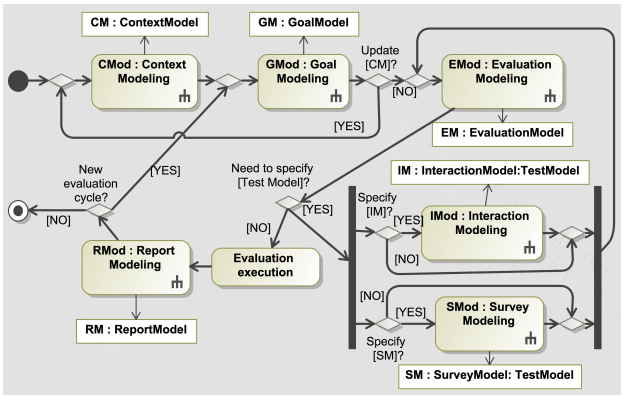

Figure 1: USE-ME activity diagram taken from [1]

[2] http://www.itu.int/rec/T-REC-Z.150/en/ Accessed June 15 2017

[3] http://jucmnav.softwareengineering.ca/jucmnav/ Accessed June 15 2017

[4] http://osate.org/ Accessed June 15 2017

[5] https://wiki.sei.cmu.edu/aadl/index.php/RDALTE/ Accessed June 15 2017

The process of usability evaluation using the USE-ME framework is presented in the activity diagram of figure 1. First, a Context Model is produced, which supports the description of the language context. The context modeling process enforces users profiling and the creation of prioritized user hierarchies, as well as modeling of the physical, technical and social environments. Further, it requires the specification and prioritization of workflow scenarios, whose actors should be already a part of a context model. The next step consists of describing the usability goals during goal modeling. While specifying the goals and their scope (e.g. by using context model instances), it is likely that new context elements are found which are relevant for the use of the DSL. In this case, it is necessary to update the Context Model and proceed with specifying a Goal Model until there is at least one usability goal for which only an actor representing an expert evaluator is responsible. This usability goal can be traced back to the functional requirements on which it depends, and its usability requirements are defined with associated metrics, which will determine success after evaluation. Further, it is necessary to define an Evaluation Model, which highlights evaluation goals and their corresponding evaluation steps. There is a *'need to specify a [Test Model]'* or reuse an existing one. The Test Model is crucial for the usability assessment process and can be defined as an Interaction Model or/and a Survey Model. These two modeling activities depend on the same Evaluation Model and should be performed in parallel in order to complement each other. However, for certain types of evaluation, it is not necessary to develop both models. For instance, when performing a heuristic evaluation, a checklist implemented as a Survey Model can be sufficient. When the Evaluation Model is ready, we can proceed with the 'Evaluation execution'. Finally, it is necessary to analyze the results of the test models and to create the Report Model, that recommends a Goal Model extension 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 finalize the assessment period. Ideally, this decision will eventually indicate the end of the development cycle.

3 FEASIBILITY / COMPARATIVE STUDY

We first studied the concepts of both the RDAL-REMH and USE-ME approaches and compared them in order to evaluate the required effort and potential benefits of extending RDAL-REMH to support USE-ME. In some preliminary work [5] we showed that many RDAL-REMH concepts are also partially supported by USE-ME (see table 1). However in RDAL there is no specific focus on usability and usability elements can only be identified by using the RDAL user-defined category system. Nevertheless, the modeling of all other RE concerns with RDAL canl provide an essential basis for the USE-ME viewpoint on usability.

A strength of USE-ME is its usability goal coverage or achievement analysis supporting iterative development of the DSL. A similar optimization of quality attributes approach was also developed for RDAL but in the context of embedded systems development during the refinement of architecture models [22]. However, the RDAL approach does not support the notion of associated context or design of empirical studies, which are necessary for evaluation of usability. As for use cases, it was observed that USE-ME could greatly benefit from the modeling of use cases using the Use Case Maps sub-language of the URN by allowing their simulation thus complementing the corresponding Workflow concept in USE-ME.

Table 1: Mapping the REMH best practices to RDAL/AADL-UCM and USE-ME

	REMH Best Practice	RDAL-AADL-UCM Concepts	USE-ME Concepts
1	Develop the System Overview, System Context and System Goals	RDAL System Overview and Context RDAL Goals	Context Model and Usability Goal Model
2	Identify the System Boundary	AADL features from the system component identified by the RDAL system overview	N / A
3	Develop the Operational Concepts	UCM use cases	Workflows
4	Identify the Environmental Assumptions / External Entities	AADL components from the environment identified from the RDAL system overview and RDAL Assumptions	User Hierarchy and Environment Context (technical, physical and social environment)
5	Develop thje Functional Architecture and High-Level Requirements	RDAL top level of hierarchy requirements	N / A
6	Revise Architecture to meet Implementation Constraints	RDAL evolution traceability links	N / A
7	Identify the System Modes	AADL system operation modes	Scope of usability goals
8	Develop the Detailed Behaviour and Performance Requirements	RDAL requirements refining the top level requirements	Only addresses usability performance requirements
9	Develop the Software Requirements	RDAL refining specification containing requirements refining the system requirements	N / A
10	Allocate Systems Requirements to Subsystems	RDAL / AADL refining specifications	N / A, but could be considered
11	Provide Rationale	RDAL Rationale construct	Method of Usability Goal

It should also be noted that some parts of the RDAL-REMH approach are not relevant for DSL development. For example, the software requirements best practice # 9 dealing with the translation of system requirements into software requirements taking into account the different representation of system variables in software does not need to be considered. Such software requirements do not exist for DSL development.

Overall, given this comparison, we conclude that many of the features provided by RDAL-REMH can be beneficial for USE-ME as it would avoid redeveloping these constructs for the USE-ME languages. Furthermore, this comparison will be the support of a mapping defined between the concepts of the languages for their integration as described in section 5.2.

A major issue that was found, however, is the lack of an appropriate language for representing the architecture design of a DSL in a similar fashion provided by AADL for embedded systems. Reusing AADL for modeling a DSL and its environment would not be appropriate due to the difference between the domains. This triggered the development of a simple design language for the specification

of DSL-based systems, namely the DSL-based Systems Specification Language (DSSL) introduced below.

4 DSSL

The purpose of the DSSL language is to model the design of a DSL being developed and the way it is used in its environment. Goals, requirements and environmental assumptions can then be assigned to elements of this representation of the DSL under development. Furthermore, DSSL can integrate descriptions of the syntaxes of the DSL implemented as an Ecore meta-model and potentially including graphical and / or textual concrete syntax(es) respectively represented as Sirius or / and Xtext grammar models.

4.1 Declarative Specification

Because there are typically several contexts of use for a DSL, the DSSL needs to provide declaration of types for the various elements present in the contexts. Such types allow reusing the characteristics of the declared types across various contexts.

The core DSSL declarative concepts are presented in figure 2. The *DSSL specification* class is used as a root container of all elements of a DSSL specification. Such elements consist of *context specifications* and *entity types* to be instantiated in a context specification for representing the developed DSL and the entities it interacts with. Interaction capabilities with other entities are described by *interaction features* contained by entity types. An interaction feature must refer to a *reference* declared in subclasses of entity type. Such reference is used as typing for the interaction between instances of entity types in a context of use.

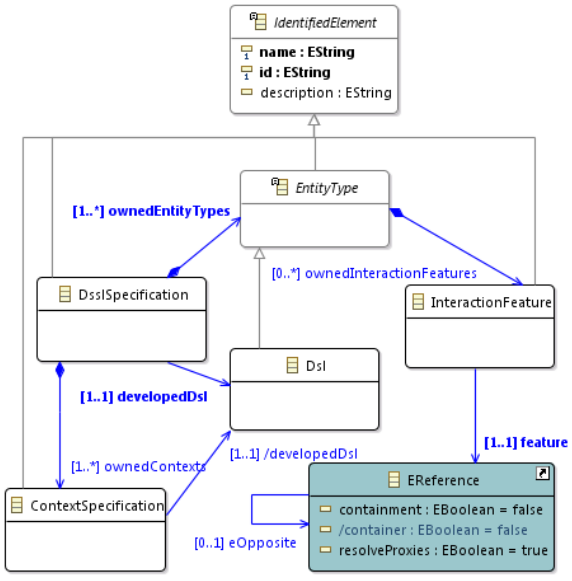

Figure 2: The core DSSL declarative elements

Subclasses of the DSSL entity type class are provided to represent *users*, *tools*, *workplaces*, *documentations* and *physical systems* (e.g. a robot) as depicted in the class diagram of figure 3. A tool can *control* or be *controlled by* other tools. Tools are further specialized into *hardware tool* and *software tool*. A software tool can support a number of DSLs and, conversely, a DSL can be supported by a number of software tools. Subclasses of software and hardware tools such as *operating systems* and *computers* are provided and declare properties specific to these elements. For instance, a computer *executes* a number of software tools, and conversely, each tool may be executed by several computers. A computer makes use of a number of *display* devices of some resolution and *color schemes*.

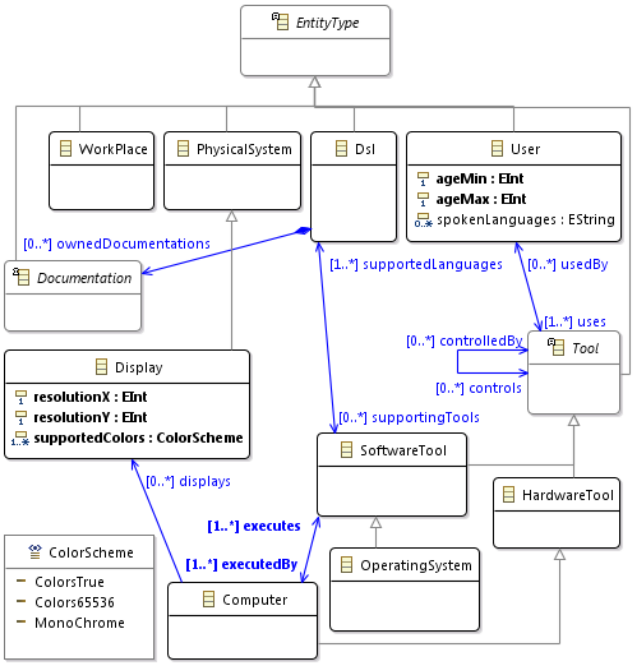

Figure 3: DSSL entity type specializations

4.2 Context Specification

The purpose of the context specification concepts is to provide means to specify the various contexts of use of the developed DSL and its interaction with entities of its environment. The DSSL context concepts are presented in figure 4. A *context specification* captures a set of *entity instances* representing the various entities involved when the DSL is used and how they interact with each other. Each entity instance refers to an entity type from the declarative specification and thus inherits its declared characteristics.

Two entity instances can be connected to each other via an *entity instance connection*, which relates a source interaction feature of the entity type of a source entity instance to a destination interaction feature of the entity type of a destination entity instance. Compatibility of the connected interaction features is checked by inspection of the type of the Ecore reference that must belong to the class of the entity type, which must be compatible with the class of the entity type of the connected entity instance (that is, they can be from either the same class or from a subclass). Examples of this are provided in section 5.1.1.

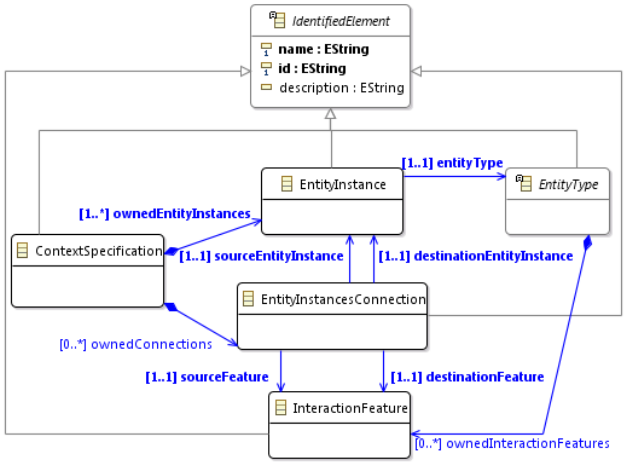

Figure 4: DSSL context instance elements

4.3 DSL Specification

The purpose of the DSL specification concepts is to allow modeling the DSL under development and its internal constituents such as its abstract and concrete syntaxes, its semantics, documentation and optionally some feature diagrams that may have served in elaborating the DSL. The DSL-related concepts are shown in figure 5.

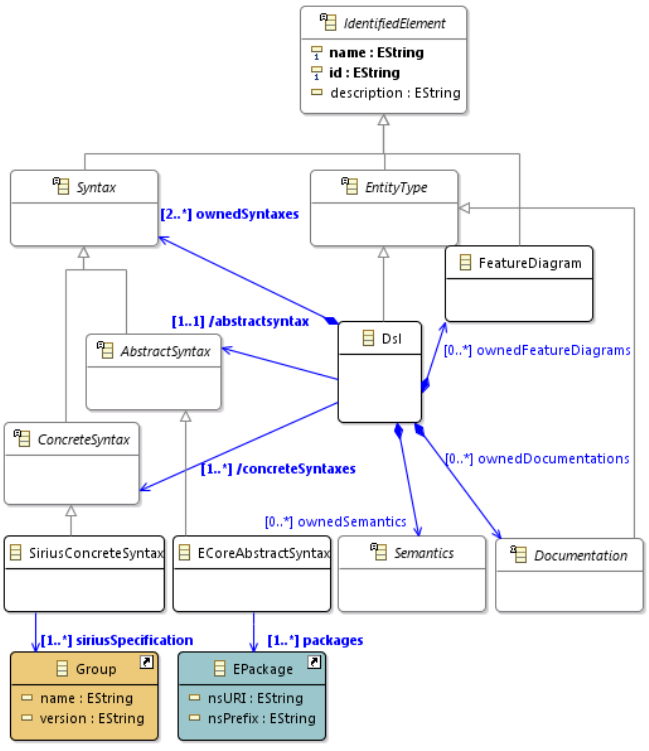

Figure 5: DSSL DSL description elements

A *Dsl* entity type owns a set of *syntaxes* providing the language vocabulary. A syntax can be of *abstract syntax* or *concrete syntax* kind according to its intended use by respectively computer programs or humans. Concrete implementations of such syntaxes must be provided to contain the DSL vocabulary that can be expressed as metamodels, grammars or graphical syntax models. The DSSL language provides default implementations classes for Ecore metamodels and Sirius diagram notation models[6].

In order to achieve this, an *Ecore abstract syntax* class can be instantiated and refer to a set of *Ecore packages* providing metamodel classes and properties for the abstract syntax of the DSL. As in this DSL we are using a graphical syntax, a *Sirius concrete syntax class* can be instantiated and refer to a set of *Sirius specifications* providing graphical notations for nodes and edges of diagrams for the DSL.

Providing such concrete syntax implementations allows for assigning requirements to the contained elements for verifying important properties such as usability. This is illustrated in section 5.1.4.

5 APPROACH

We used the Gyro DSL[7] to illustrate our approach. Gyro was selected because it was already used to validate the USE-ME framework. It is a visual language that allows children to develop programs to control Arduino robots through the manipulation of visual elements. It avoids having to program the Arduino textual code directly, which requires technical skills not owned by the vast majority of children. Gyro specifications programmed by children are automatically translated into Arduino textual code. This activity is expected to help children to quickly understand the programming mechanisms. A strong engagement factor is that children have the opportunity to observe their own developed program running on a real physical robot. This way, they receive feedback on the implications of changes in their program.

Gyro's first prototype (originally named Visualino) was developed as a joint project between Artica, a company that specialises in the development of robotic and audio-visual solutions in Lisbon and the NOVA LINCS research lab. During the development, exploratory evaluations were conducted with the objective of assessing the usability of the visual syntax provided by the language, the learnability of the associated tooling and users satisfaction, which is one of the primary characteristics to be assessed during usability evaluations.

5.1 RE for Gyro with RDAL-REMH

The requirements specification for Gyro follows the REMH best practices of Table 1. We combine RDAL, DSSL and UCM to support these practices with models. This combination of languages is achieved with dedicated traceability links starting from RDAL models to DSSL and UCM models. In order to visualize the combined models, we provide a graphical representation for the combined languages composed of the individual notations of the individual languages. Diagrams of this representation will be displayed to present the modeling the Gyro DSL development example.

[6]https://www.eclipse.org/sirius/ Accessed June 15 2017
[7]http://gyro.artica.cc/ Accessed June 15 2017

5.1.1 System Overview. The very first practice of the REMH advocates to capture a synopsis for the system to be developed including its purpose, and to provide an overview of the system and its environment for all the contexts in which it will be used. It also recommends to capture preliminary system goals.

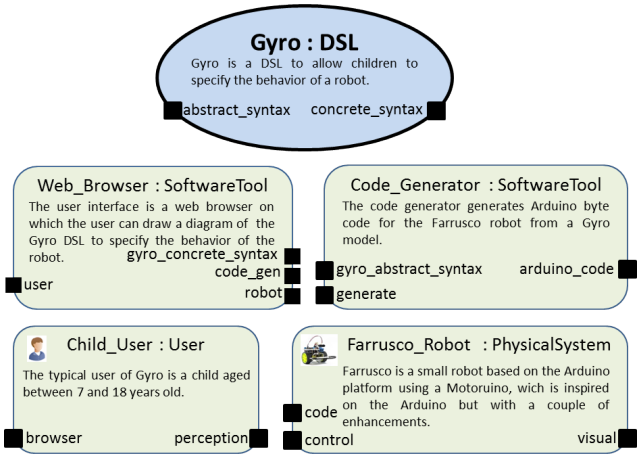

Figure 6: A system overview diagram for the the Gyro language

We consider that the system to be developed is a complete DSL including its abstract and concrete syntaxes and its semantics. The use of a DSL typically involves a user interface displaying the DSL concrete syntax and some computer program(s) that will perform some business logic with models of the DSL making use of the its abstract syntax. For the Gyro DSL, the external entities consist of a web browser serving as user interface, a child user, a code generator that transforms Gyro models into Arduino byte code and a Farrusco robot that executes the behavior programmed in Gyro. We use concepts of both RDAL and DSSL to describe these entities. On the DSSL side, instances of the *DSL*, *SoftwareTool*, *User* and *PhysicalSystem* entity types are used for that as shown in figure 6. For each entity, a description in natural language is provided and a set of *interaction features* that describe how instances of the entity types can interact with each other.

On the RDAL side, an instance of the RDAL *system overview* class is provided to identify the DSSL elements that are part of the system overview. Such system overview element is represented by the canvas on which the DSSL entity types are drawn in figure 6. The RDAL system overview is linked to the *DSSL specification* that contains the entity types. The system to be developed (the Gyro DSL entity type) is also identified via a dedicated traceability link from the RDAL *system overview* to the DSSL Gyro DSL entity type. This link also allows to compute the system boundary as the set of interaction features of the Gyro DSL entity type and stored in a property of the RDAL system overview class. The system overview class also provides additional text attributes to capture the purpose of the system and its synopsis.

System Context. The REMH recommends context diagrams for describing the different contexts in which the system is to be used in its environment. Such diagrams are captured again as a combination of RDAL and DSSL elements. On the DSSL side, a *context specification* instance is created containing a set of *context entity instances*, each of which being typed by one of the predefined *context entity types* declared in the system overview (figure 6).

A context diagram for the normal use of Gyro is shown in figure 7, where instances of the entity types of the system overview are depicted as rounded boxes connected to each other via interaction features declared in their respective types. The DSL system to be developed is highlighted by a thicker border and darker background color.

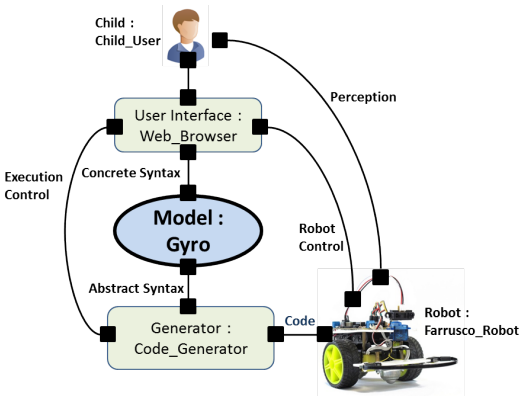

Figure 7: A context diagram for the normal use of the Gyro language

For our example, a child user interacts with the web browser serving as user interface, which itself interacts with the Gyro DSL through its concrete syntax. The Gyro DSL interacts through its abstract syntax with the computer program performing code generation for the robot. The user can trigger code generation from the user interface and then start the robot from the user interface and observe its behavior.

System Goals. System goals are captured in RDAL in terms of functional and non-functional goals. RDAL makes a clear distinction between goals and requirements. As opposed to requirements, which are either verified or not by the system in a Boolean manner, goals may be partially achievable - this is denoted with a percentage of achievement from 0% to 100%. Furthermore, goals can have unresolved conflicts, unlike requirements, since this would indicate an unfeasible system. The level of achievement and conflict among goals can be used to support design optimization and trade-offs, as in [22].

Each goal can be associated with a set of stakeholders from which it origins. A set of rationale elements can also be associated with a goal describing why the goal exists. Each rationale must however be linked to a set of stakeholders of the goal from which it originated.

Finally, RDAL goals can be linked to UCM use cases that detail how the goal is achieved with various scenarios describing the interaction of the user with the system and other entities. The advantage of such practice is to indicate why the use case exists and ensure it is a needed behavior.

For the Gyro DSL, the functional goals are:

- G1: The user should be provided with concepts to specify behaviors of robots
- G2: The teacher should be supported to create customized configurations
- G3: The software engineers should be supported to create new language components
- G4: The DSL should be easy to use for non robot specialists

RDAL goals are depicted as rounded corner boxes in the diagram of figure 8 and linked to their stakeholders and rationales, which are respectively represented as man and cloud symbols. Also represented on the diagram is the use case for programming the behavior linked to the functional goal G1 it aims to achieve. The modeling of such use cases is presented in the next section that covers the development of the operational concepts best practice of the REMH.

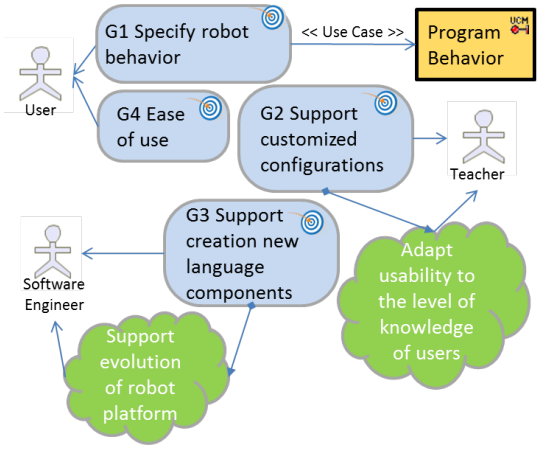

Figure 8: A diagram of the goals for the Gyro language

5.1.2 Operational Concepts. The purpose of this practice is to develop scenarios for the use of the system. The REMH suggest to capture such scenarios with use cases. We have chosen the UCM sub-language of the URN for that. UCM has the advantage of showing on the same diagram both the entities (actors or systems) and the actions that they perform in a quite compact notation (figure 9). An entity is represented by a box containing the actions (steps) performed by the entity. Actions are represented as crosses located along a path representing sequences of actions, which may have forks and joins for representing different scenarios occurring upon specific conditions. Use cases can be reused by being called from other use cases as represented by diamonds (figure 9). This is particularly useful to capture exceptions cases specifying how entities respond to exceptions in a modular way.

We present a few use case diagrams for Gyro that were developed with the jUCMNav tool. jUCMNav provides a graphical editor and a simulator for use cases scenarios.

Normal Use of the DSL. Figure 9 shows the use case for the normal use of the Gyro DSL where entities in the use case diagram correspond to entities in the context diagram of figure 7 as identified by their identical names. The use case describes a dialog between the child user and the web browser user interface. The path shows how the browser responds to actions (crosses) initiated by the child. Such responses are captured as sub-use cases describing the interaction between the browser and other entities such as the DSL model, the code generator and the robot. Pre and post conditions can be set describing the conditions under which the use case scenarios can be executed and what changes are performed after the use case.

The user first launches the Gyro programming application. The user interface then displays the start page with appropriate action menus. The user then asks to create a new robot behavior. The browser displays the programming page. The user then programs a behavior as described by a sub use case. Whe the program is completed, the user tries to execute the programmed behavior on the robot. Then, two sub cases are provided depending on whether the robot is present or not in the environment. The case when the robot is absent corresponds to an exception case deviating from the normal sunny day behavior. A specific sub use case is provided to describe how the system should handle this exception. When the behavior is correct, the user saves the model and the use case ends.

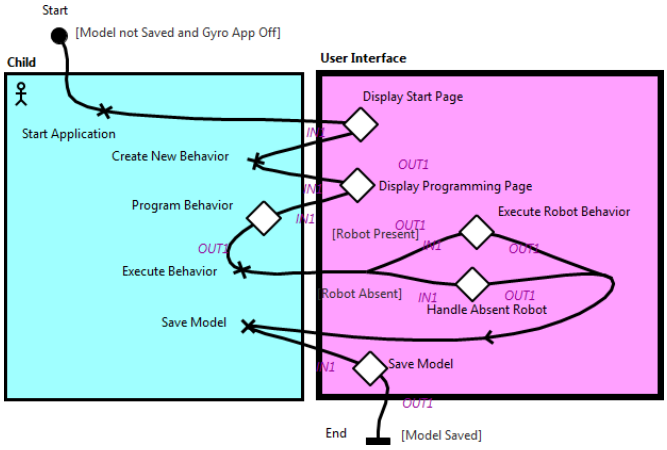

Figure 9: A UCM diagram for the normal use of the Gyro DSL

"Create Branch" Sub Use Case. The complete set of use cases for the Gyro DSL is too large to be presented in this paper. However one of the sub use cases for creating a branch in a Gyro model is presented to further illustrate the best practice. This sub use case will also be used to illustrate how requirements for the system functions can be traced to use case steps from which they were identified. The bottom part of figure 11 shows this sub use case. The user either selects a parallel or a sequential node and the user interface responds to the action by displaying the graphical concrete syntax element on the interface. The red trace on the figure represents the simulation of the scenario for the sequential node.

5.1.3 Environmental Assumptions. The REMH recommends to specify the assumptions the system makes on its environment in order to operate correctly. Identification of a system's environmental assumptions is essential for enabling the reuse of the system in different environments. It has been shown that failure to identify environmental assumptions can lead to misuse of the system and is a common cause of failure [21].

An example of a reasonable assumption for the Gyro DSL could be that the user interface can display colors. This assumption would be violated if the user interface turned out to be a basic LCD display located on the robot, for example. The Gyro DSL highly relies on such assumptions, since its concrete graphical syntax makes use of colors to indicate invalid specifications. Another assumption is that the user can speak a specific language, such as English, in which the DSL's concrete syntax is expressed.

The RDAL *assumption* construct is used for modeling environmental assumptions as shown in figure 10. The assumption is assigned to the DSSL *user* entity type of figure 6 and its constraint, expressed in the Object Constraint Language (OCL) [8] checks that English belongs to the languages spoken by the user. The rationale for this requirement is captured and linked to the software engineer stakeholder that issued this assumption based on the concrete syntax he defined.

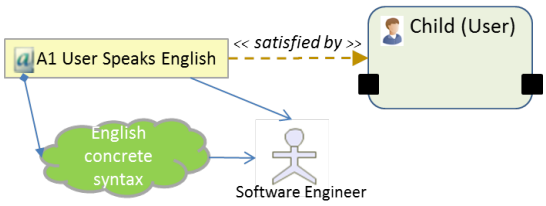

Figure 10: Example environmental assumption for the user of the Gyro language

5.1.4 System Functions. Inspired from the four variable model proposed by Parnas and Madey [28], the REMH recommends to capture a complete and consistent set of detailed system requirements that define how the system must change the variables it controls in response to changes of the variables it monitors. This shall be done for each possible system state and inputs and, if relevant, the allowed tolerance of the specified values and performance characteristics should also be specified.

Functional Requirements. Applied to DSL development, this consists of specifying what syntaxes should be defined to support the specification of the behavior of a robot. In order to do this, a hierarchy or RDAL refined requirements is defined, starting from a very high level requirement for the Gyro DSL entity type (figure 11) refined into two requirements; a requirement R1.2 stating that an abstract syntax should be provided and a requirement R1.1 stating that a concrete syntax should be provided. R1.2 is assigned to a DSSL *Ecore abstract syntax* instance that refers to an Ecore package providing the DSL classes and relationships, while R1.1 is assigned to a DSSL *Sirius concrete syntax* instance that refers to a Sirius model providing graphical syntax elements for the DSL. Both of these requirements must be verified for the refined requirement R1 to be verified.

The leaves of the requirements refinement tree (R1.2.1 and R1.2.2) are assigned to the Gyro Ecore package to check that it contains the required sequential and parallel classes. Identifying such classes is achieved by an OCL expression examining the classes of the package and checking for predefined Ecore meta annotations attached to the

[8] http://www.omg.org/spec/OCL/

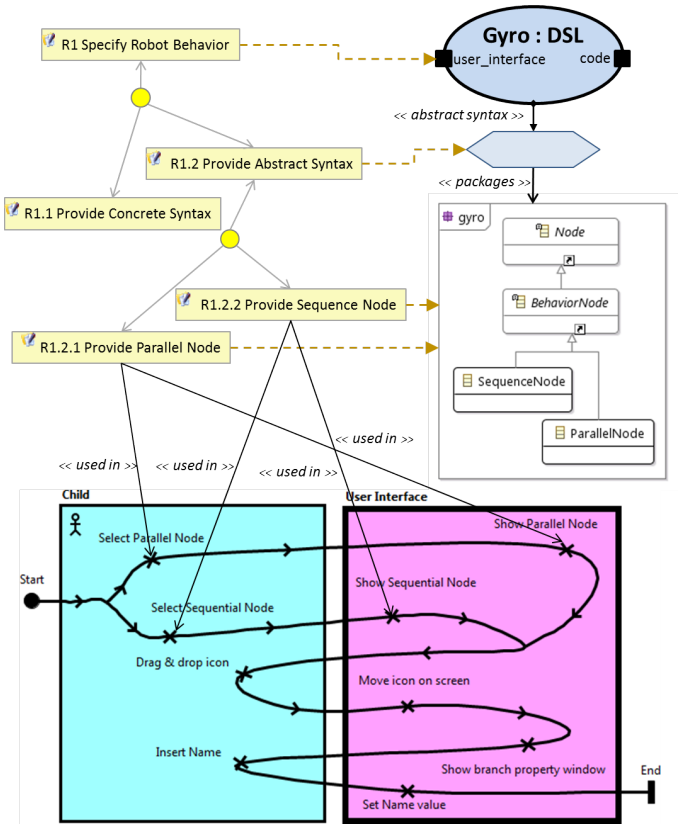

Figure 11: Example functional requirements for Gyro's abstract syntax

classes and specifying their role. Also note that R1.2.1 and R1.2.2 are also linked to steps of the UCM use cases from which their need was discovered (section 5.1.2).

Figure 12 shows similar requirements for the Gyro concrete graphical syntax specified as Sirius models. R1.1 is refined by R1.1.1 that requires that a Sirius diagram definition exists. Such requirement is refined again by a set of requirements for each required graphical element of the syntax such as nodes and edges. Those requirements are expressed by OCL queries that check for the existence of a graphical element associated with the Ecore class of the given role.

Non-Functional Requirements. The previous subsection has shown how functional requirements are captured. However non-functional requirements can be captured as well, especially those for the usability of the concrete syntax.

We illustrate this by presenting an example requirement for the consistency of a graphical notation. For instance, the graphical syntax of the AADL suffers from several inconsistencies. The virtual / abstract components are represented by a dashed border, but that same border is also used for some non-abstract components. The dashed border, which at first seems to indicate the virtuality characteristic of such components, is, after all, also used for non-abstract components, thus leading to an inconsistent and confusing notation. This problem could have been automatically detected by a

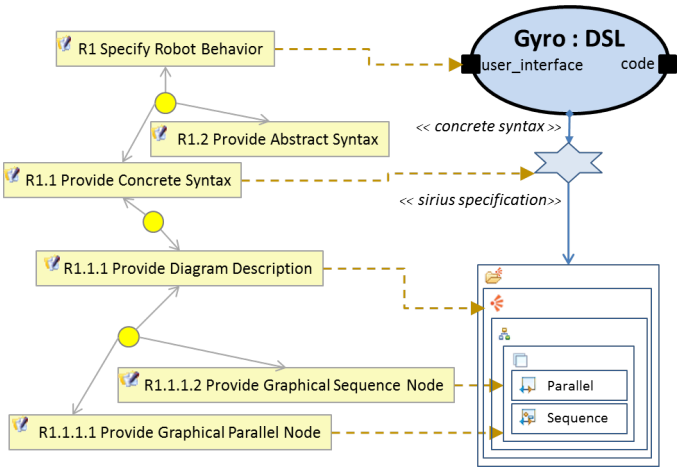

Figure 12: Example functional requirements for Gyro's concrete syntax

consistency notation RDAL requirement assigned to the graphical nodes of the notation.

Applied to Gyro, the metamodel includes a set of nodes that describe behavior such as the parallel and sequence nodes. We create a RDAL requirement that is assigned to the package containing the Gyro classes. The requirement checks for all contained classes that if the class is a behavioral node (i.e. it extends the *Behavior* class) then its corresponding Sirius notation share some visual attribute identifying the behavioral characteristic.

Many more requirements of that kind could be added, for example, to verify that best practices in defining graphical notations such as those suggested in Moody's *"Physics of Notations"* [25] are followed.

Similarly, RDAL non-functional goals and requirements are used to respectively support optimization of usability and require a minimum usability threshold for concrete syntax elements. For example, the RDAL non-functional goal G4 (figure 8) for maximizing usability is assigned to the Sirius model. The level of achievement of the goal (a number between 0% and 100%) is then valued through an annotation of the graphical syntax model following a usability evaluation with USE-ME. The result is then retrieved from the annotation by the goal's OCL expression to provide the level of achievement of the goal. Associated with this goal, we can also define a requirement that the resulting usability is greater than a given threshold to indicate if the notation is usable enough or not for the child user.

This completes the presentation of the modeling of Gyro with RDAL-REMH. In the next section, we present how USE-ME is integrated to support usability-driven DSL development.

5.2 Integrating RDAL-REMH and USE-ME for Usability-Driven Development

The RDAL-REMH approach lacks support of context-aware goal evaluation, which is necessary for usability evaluation. Usability evaluation is performed in a concrete context of use, e.g. with particular users, in a specific environment and performing selected scenarios. This means that after the validation of a usability goal and associated requirements, the results reflect only a partial scope. This is because it would be too expensive to perform evaluations taking all different combinations of the context model instances (e.g. using all possible robot configurations, testing all possible scenarios with participants, and having a significant number of participants having all possible combination of demographic and knowledge characteristics). Therefore, the USE-ME approach suggests a calculation of a *Success Coverage* of the usability goal, which will reflect the percentage of the scope which was taken in consideration during a validation, when compared to the complete context specification of the usability goal.

However, many context-related concepts and the DSL architecture are already captured in the RDAL-REMH approach described in the previous section. Therefore, it is necessary to enable usability evaluation to reuse and refer to the RDAL artifacts while performing context and goal modeling activities suggested by the USE-ME approach. On another hand, while applying the USE-ME approach it is likely that context and goal elements will be extended, or new ones discovered therefore directly contributing to the requirements refinement and DSL artifacts specification. In the following, we present an integrated process and highlight interaction points and information flow between the RDAL-REMH and USE-ME approaches (see figure 13). The idea is to connect the RDAL non-functional goals referring to usability with a 'Quality in Use' Usability Goal of the USE-ME Goal Model. This root goal represents the highest objective of the USE-ME modeling approach (usability for all possible workflows, environment combinations and profiles).

USE-ME provides a Utility Package which supports mapping of the artifacts relevant for the application of the USE-ME process. First, it is to define a DSL and refer to its abstract and concrete syntaxes, which can be done directly by correlating a DSSL Gyro specification with a *DSL* artifact of USE-ME[9] and relating a concrete/abstract syntax version in each development iteration. This helps to trace which evaluation context was considered in the USE-ME application cycle. Further, the USE-ME *DSL* has an *Existing Goal Model*, which refers to the complete RDAL requirements specification model. The functional goals are further obtained from the RDAL *Goals Package* and functional requirements from the RDAL *Requirements Package*. USE-ME expects correlation of requirements with a goal to which it contributes. Finally, the UCM use case diagrams can be mapped to the *Process Model* USE-ME artifact. After this initial mapping, we follow the USE-ME activities (see figure 1) to describe the further integration.

5.2.1 Context Modelling. During context modeling it is necessary to specify *User Profiles* and prioritize the *User Hierarchy* taking into account that there is the need to perform a usability evaluation of each of them. The USE-ME framework suggests that the first level children *User Profiles* should be referred as the DSL development stakeholders. We can obtain information about them from the RDAL *Stakeholder* definitions; namely 'Teacher', 'Software Engineer' and 'User'. Further, USE-ME suggests the addition of a new stakeholder representing a usability evaluator, as it is favorable

[9]Gyro is named Visualino in the model as specified usability evaluation was performed on a previous version of the language

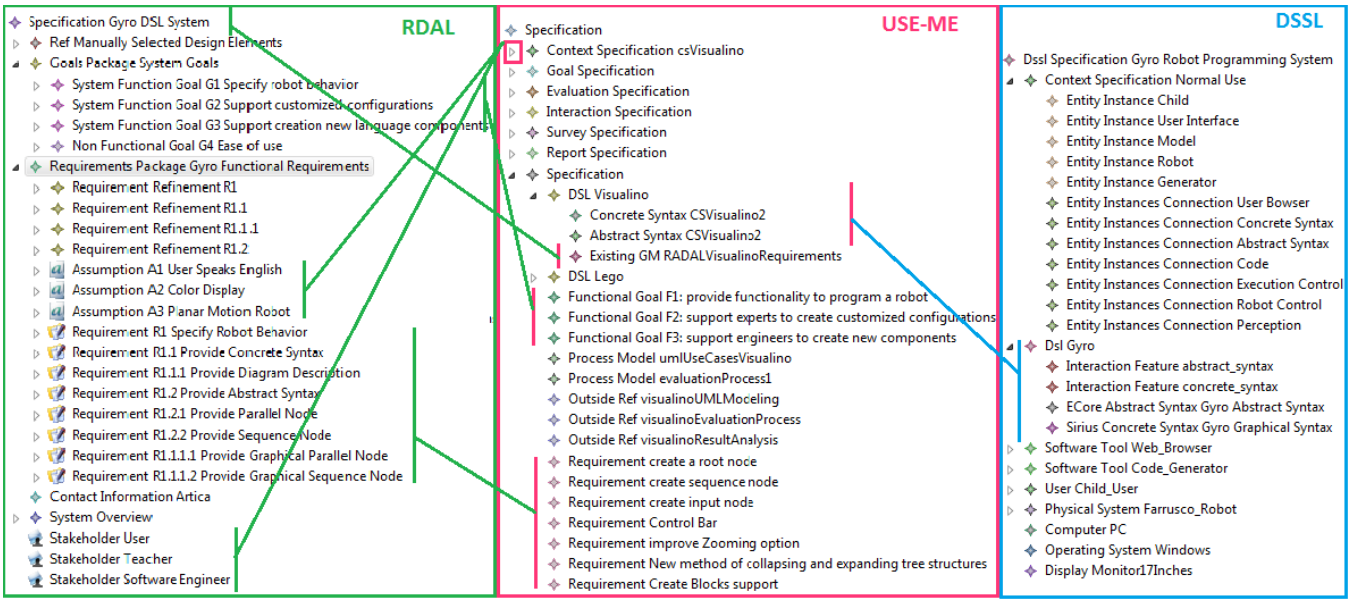

Figure 13: Integration points between RDAL, DSSL and USE-ME

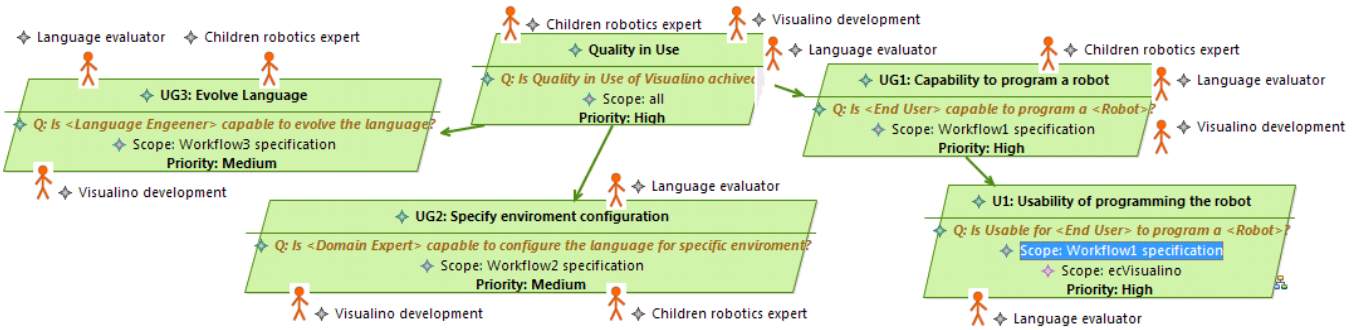

Figure 14: USE-ME Goal Model (from [1])

that evaluators do not take another role in a development project. From this initial hierarchy, the 'User' profile has a 'high' priority for usability evaluation and should be further divided into sub-profiles and characterized by expected knowledge sets and demographics constraints. We can obtain additional information from RDAL *assumptions*. For instance, the System Overview indicates that one sub-profile should be a 'Child' (figure 6) and an environmental assumption captures that the child is expected to speak English (figure 10).

The next step consists of providing the Context Environment of USE-ME. In the DSSL specification, we have context entity instances describing the considered environmental elements (figure 6). These elements, are classified in USE-ME into physical, social and technical *Environmental Variables*. For instance, a 'Web Browser' is a technical environmental variable, and it is further specified with tools which are taken into consideration (e.g. Google Chrome, Safari, etc.). The 'Robot' is a physical variable, and it details a robot version and for instance a different configuration of sensors which can be used. Finally, for instance, a social environment variable can be the 'Country' and should list the countries in which the DSL is distributed.

Finally, the context modelling ends with the creation of USE-ME workflows representing a 'high' level scenario. From the functional System Goals of figure 8, we can distinguish at least three different workflows to be considered, of which the one addressing the 'Programming a robot behaviour' should have a 'high' priority. The scenarios for the workflow can be reused from the operational concepts described by UCM use-case diagrams. USE-ME implies that actors of these scenarios (e.g. User and Web Browser) need to be specified previously as a *User Profile* or *Environmental Variable*.

5.2.2 Goal Modeling. USE-ME divides a root usability goal into sub-goals, which refer to distinct context model parts (e.g. different user profiles, environment elements or workflows), called 'scope'. As we had created three workflows from System Goals, with actors having a different 'User Profile', we had a three corresponding usability goals (figure 14). These goals are prioritized, and one having

a 'High' priority is divided into a sub-goal for which only the evaluator stakeholder is responsible. For this kind of goal it is necessary to create a *Method* which defines the measurable requirements and dependencies which impact the goal evaluation. Usually, such evaluation depends on whether the given functionality is provided, namely it relates to the *Functional Goal* from Utility, with a set of functional requirements, which will support the use case. When specifying the usability evaluation, the scenarios which can be used will depend on the functions that are provided.

One example of defining a metric for the usability goal is by use of GQM (Goal, Question, Metric) analysis [37]. For instance, one *Usability Requirement* can be a **Satisfaction**. In this case the associated question is 'How much is the {User} satisfied with {Gyro}?' and it has predefined 3 metrics: Confidence (self rated confidence score in a Likert scale), Likability (self rated likeability score in a Likert scale) and Learnability (self rated learnability score in a Likert scale). Another common indicator of usability is **Effectiveness**, which addresses the question 'Is the {User} able to correctly implement a given {Use Case}?'. It can be measured, for instance, with a percentage of correctly implemented concepts in {Use Case}.

5.2.3 Evaluation and Report Modeling. The evaluation modelling is performed only when context and goal models are completed. Evaluation specification elements are extracted mostly from existing models and instantiated into real objects. For instance, for one of the Gyro assessments where the *Usability Goal* 'U1' was evaluated, the participants were chosen to be secondary school students, as they have a 'Child' *User Profile*. The evaluation was set up to be a comparison to another DSL designed with a same purpose. To access a background of candidates, questions were directly defined from the profile characterization. The scenarios which were used for learning and testing are selected from the workflow assigned to our usability goal under evaluation. With RDAL we can easily obtain a list of the functional requirements which are satisfied, and indicate which scenarios can be performed for the current version of the language.

After evaluation execution, the Result models are obtained and USE-ME report modeling *Recommendations* are created. These recommendations contain suggestions of new requirements, which are justified by the executed evaluation model. Finally, for the evaluated goal, a success scope is calculated, which contains evaluation results and their impact depending on the size of the evaluated scope, and a real context specification associated with the goal. The recommendations can contain new suggestions for new requirements, or modification of either existing ones or of the associated context. If there are new requirements or goals to be introduced, they should be mapped back to the RDAL models.

6 RELATED WORK

Effective communication with stakeholders is extremely relevant for RE [13]. Indeed, the usability of software engineering visual languages is becoming a hot topic, building on works such as Moody's "Physics of Notations" [25]. This has inspired evaluations on the visual notations of languages such as KAOS [23], *i** [9, 27], or UML [26], and even on the way layout affects model understandability [29, 31, 32, 34]. This concern on usability is also applicable to DSLs. For example, an assessment on cognitive dimensions can be leveraged to increase DSL usability [4]. It is also common to find evaluations on DSLs using a more "traditional" empirical software engineering approach (see, e.g. [17, 19]). A more recent trend has been to directly involve the end users in the design of the DSLs [15, 16, 30, 38]. These and other usability promoting approaches can be integrated in USE-ME. The framework is open to adopting new forms of usability evaluations.

DSLs and Model-Driven Development (MDD) can be used to build frameworks for Requirements Engineering. Examples include building frameworks or derived DSLs for GORE approaches such as KAOS [12], *i** [14, 33], and model transformations between KAOS and *i** [24]. In [41] the authors build a DSL similar to Mind Maps to capture requirements and, from those, automatically derive the corresponding Feature Models[10] for variability analysis.

Kolovos *et al.* identified high-level requirements for DSLs, where usability is considered, but not regarded as a priority [18]. Indirectly, however, usability receives attention as a desirable side-effect of simplicity which, in turn, is a key quality feature for DSLs. However, evidence of using Requirements Engineering techniques and tools during the DSL lifecycle is relatively scarce, especially with usability concerns. The closest works in this direction are on the topic of Domain Engineering, also called Product Line Engineering [10], which consists of reusing domain knowledge to derive new software products. In this case, the DSLs are designed (metamodel for syntax description) and implemented, after capturing the Domain Model (core concepts), to deal with the variability and commonalities of the product's specification.

The design phase of DSLs development can be supported by capturing Domain Ontologies to derive the Language Meta-model [35]. In [40], a framework based on OWL is developed to support the design of the DSL syntax. However, these approaches do not focus on capturing the high-level goals for developing the language, and do not address the usability of the language for the end-user.

7 CONCLUSIONS AND FUTURE WORK

This paper presented a usability-driven requirements engineering approach for DSL development. The RDAL-REMH approach used for embedded systems development with the AADL language for system architectures has been adapted to DSL development by replacing the AADL with the DSSL language, a new DSL that we developed for modeling the DSL under development and its environment. We then provided a mapping between the USE-ME language and the combined RDAL-REMH languages for integrating the USE-ME usability driven development into the RDAL-REMH general RE approach. The approach has been illustrated by the development of the Gyro visual robot programming DSL. To our knowledge, no such comprehensive RE approach has been developed for DSL development, and we expect several benefits from supporting the REMH best practices with models and the integrated USE-ME usability-driven development approach.

Future work will involve completing the implementation of the graphical notation and tools for the languages and implement a view mechanism for the developed mapping between RDAL-REMH and USE-ME. The EMF Views tool [8] is a good candidate for this. Then, a comprehensive evaluation for more complex DSLs than Gyro can be performed to assess the benefits of our approach.

ACKNOWLEDGMENTS

The authors would like to thank the COST Action IC1404 Multi-Paradigm Modeling for Cyber-Physical Systems (MPM4CPS) for the context and partial support to this work, as well as NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MA Project (Grant: FCT/MCTES TUBITAK/0008/2014).

REFERENCES

[1] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2017. Usability Driven DSL development with USE-ME. *Computer Languages, Systems and Structures (ComLan)* (2017). (in press).

[2] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2017. Usability Software Engineering - Modeling Environment (USE-ME 1.1). *Faculdade de Ciências e Tecnologia, Universidade Nova da Lisboa* (2017). https://doi.org/10.5281/ZENODO.345941

[3] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. 2011. How to reach a usable dsl? Moving toward a systematic evaluation. *Electronic Communications of the EASST: 5th Int. Workshop on Multi-paradigm Modeling (MPM 2011)* 50 (2011), 13.

[4] Alan F Blackwell, Margaret M Burnett, and Simon Peyton Jones. 2004. Champagne prototyping: A Research technique for early evaluation of complex end-user programming systems. In *2004 IEEE Symposium on Visual Languages and Human Centric Computing*. IEEE, Rome, Italy, 47–54.

[5] Dominique Blouin. 2017. Report on the Short Term Scientific Mission on 'Combining Modeling Languages to Support Usability-Driven DSL Development with USE-ME'. In *Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)*. European cooperation in science and technology (COST IC1404), 1–15.

[6] Dominique Blouin and Holger Giese. 2016. Combining Requirements, Use Case Maps and AADL Models for Safety-Critical Systems Design. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 266–274.

[7] Dominique Blouin, Eric Senn, and Skander Turki. 2011. Defining an annex language to the architecture analysis and design language for requirements engineering activities support. In *2011 Model-Driven Requirements Engineering Workshop*. IEEE, 11–20.

[8] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. 2015. EMF views: A view mechanism for integrating heterogeneous models. In *International Conference on Conceptual Modeling*. Springer, Stockholm, Sweden, 317–325.

[9] Patrice Caire, Nicolas Genon, Patrick Heymans, and Daniel L Moody. 2013. Visual notation design 2.0: Towards user comprehensible requirements engineering notations. In *RE'13*. IEEE, Rio de Janeiro, Brasil, 115–124.

[10] Krzysztof Czarnecki and Eisenecker Ulrich. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA. 864 pages.

[11] Julien Delange, Peter Feiler, and Ernst Neil. 2016. Incremental life cycle assurance of safety-critical systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

[12] Patrícia Espada, Miguel Goulão, and João Araújo. 2013. A framework to evaluate complexity and completeness of KAOS goal models. In *International Conference on Advanced Information Systems Engineering*. Springer, 562–577.

[13] Daniel Méndez Fernández, S. Wagner, Marcos Kalinowski, M. Felderer, P Mafra, Vetrò, Tayana Conte, M.-T. Christiansson, C. Greer D. Lassenius, T. Männistö, M Nayabi, M. Oivo, B. Penzenstadler, Pfahl Dietmar, R. Prikladnicki, Gunther Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, José Luis de la Vara, and Roel Wieringa. 2016. Naming the Pain in Requirements Engineering - Contemporary Problems, Causes, and Effects in Practice. *Empirical Software Engineering* (August 2016), 1–36.

[14] Catarina Gralha, João Araújo, and Miguel Goulão. 2015. Metrics for measuring complexity and completeness for social goal models. *Information Systems* 53 (2015), 346–362.

[15] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2013. Enabling the collaborative definition of DSMLs. In *International Conference on Advanced Information Systems Engineering*. Springer, 272–287.

[16] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2016. Collaboro: a collaborative (meta) modeling tool. *PeerJ Computer Science* 2 (10 2016), e84.

[17] R B Kieburtz, L McKinney, Jeffrey M Bell, J Hook, A Kotov, J Lewis, D P Oliva, T Sheard, I Smith, and L Walton. 1996. A software engineering experiment in software component generation. In *Proceedings of the 18th international conference on Software engineering (ICSE'1996)*. IEEE Computer Society, 552.

[18] Dimitrios S Kolovos, Richard F Paige, Tim Kelly, and Fiona A C Polack. 2006. Requirements for domain-specific languages. In *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, Vol. 2006. 1,4.

[19] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Varanda João Maria Pereira, Matej Črepinšek, Cruz Daniela Da, and Rangel Pedro Henriques. 2010. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems* 7, 2 (2010), 247–264.

[20] David L Lempia and Steven P Miller. 2009. *Requirements engineering management findings report.* Technical Report. Technical report DOT/FAA/AR-08/34, Federal Aviation Administration.

[21] David L Lempia and Steven P Miller. 2009. Requirements engineering management handbook. *National Technical Information Service (NTIS)* 1 (2009).

[22] Grzegorz Loniewski, Etienne Borde, Dominique Blouin, and Emilio Insfran. 2013. Model-Driven Requirements Engineering for Embedded Systems Development. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 236–243.

[23] Raimundas Matulevičius and Patrick Heymans. 2007. Visually effective goal models using KAOS. In *International Conference on Conceptual Modeling*. Springer, 265–275.

[24] Rui Monteiro, João Araújo, Vasco Amaral, Miguel Goulão, and Pedro Patrício. 2012. Model-driven development for requirements engineering: The case of goal-oriented approaches. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. IEEE, 75–84.

[25] Daniel Moody. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35, 6 (2009), 756–779.

[26] Daniel Moody and Jos van Hillegersberg. 2008. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In *International Conference on Software Language Engineering*. Springer, 16–34.

[27] Daniel L Moody, Patrick Heymans, and Raimundas Matulevičius. 2010. Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. *Requirements Engineering* 15, 2 (2010), 141–175.

[28] David Lorge Parnas and Jan Madey. 1995. Functional documents for computer systems. *Science of Computer Programming* 25, 1 (10 1995), 41–61.

[29] Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. 2010. An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering* 15, 5 (2010), 493–522.

[30] Jesús Sánchez-Cuadrado, Juan De Lara, and Esther Guerra. 2012. Bottom-up meta-modelling: An interactive approach. In *MODELS (Lecture Notes in Computer Science)*, Robert B. France, Jrgen Kazmeier, Ruth Breu, and Colin Atkinson (Eds.), Vol. 7590 LNCS. Springer Berlin Heidelberg, 3–19.

[31] Mafalda Santos, Catarina Gralha, Miguel Goulao, João Araújo, Ana Moreira, and Joao Cambeiro. 2016. What is the impact of bad layout in the understandability of social goal models?. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 206–215.

[32] Bonita Sharif and Jonathan I Maletic. 2010. An eye tracking study on the effects of layout in understanding the role of design patterns. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 1–10.

[33] Lyrene Silva, Ana Moreira, João Araújo, Catarina Gralha, Miguel Goulão, and Vasco Amaral. 2016. Exploring Views for Goal-Oriented Requirements Comprehension. In *Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings 35*. Springer, 149–163.

[34] Harald Störrle. 2011. On the impact of layout quality to understanding UML diagrams. In *VL/HCC, 2011*. IEEE, 135–142.

[35] Robert Tairas, Marjan Mernik, and Jeff Gray. 2008. Using ontologies in the domain analysis of domain-specific languages. In *International Conference on Model Driven Engineering Languages and Systems*. 332–342.

[36] Axel Van Lamsweerde and Emmanuel Letier. 2004. From object orientation to goal orientation: A paradigm shift for requirements engineering. In *Radical Innovations of Software and Systems Engineering in the Future*. Springer, 325–340.

[37] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. 2002. Goal question metric (GQM) approach. *Encyclopedia of Software Engineering* (2002).

[38] Maria Jose Villanueva, Francisco Valverde, and Oscar Pastor. 2014. Involving end-users in the design of a domain-specific language for the genetic domain. In *Information System Development*. Springer, 99–110.

[39] Markus Völter, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, and Wachsmuth. 2013. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform. 558 pages.

[40] Tobias Walter, Fernando Silva Parreiras, and Steffen Staab. 2009. OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In *Model Driven Engineering Languages and Systems SE - 32 (Lecture Notes in Computer Science)*, Andy Schürr and Bran Selic (Eds.), Vol. 5795. Springer Berlin Heidelberg, 408–422.

[41] Fernando Wanderley, Denis Silva da Silveira, João Araujo, and Maria Lencastre. 2012. Generating feature model from creative requirements using model driven design. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*. ACM, 18–25.