

SnapMind: A Framework to Support Consistency and Validation of Model-Based Requirements in Agile Development

Fernando Wanderley
CITI – FCT
Universidade Nova de
Lisboa
Caparica, Portugal
f.wanderley@fct.unl.pt

António Silva
CITI – FCT
Universidade Nova de
Lisboa
Caparica, Portugal
ap.silva@campus.fct.unl.pt

João Araujo
CITI – FCT
Universidade Nova de
Lisboa
Caparica, Portugal
joao.araujo@fct.unl.pt

Denis S. Silveira
PROPAD
Universidade Federal de
Pernambuco – UFPE
Recife, Brasil
dsilveira@ufpe.br

Abstract—Two fundamental principles and values of agile methods are customer satisfaction by rapid delivery of useful software and the improvement of the communication process by continuous stakeholders' involvement. But, how to deal with customers' satisfaction and find a better visualization model at the requirements level (which stakeholders can understand and be involved) in an agile development context? Also, how this visualization model enhancement can guarantee consistency between agile requirements artefacts (e.g., user stories and domain models)? Thus, to answer these questions, this paper presents the *SnapMind* framework. This framework aims to make the requirements modelling process more user-centered, through the definition of a visual requirements language, based on mind maps, model-driven and domain specific language techniques. Moreover, through these techniques, the SnapMind framework focuses on support for consistency between user stories and the domain models using a model animation technique called *snapshots*. The framework was applied to an industrial case study to investigate its feasibility.

Index Terms—Agile Software Requirements; Model-Driven Engineering; Domain-Specific Languages; Mind Map; Snapshots.

I. INTRODUCTION

Several agile methods have been developed and widely adopted to deliver software faster and ensure that the software meets the customer changing needs [1]. All agile approaches share some common principles: improved customer satisfaction; fast adaptation to changing requirements; frequently delivery of working software; and closer collaboration among business people, domain experts and developers [2]. The requirements engineering is the basis of all software products and quality attributes; but requirements modelling and consistency verification between models activities although common to all development methodologies (agile and traditional), still have problems [10,14,15]. According to a Standish Group study [3] about results in agile teams, five out of the eight major factors of project failures are about requirements: incomplete requirements, low customer involvement, unrealistic expectations, changes in requirements and unnecessary requirements are some of the causes. The CHAOS report [3] shows projects with lack of user involvement resulting in poor performance. User collaboration has a major effect on project resolution, be large or small.

Also according to the recent Ambler's survey¹ two main factors for customers' satisfaction and stakeholders' involvement are: (i) reduced engagement of stakeholders in requirements modelling process and; (ii) limited verification and validation activities applied at early phases of software development.

The first factor reported by Ambler's survey – more stakeholders' engagement – is related to the effectiveness of requirements models (and their visual notation) used. According to [15] visualization, at early phases, keeps stakeholders interested and engaged, which is essential to diminish the semantic gap between domain experts and requirements engineers. Recent empirical results reported by Caire *et al.* [4] reinforced the importance of effective communication by requirements models with naïve users. According to their study, the visual notations used by requirements models (e.g.: i*) must improve the semantic transparency, i.e. the clarity, objectivity and meaning of the visual notations adopted, thus reducing the cognitive load. The second factor reported by Ambler is related to verification and validation techniques to check inconsistencies between requirements models (e.g.: user stories and domain models).

As an alternative for mitigating the problems mentioned, this paper presents the *SnapMind* framework. This framework has the following main objectives: (i) providing a visual requirements language, based on mind maps, to represent both user stories and domain models, developed by using model-driven engineering and domain-specific modelling language techniques; (ii) and transforming these requirements models, expressed with a mind map based language, to the USE-tool [5]. The USE tool will generate a set of *snapshots* (objects diagram) as a validation feedback mechanism for the SnapMind framework.

The main contributions of this paper are: verifying inconsistencies between user stories and domain models by *snapshots* generation; support for the validation process, including the definition of a visual and cognitive requirement language. User stories and domain models were designed by the same model language – mind maps – which is key for stakeholders' engagements in requirements modelling in agile process.

¹Agile Initiation Survey 2013 - ambysoft.com/surveys/projectInitiation2013.html.

The remainder of this paper is structured as follows. Section 2 gives some background on User Stories and Domain Model, Snapshots, Mind Map Modelling and Domain-Specific Modelling Language. Section 3 describes the *SnapMind* framework definition and its process. Section 4 applies the framework to a case study considering 3 different situations. Section 5 presents some discussion about the framework. Section 6 reports some related work. Finally, Section 7 draws some conclusions and points out directions for future work.

II. BACKGROUND

A. User Stories

In comparison to the traditional requirements engineering process, the agile requirements process is less document-centric and more user-centred. The agile requirements rely on people's expertise, competency and direct collaboration rather than on rigorous, document centric processes to produce high-quality software [10]. The main focus of agile requirements process is on creating artifacts that have aggregate business value in a simple way [12]. Ambler [10] and Leffingwell [11] emphasize that the effectiveness of agile requirements models is related to the power of knowledge communication among all stakeholders, in a specific domain. User stories are one of the most used models in agile development. The main strength of user stories is in telling the functional behavior of systems. According to [16,17], telling stories about systems helps to ensure that stakeholders share a sufficiently wide view to avoid missing vital aspects of problems space. While user stories focus on describing each user interaction and the respective system response, the acceptance criteria of user story focuses on validating and verifying if the system's behaviour related to the user goal will be implemented in early phases by executable specifications [16,17,18]. Acceptance testing is the process of verifying stories that were developed, such that each story works exactly the way the customer team expects it to work [16,18]. While user stories are intentionally written as short and coarse statements, the stories acceptance (Figure 1) provides the necessary precision to assure that the user story is implemented correctly and covers the relevant functional behavior [16,18,19].

Scenario: As a client I want to rent a car	
Given	a car group with a list of 15 available cars
When	client with driver license with 2 years
Then	rent a car is Valid
When	client with Age equals to 16 years
Then	rent a car is Not Valid

Figure 1. *RentaCar* story acceptance test written in Jbehave² language [19].

B. User Stories and Domain Model

A domain model is a conceptual model of all the topics related to a specific problem of a domain space (Figure 2). It describes the entities, their attributes, relationships, invariants and constraints, which define the domain space [20,21]. According to [20], conceptual models are needed to achieve a common understanding of the system domain among all stakeholders. Loucopoulos [22] defines conceptual modelling

as “the activity of formal describing some aspects of the physical and social world around for purposes of understanding and communication”.

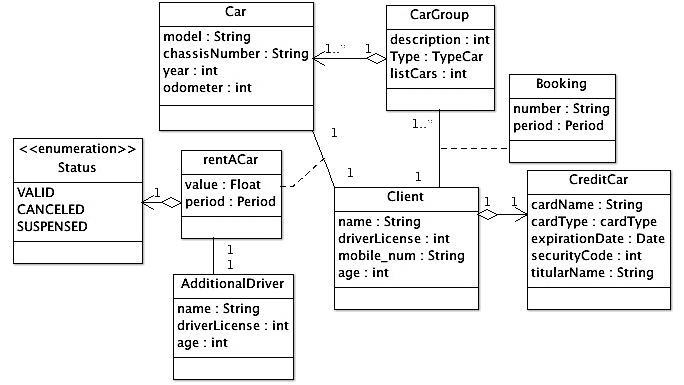


Figure 2. Rent a Car conceptual model.

Fowler [23] affirms that to have information systems executing an expected and appropriate behaviour, it is necessary to have satisfactory domain knowledge among the stakeholders. The structure to represent this domain knowledge is called conceptual schema. Conceptual modeling must precede system design so it must be realised during requirements engineering [20]. There is a direct correlation between user stories and domain models [11,16]. Scenarios are the narrative of system behavior, i.e., the specification of states for some objects defined in the domain model. The user steps relate some object states, which were defined in the domain model (See Table I – Formal Definition).

C. Snapshots

The *snapshot* is a depiction (usually as a drawing) of a set of objects and the values of some of their attributes at a particular point in time [36]. Some authors [36, 37] use snapshots to validate domain models through successive animations to verify that the static associations can represent all the information of interest. It consists of showing how an executable model of a software-based system behaves in response to external events and user inputs. The simulation is visualized on a textual or graphical model representation by highlighting the current model element being executed. In this paper, we will call simulation the execution of a model and animation the visualization of a simulation in some graphical form. Gogolla and Richters [37] presented a formal definition of *snapshots*. We can observe in Table I the “Given” step of acceptance test (Figure 1) and the relation with respective instance objects created in formal definition. In other words, each step from a user story describes (in natural language) a system state. The first step (Given) of our user story relates to *CarGroup* and *Car* objects.

TABLE I. SNAPSHOT GENERATION BY USE LANGUAGE.

User Story Step	Formal Definition	USE Language
Given a car group with a list of 15 available cars .	$\sigma_{CLASS}(CarGroup) = \{cG_1\}$ $\sigma_{CLASS}(Car) = \{c_1\}$ $\sigma_{ATT}(listCars)(cG_1) = 15$ $\sigma_{ASSOC}(has) = \{cG_1, c_1\}$!create cG1:CarGroup !create c1:Car !cG1.listCars = 15 !insert has(cG1,c1)

² Jbehave - <http://jbehave.org/>

D. Mind Mapping Modelling: An User-Centred Approach

A mind map is a diagram used to view, classify and organize concepts, and to generate new ideas in a simple and intuitive way. It is used to connect words, ideas and concepts to a central idea or concept [24]. It is similar to a semantic network, or a cognitive map, but without restrictions on the types of connections used. A mind map is a radial diagram that, through a vocabulary (i.e., set of keywords), may cognitively represent and model a concept or a specific domain. According to Buzan [24], the main benefits of using mind maps are: organization of ideas and concepts, emphasis on the relevant keywords, association between elements in branches, grouping ideas, support visual memory and creativity, and trigger innovation.

Some studies [25,26] emphasize that this kind of representation facilitates the information process, by reducing the cognitive load for absorbing the domain concepts and their goals. Mind maps are used in our approach as a visual requirements representation to improve the learnability (ease of understanding) and to improve communication among all stakeholders. Both academy and industry have been considering techniques and mind map based tools for managing requirements elicitation in agile development. Mahmud reported [27] interesting results collected during an experiment with experts and non-experts using mind maps. Kof *et al.* [28] used the concept of map visualization for requirements tracing. Laplante [29] mentioned that mind maps could be used during eliciting discussion with stakeholders aiming to capture the essential parts (business terms) of problem space. Village *et al.* [30] reports the learnability characteristic of use map visualization in early software engineering design. Alexander *et al.* [24] reports that mind maps can help requirements engineers to remember and make good holistic visualizations about all system details, using a mixing of words, pictures and other rich resources. Sutcliffe *et al.* [31] discuss their experience about user-centred requirements engineering, where mind maps were used to encourage epidemiologists (domain experts) to make more use of visualisation languages.

According to Moody and Heymans [32], the visual notation plays a critical role in requirements engineering (RE), and has dominated RE practice from its earliest beginnings. Visual notations plays a critical role in communicating with end users and customers, as diagrams are believed to convey information more effectively to non-technical people than text. In this paper we adopt a user-centred visual representation based on mental models [33] through Domain-Specific Modelling Languages techniques.

E. Domain-Specific Modelling Languages

Some of the recently proposed Model-Driven Engineering approaches are based on Domain-Specific Languages (DSLs) [34]. A DSL is a language offering expressive power focused on a particular problem domain, such as a specific class of particular or application aspect [34,35]. They offer substantial gains in expressiveness and ease of use compared with general-purpose languages (GPL) in their domain of application. The emergence of domain-specific modeling (DSM) has challenged

the notion of general-purpose modeling (as done with UML). DSM has enabled both end-users and software developers in describing the key characteristics of a system from the perspective of the problem space, without getting overwhelmed by the accidental complexities of the solution space [34, 35]. The DSL anatomy is essentially composed by: (i) abstract syntax, (ii) concrete syntax and (iii) the semantics definition [35]. The abstract syntax describes the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding. The concrete syntax describes specific representation of modeling language, covering encoding and/or visual appearance. And the semantics definition describes the meaning of the elements defined in the language and the meaning of the different ways of combining them. The abstract syntax is performed by metamodeling techniques and well-formed rules. In the concrete syntax development of our visual language for requirements modelling with mind maps it was used the Eugenia³ framework. Eugenia is a tool to facilitate the use of the GMF⁴ framework. GMF is a valid choice for concrete syntax development of graphical DSL(s) [35]. The first step for DSL development is the metamodel definition. The design of the metamodels was defined using Ecore⁵ language. The mind map metamodel (Figure 5) is the base metamodel for model-driven extension (metamodel composition) to derive domain models and user stories editors. So, the contribution of this DSL development is a twofold: (i) make a more active end-user participation in early phases (e.g.: domain modelling) and (ii) transform these requirement models (domain models and user stories) in USE-tool specific files, aiming to generate *snapshots* to check inconsistencies between them.

III. THE SNAPMIND FRAMEWORK

The *SnapMind* framework proposed in this paper is mainly composed of three components: (i) the Domain Modelling Visual Editor, (ii) the User Scenario Visual Editor, and (iii) the USE-tool (Figure 3). Both visual editors are key to make the requirements modelling process more user-centred. They are used to build domain model and user scenario both using adapting mind maps. The idea is to improve the cognitive effectiveness and involvement of domain experts and business specialists (as collaborative modelling resulting in a feedback mechanism). As shown by the diagram in Figure 3, these editors transform the requirements models into USE input artefacts format aiming to check inconsistencies between them.

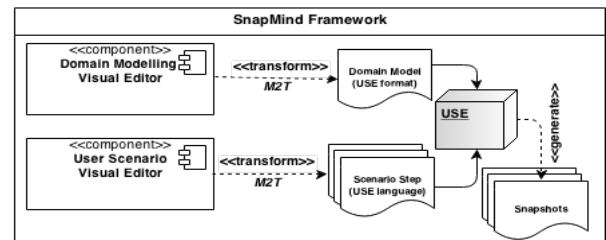


Figure 3. The SnapMind Framework.

³ Eugenia Eclipse Framework - <http://www.eclipse.org/epsilon/doc/eugenia/>

⁴ Graphical Modeling Framework - <http://www.google.com>

⁵ Ecore tools - <http://www.eclipse.org/ecoretools/>

A set of transformation rules are defined using the mind map metamodel, the USE domain model and the USE BNF grammar. The transformation language EGL⁶ was used to implement these rules. Moreover, the *SnapMind* framework also provides static analysis through the USE-tool, as a separated concern of developer's team. This tool receives as input: (i) the domain model in USE format and; (ii) a set of USE script commands, each one related to a step (Given, When and Then) of the user scenario represented by a mind map. The USE tool is used as a black box, i.e., it serves as a filmstrip generator – a filmstrip is a sequence of snapshots-which serves as a mechanism for verification and validation models for developers.

A. The SnapMind Process Overview

The process designed (Figure 4) is composed by three activities. The first activity is the software specification (containing the definition of the domain model and the user story as sub-activities). The second activity is concerned with the validation models (through animation), which involves snapshots generation, executed by the USE tool. The third activity is the consistence verification activity, reported by USE. The Domain Modelling phase of the Specification activity is an iterative and incremental process. First the requirements engineer captures the essential elements of the system domain together with the domain expert or business specialist (end-users) and after that, the requirements engineer refines this conceptual mind map adding specific concerns (e.g. *Enum*, *AssociationClass* and *OCL* as business rules) as reported in our previous work [8]. The User Stories definition is also a sub activity of software specification where the stakeholders write the expected behaviors from the system in natural language, as shown in Figure 11. After the user stories are specified, the requirements engineer represents these scenarios using the visual language editor (Figure 10) and show them to the end-user, for validation purposes, where user feedback is expected.

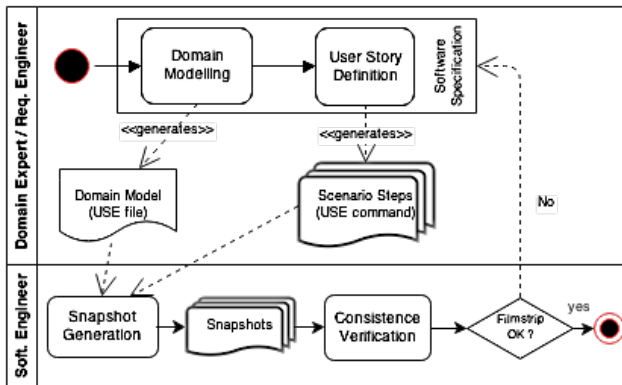


Figure 4. The SnapMind process.

So, the domain modelling and user stories specification activities only generate their output artifacts (for validation) after the end-user feedback. These artifacts are then transformed into USE text files (conceptual model and

scenarios) to be used in snapshots generation. Finally, the Software Engineer will verify the consistency between requirements models, reported by USE.

IV. APPLYING THE SNAPMIND FRAMEWORK

We apply the process to the Audiobus system. The Audiobus is an intelligent system (developed by Mobiciti⁷) that uses geographic information systems, which broadcasts digital surround sound for urban transports; customize messages and contents exclusively developed for the surrounding area, providing personalized advertising in each vehicle. Figure 6 defines the conceptual model of the Audiobus system. Then, to apply our framework we request to the Mobiciti domain expert a user story, which had a high business value for him. This scenario specifies how to broadcast in a specific area with georeferenced positions. In other words, the expectation of the Mobiciti domain expert is when the bus is on a specific position (e.g. 10m before of the Point of Interest) the bus must change the broadcast to a specific advertising on that Point of Interest. This user story is represented in Figure 11. Thus, based on this scenario, we describe 3 situations. Also, when developing the Domain Model, the Mobiciti Domain Expert gave us a business rule (domain constraint) that we mapped into an OCL USE Rule and inserted in our tool. The rule stated that the volume of the mobile phone must be between 0.0 to 1.0 decibels.

A. Domain Modelling

The domain modeling activity consists of the conceptual analysis of the domain of an information system and generating an artifact of this conceptual model in USE format. This conceptual analysis is performed with the help of an editor created (described below) from mind maps, with the purpose of assisting cognitively non-specialist users of IT. So, once defined the base mind map metamodel and their derivations for building DSLs, another important factor is the definition of the concrete syntax (the visual notations).

According to [34] the success of the development of a graphical language depends on how the visual notation fits the domain. Thus, these editors were built taking into account adaptability, where visual notations are defined with end-user collaboration (i.e., participatory design [38]) through multiple GMF extensions. In other words, we have default visual notations to represent an entity concept, using the Domain Modelling editor whose model elements are listed in Table IV. Basically, an *entity* must be related to a concept of the end-user domain. So, we suggest that the definition of the visual notation that represents a specific concept is conducted with the help of the domain expert. Thereby, we are improving the end-user participation in the development process of our graphical language through the Physics of Notations theory [39]. It considers semiotic clarity (there should be a 1:1 correspondence between concepts and graphical symbols), perceptual discriminability (different symbols should be clearly distinguishable from each other) and semantics transparency

⁶ Epsilon Generation Language - <http://www.eclipse.org/epsilon/doc/egl/>

⁷ Mobiciti company – www.mobiciti.com

(use visual representations whose appearance suggests their meaning).

We use the participatory design activity [4,38] to define collaboratively (with the Mobciti Designers) the symbols related to the domain concepts (Table II) from an industry case study (the Audiobus system developed by Mobciti). According to the DSL development process, these symbols represent the concrete syntax of our language for Domain Modelling and User Scenario editors.

TABLE II. VISUAL NOTATIONS FOR REQUIREMENT VISUAL LANGUAGES.

Domain Model Concepts	Default Visual Notations	Audiobus Concepts	Domain-aware Visual Notations
RootNode		Bus (extends Entity)	
Entity		Broadcast (extends Entity)	
AssociationEntity		BusStop (extends Entity)	
Enum		MobileDevice (extends Entity)	
Attribute and Constant	<<no icons>>	Coordinates (extends Entity)	

Domain Modelling Visual Editor. This component is an eclipse-based editor. The first activity of development of the domain-modelling editor was the metamodel definition. The metamodel was defined by specializing the mind map metamodel. We extend the mind map metamodel by inheritance strategy aiming to improve the modularity of the metamodel. We can observe in the metamodel of Figure 7 that the domain model elements $\{Entity, Attribute, AssociationEntity, Enum, \text{and } Constant\}$ extend the main elements of the mind map metamodel. For example, the *Entity* element <<extends>> the *Node* element from Mind Map. The metamodel definition is related to abstract syntax definition of our DSML. To improve the semantics of our language, it was needed to define well-formed rules. These rules were implemented using EVL⁸ from Epsilon tools. One of the rules developed to improve the semantics of domain modelling language was “the *Attributes* elements only have one link to a parent *Entity*”. Part of the respective code (EVL) is shown in Figure 5.

```

context MindMapModel {
constraint AttributeCanOnlyPointToEntity {
check { var n = self.nodes.select(n:Node |
isAttribute(n.type()) and self.edges.select(c:Edge | c.target
=n and(isEntity(c.source.type()))).isEmpty();
return n.isEmpty(); }
message : 'Attributes can only link to Entities' } }

```

Figure 5. EVL rule for Domain Models

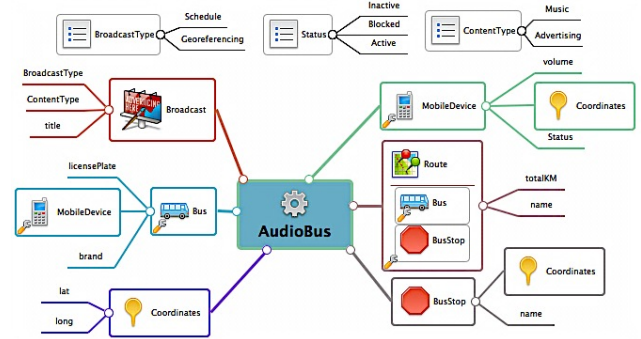


Figure 6. Visual Editor for Domain Modelling (Audiobus System).

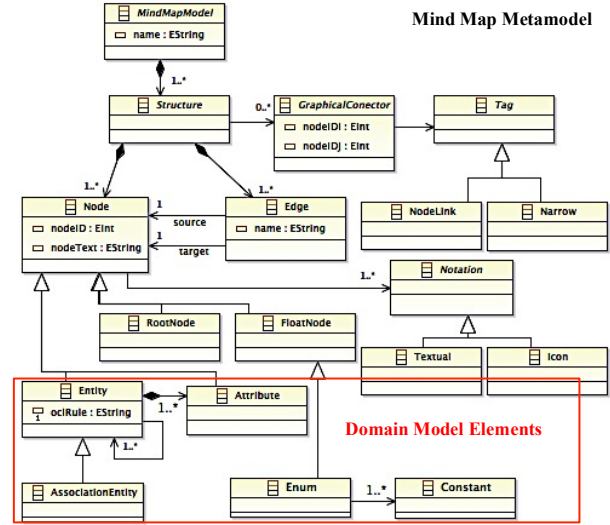


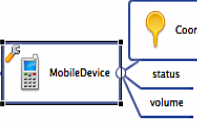
Figure 7. Metamodel for Domain Model based on Mind Map.

As a result of metamodeling definition, we have the abstract syntax designed for Domain Modelling Visual editor, with well-defined rules and the graphical representation (concrete syntax), which has the appearance shown in Figure 6.

Generation of USE’s Domain Model. One of the most important features of this editor is the transformation of the domain model (represented using our mind map based language) into the conceptual model in the USE language (see Table III). The *MobileDevice* is a *Node* defined as an *Entity*, which has *Coordinates*, *status* and *volume* as their attributes (child nodes). So, the *MobileDevice* node is transformed into a class of USE conceptual model. The *Coordinates* node was defined as another entity (see Figure 6), so, it will be transformed into an attribute of *MobileDevice* with data type *Coordinates*.

⁸ Epsilon Validation Language - <http://www.eclipse.org/epsilon/doc/evl/>

TABLE III. TRANSFORMATION OF A DOMAIN MODEL INTO USE FORMAT.

Domain Model Editor	Output File (USE format)
	<pre> class MobileDevice attributes coordinates:Coordinates status : Status volume : Real end constraints context md1:MobileDevice inv deviceVolume:MobileDevice.allInstances->forAll(md md.volume <= 1.0 and md.volume >= 0.0) </pre>

Also, since we are mapping domain knowledge, we support a way to add domain constraints with OCL. Each Entity can add an OCL invariant. Those rules are also mapped to the USE Script. Since invariants state domain constraints of an attribute, by stating a value restriction we can infer the type of the attribute. By doing so, we can have the four basic types implied (*String*, *Real*, *Integer*, *Boolean*).

Note that, our tool does not validate the OCL syntax, which is a concern for the user responsible for editing the diagram. We parsed the invariant rules by searching *relational operators* ("=" | ">" | "<" | ">=" | "<=" | "<>") and check the element that proceeds it. If no match is found, the type of the attribute will set the default value, *String*. Besides a direct transformation to the USE Tool, our tool has some helping functionalities for domain model design. For instance, if an attribute has the same name of an existing *Enum* (e.g. the *status* attribute of *MobileDevice*), it implies that the same attribute has its type equal to the *Enum*. Furthermore, one aspect of the USE's conceptual model is the relationships between all objects in the domain.

In order to define all the relationships between Entities declared in the Domain Modelling editor, we define an algorithm that starts from the *RootNode* and creates an association named "*controls i*", where *i* counts processed entities. Regarding Entities, the ones who have other Entities as an attribute, like the example presented in Table V (*MobileDevice* and *Coordinates*), the algorithm works as follows: for each entity that is composed by other entities we create an association named "*has i*", where *i* is the current counter of processed entities. For every Entity parsed the *i* is incremented.

This information about the relationship generated is persisted in a CSV file. Each line of the file has an entry with *sourceEntityName*, *targetEntityName*, *AssociationName*. The file is created during the transformation to a USE domain model. The Figure 8 shows the result of the transformation to use domain model with *controls* and *has* relationships, creations of *enums*, and data type entities. However, the model shown in Figure 8 still needs refinements in order to improve the quality of conceptual schema (e.g., association refinements, data types).

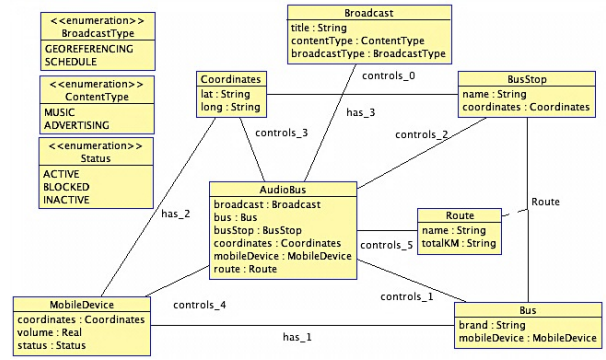


Figure 8. Conceptual Model of Audiobus in USE file.

B. User Story Definition

This is the activity where the Requirements Engineer represents the user story in visual language (using the User Story Visual Editor) and also expects feedback from the user, to check if the representation has not lost any semantics of expected behavior. This visual representation is important because, in addition to having a more cognitive communication with the user, this language aims to transform every step of the story into the USE-tool scripts for snapshots generation.

User Story Visual Editor. The development of this editor is analogous to the Domain Modelling editor. The abstract syntax definition through metamodeling techniques extends the same base (mind map) metamodel using inheritance strategy (Figure 9). We can observe in the metamodel of Figure 9 that the main elements to represent a user story (in an acceptance test) are the set elements {*Step* (Given, When, Then), *Entity*, *Attribute*, *Value*}, which extend the main elements of mind map metamodel.

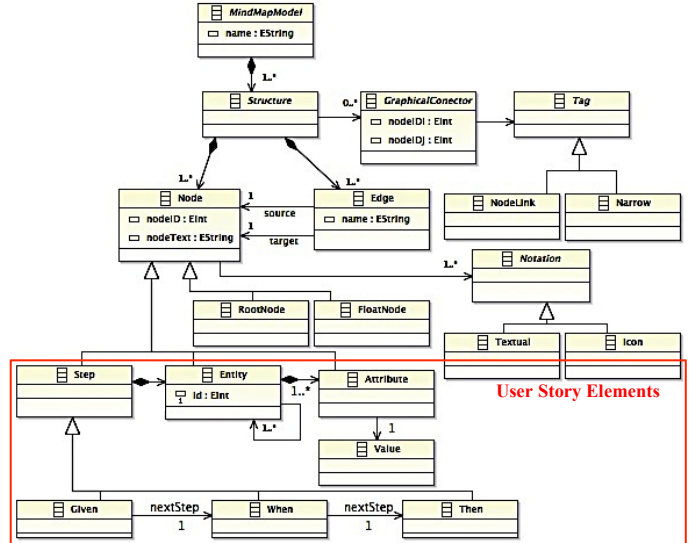


Figure 9. Metamodel for User Story based on Mind Map.

For example, the abstract element Step <<extends>> Node element from the mind map metamodel. Each Step defined contains many Entities; each Entity is composed by many Attributes where each Attribute has one and only one Value.

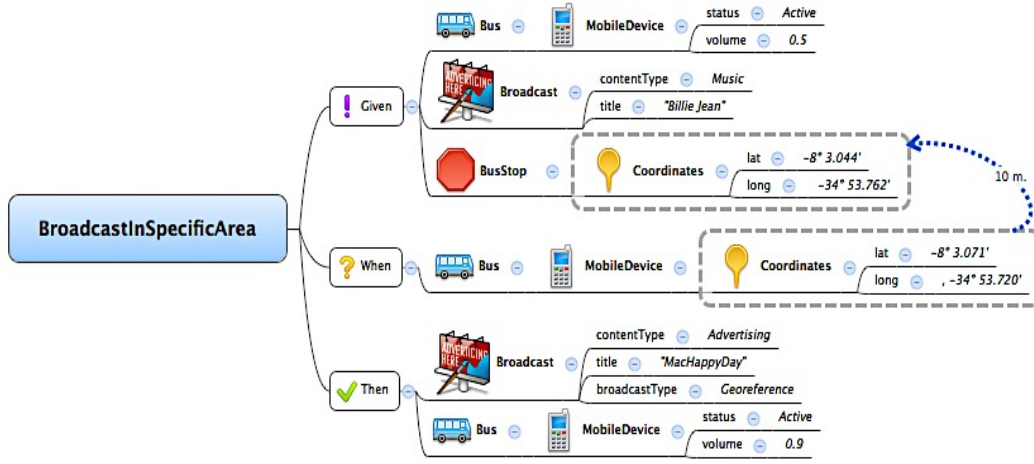


Figure 10. Visual Editor for User Story.

The metamodel implies the order and cardinality of Steps. The first Step is *Given*, followed by *When* and finally by the *Then* step node. This order constraint is reinforced with rules. In this sense, well-formed rules were defined to improve the semantics of our visual language for the User Story Editor in EVL. This rule specifies the implication of the order of Steps defined for a User Story, which will always have a *Given*, *When* and *Then* (in this order). Eventually, it will be necessary to create another scenario, if the user story may need another *When* and *Then* steps for the same *Given*, as the scenario shown in Figure 1. In SnapMind, user stories represent a specific state of the system at a given time (t). For this reason, the actual stage of User Scenario Visual Editor just represent only *Given*, *When* and *Then* steps. As a result of the metamodel definition with EVL rules and the concrete syntax (Table II), a model built using the visual editor for User Story has the appearance as shown in Figure 10. This is a visual representation for a User Story depicted in Figure 11. The tool does not yet support the arrow and enclosing rectangle.

```
!broadcast.title := 'Billie Jean'
!create busStop : BusStop
!create coordinates : Coordinates
!coordinates.lat := '-8 3.044'
!coordinates.long := '-34 53.762'
!busStop.coordinates := coordinates
!insert (ctrlAudioBus, bus) into controls_1
!insert (ctrlAudioBus, broadcast) into controls_0
!insert (ctrlAudioBus, busStop) into controls_2
!insert (bus, mobileDevice) into has_1
!insert (busStop, coordinates) into has_3
```

As we can see, the first line of the command creates a controller object (*ctrlAudioBus*). It means that, each user scenario will always create an object of type controller, which knows the states (and their transitions) of related objects. Notice that each entity is created and then they are associated with each other via the associations created previously in the domain model (Figure 6). We have defined two essential components of our framework (i) the visual editors for domain modelling (Figure 6) and (ii) the user stories specifications (Figure 10). Another contribution is the generation of output USE files (Table III and Table IV).

Given a bus with the mobile device in active status and playing the music "Bille Jean" with volume of 0.5 decibels and the position of a bus stop with coordinates LatX -8° 3.004' and LogY -34° 53.762'
When the bus is at 10 meters of this point of interest LatX -8° 3.071' and LogY -34° 53.720'
Then this bus will play the "Mac Happy Day" broadcast advertising with volume of 0.9 decibels.

Figure 11. Audiobus scenario for Broadcasting in a Specific Area.

Generation of Stories Scripts (USE Commands). Besides increasing the cognitive visualization of a user story, this editor transforms each step (Given, When and Then) in a USE command file, which has the specification for *snapshots* generation. Table IV shows the transformation from Given Step of the User Story in Figure 10 into USE file commands.

TABLE IV. TRANSFORMATION OF GIVEN STEP IN USE COMMAND.

```
!create ctrlAudioBus : AudioBus
!create bus : Bus
!create mobileDevice : MobileDevice
!mobileDevice.status := #ACTIVE
!mobileDevice.volume := 0.5
!bus.mobileDevice := mobileDevice
!create broadcast : Broadcast
!broadcast.contentType := #MUSIC
```

C. Snapshots Generation and Verification

This activity tries to verify inconsistencies (in the "Filmstrip OK" decision point of the process); it is related to three possible situations for analysis: (i) a complete *Filmstrip* animation – a filmstrip is a set of *snapshots* – where in this case, if the animation is complete for each scenario, it indicates that the user story is consistent with the domain model; (ii) an incomplete *Filmstrip* animation – in this case, USE generates a snapshot (but not a complete *Filmstrip*) indicating that some business rule (expressed in OCL) is violated and; (iii) there is no snapshot – in this case USE indicates that some information in the user scenario (entity or attribute) is not defined in domain model.

D. Situation I: A Complete Filmstrip Animation

The first case is a successful feedback analysis about the user story specified in Figure 11, (represented in visual language of Figure 10). In this case, we will have a complete *Filmstrip* with three snapshots (*Given*, *When* and *Then* Snapshots). The first *Snapshot* represents the *Given* step

(Figure 12). The *ctrlAudioBus* object represents the controller of the scenario, and in this situation, it has three main objects: *broadcast*, *busStop* and *bus*. Now, each created object needs to be instantiated with the values that represent the *Given Step*. In this case, the *busStop* has *coordinates* (latitude and longitude); and the *bus* has a *mobileDevice* (status active and volume equals to 0.5).

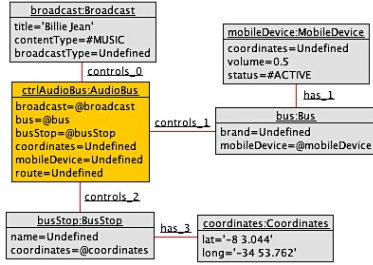


Figure 12. Given Snapshot.

In the second step, the *When* step (Figure 13), a new object (*mobileCoordinates*) is created and associated to the object *mobileDevice*. This represents the instant of the system when the *mobileDevice* has certain coordinates. This represents the instant when the bus is at 10 meters of the bus stop.

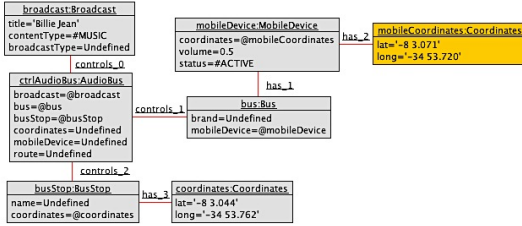


Figure 13. When Snapshot.

The final step, the *Then* step (Figure 14), represents the final need of the customer and in this case he observes the broadcast to change. To do so, the *broadcast* object changes his title to "Mac Happy Day" and the attribute *volume* of the object *mobileDevice* is set to 0.9.

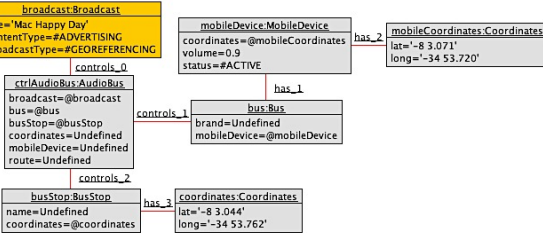


Figure 14. Then Snapshot.

E. Situation II: An Incomplete Filmstrip Animation

In this case, the scenario is the same as the previous one (Figure 10) only with a small change. According to Figure 15 the attribute *volume* of the object *MobileDevice* is set to 1.5. That is a clear violation of the OCL Rule stated previously that sets the *volume* between 0.0 and 1.0.

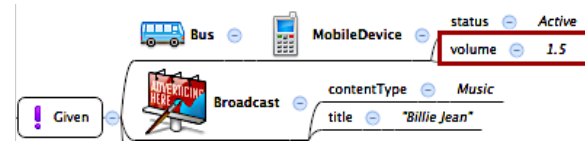


Figure 15. Given step with OCL error.

When we generated the *Given Snapshot*, the USE Tool will generate a snapshot with a failed alert constraint, making it impossible to complete the animation (see Figure 16).

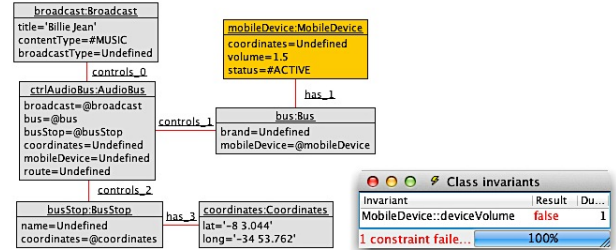


Figure 16. A Snapshot with OCL error.

After this feedback, the software engineer returns to the software specification activity and decides if this behavior (scenario) was expected and approved by the end-user and also check with him/her the business rules, anticipating future test efforts.

F. Situation III: No Filmstrip Animation

In this case the end-user wrote in their scenario some information (entity or attribute) that was not defined in the domain model (Figure 6). Observe the Given step of Figure 17. We can verify that the domain expert specified a Bus with 'totalSeats' information, in this case, not defined in domain model. So, USE will generate the feedback information shown in Figure 18.

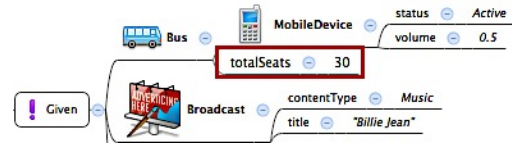


Figure 17. Given step with additional information.

```
use> !bus.totalSeats := 30
<input>:1:0: Class 'Bus' does not have an attribute 'totalSeats'.
```

Figure 18. The feedback information of USE.

Thus, the software engineer will confirm if this information (total of Seats) is missing in the domain model or if it is additional information (unnecessary), described in the respective user story. So, either we reduce test efforts in test case generation with irrelevant information or refine the domain model that represent a valid artifact for code generation or database schema generation.

V. DISCUSSION

The application of *SnapMind* framework to an industrial case study showed us initial *insights* and valuable feedbacks. The first one was the active collaboration of the most important stakeholder – the Mobiciti domain expert – when he wrote the scenario specifying the broadcasting in a specific area. After that, one of the paper authors (as a requirements engineer) represents the user scenario in the User Story Visual Editor

(Figure 10) and presented to the Domain Expert. The result was positive, as he had understood the scenario presented in the visual language. The second feedback was from the development's team: when we generated the three situations of *filmstrips*, we provided rich information about consistency of the requirements models (user scenario and domain models). The three situations (*filmstrips*) provided important feedbacks: (i) in the first case when the developer saw a complete animation, it was an indication of that user scenario was consistent with the conceptual model; (ii) in the second case, when USE informed that one constraint failed, the developer felt the need to check with the requirements engineer if this value (of Mobile Device volume) was correct with expected behavior of Mobciti domain expert and; (iii) in the third situation, USE showed an information (total of seats) that was not designed in the domain model. In this case the developer checked with the requirements engineer and Mobciti domain expert if this information was necessary. From that, the requirements engineer would be able to refine the domain. From this feedback mechanism they could generate code and database schema more consistently. Therefore, with this process there can be an initial verification mechanism informing if the user stories were specified in consistent way.

VI. RELATED WORK

The validation of requirements models through animation model technique is not a novel approach. Heitmeyer [40] proposed the SCR toolset aiming to provide a tabular notation for specifying requirements and a set of "light-weight" tools that detect several classes of errors automatically. Glinz [41] addressed the (inter) consistency verification using a semi-formal approach and systematic cross-referencing corresponding information between class diagrams and scenarios. Trong *et al.* [42] presented the UMLAnt Eclipse plugin that can animate and test the behavior of UML models. This UMLAnt is a set of tools (e.g. JUnit, EMF and Java-like Action Language - JAL). Gogolla *et al.* [37] defined the USE tool, an UML-based specification environment for validation of class diagrams using UML and OCL. Yu *et al.*, [43] presented the SUDA method (Scenario-based static analysis of UML class models). This approach transforms a class diagram into a Snapshot Model. The proposed Snapshot Model is a UML sequence diagram annotated with descriptions of operation effects. The operation effects are described using an action language – JAL. Clark, [44] proposed the model based functional testing using pattern directed filmstrips. Our SnapMind work is very related to example-based modelling/meta-modelling approaches [45,46,47,48]. These approaches reinforces the user engagements using "sketch-base" software modelling aiming to capture business elements through an interactive and iterative alternative to meta-model construction enabling the specification of example model fragments by domain experts, with the possibility of using informational drawing tools.

These related approaches reinforce the continuous need of verification and validation in requirements specification. Some of these works argues that formal modelling improves the correctness verification, other works point out the definition of

semantic action language to specify scenarios, while others presented a metamodel definition for a Snapshot Model providing the *filmstrip* generation. Compared to these approaches, our contribution is to give a better visualization to represent validations by using visual user-centred language to engage the stakeholders and facilitating participatory design. Thus, the *SnapMind* framework strengthens the importance of validation and verification process, but focusing on improvement of users collaboration in an agile process, through visual language definition based on mind maps. In this sense, the USE Tool was chosen for verification purposes, encapsulating all the validation process. Our approach differs from the others, because we created user-centred languages where the validation tools are seen as black boxes receiving as input the visual specifications created by our languages. Starting from a participatory design with all the stakeholders, specifications are created and then serve as input for the USE Tool as a formal validation component. This work highlights the importance of the understanding of the stakeholders in the visualization of requirements modelling.

VII. CONCLUSION

Tool-supported rigorous analysis of requirements models can enhance the ability of software engineers to identify potentially costly design problems earlier. Correcting design problems early also reduces effort spent on implementing faulty designs. Our work is on developing a lightweight technique for rigorously analyzing requirements-level models (domain model and user stories) in agile context.

Thus, this paper proposed user-centered visual requirements models, for domain model specification and behavioral scenarios, both based on mind maps, in agile context. This requirements language aims to improve communication, using organization of concepts and domain specific visual notations. Also, we proposed a support to validate those same scenarios against the created domain model. In order to perform such verification we recurred to the USE tool. The requirements models are transformed to the USE tool and snapshots are generated as a feedback mechanism.

In this context, the first contribution consists of two user-centered languages aiming to improve stakeholder comprehension and engagement in the requirements modelling process. Both languages were defined extending the mind map metamodel, to guarantee that the cognitive factors were not lost, providing another contribution. The third contribution relates to the validation aspects related to the domain specific visual notations models and the possibility to perform verifications between them using snapshots.

As future work we will define an automatic generation of acceptance tests based on scenarios. Finally, we are planning to define a protocol of a controlled experiment to verify the actual cognitive effect and the coverage of code generation with both BDD API(s) and with the SnapMind framework.

ACKNOWLEDGMENTS

The authors would like to acknowledge - CNPq – Ciências sem Fronteiras CITI/PEST-OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) 2011/2012) for the

financial support for this work. We also want to thank Mobciti Company for providing the case study.

REFERENCES

- [1] D. West and T. Grant. Agile development: Mainstream adoption has changed agility – trends in Real-World adoption of agile methods. Forrester Research, 2010.
- [2] K. Beck. Manifesto for agile software development. <http://agilemanifesto.org/>. 2001.
- [3] Standish Group - "The Chaos Manifesto 2013. Think Big, Act Small"- <http://versionone.com/assets/img/files/> [Accessed in Feb'14].
- [4] P. Caire, N. Genon, P. Heymans, D. L. Moody, "Visual notation design 2.0: towards user comprehensible requirements engineering notations," 2013.
- [5] M. Gogolla, F. Büttner, M. Richters, "USE: A UML-based specification environment for validating UML and OCL," Science of Computer Programming, 2007.
- [6] F. Wanderley, D. Silveira, "A Framework to Diminish the Gap between the Business Specialist and the Software Designer", QUATIC 2012, Lisbon, Portugal, 2012.
- [7] F. Wanderley, D. Silveira, J. Araujo, and Maria Lencastre. "Generating Feature Model from Creative Requirements using Model Driven Design", (SPLC '12), 2012.
- [8] F. Wanderley, D. Silveira, J. Araujo and A. Moreira. "Transforming Creative Requirements into Conceptual Models", RCIS, France, Paris, 2013.
- [9] F. Wanderley, and J. Araujo. "Generating Goal-Oriented Models from Creative Requirements using Model Driven Engineering", (MoDRE) @ RE, 2013.
- [10] S. Ambler "Agile Modelling: Effective Practices for eXtreme Programming and the Unified Process." ISBN: 0-471-20282-7, Wiley Computer Publishing, 2002.
- [11] D. Leffingwell "Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise" ISBN: 0-321-63584-1, Pearson Education 2011.
- [12] M. Daneva, E. van der Veen, C. Amrit, S. Ghaizas, K. Sikkil, R. Kumar, N. Ajmerib, U. Ramteerthkarb, R. Wieringa, "Agile requirements prioritization in large-scale system project: an empirical study", J of Syst. & Software, 86(5) 2013.
- [13] G. Kotonya, I. Sommerville, "Requirements Engineering: Processes and Techniques" John Wiley & Sons, 1998.
- [14] K. Wieggers, J. Beatty. "Software Requirements" Microsoft Press 2013.
- [15] J. Beatty and A. Chen "Visual Models for Software Requirements" Redmond, WA: Microsoft Press. ISBN 978-0-7356-6772-3. 2012.
- [16] I. Alexander and N. Maiden, "Scenarios, Stories and Use Cases: Through the Systems Development Life Cycle", John Wiley & Sons - ISBN 0-470-86194-0, 2004.
- [17] M. Cohn, "User Stories Applied", ISBN 0-321-20568-5, 2004.
- [18] G., Brian H., Pauline van. User acceptance testing: a step-by-step guide. BCS Learning & Development Limited. ISBN 9781780171678. 2013.
- [19] D. North, "Behavior-Driven Development" - <http://dannorth.net/introducing-bdd>.
- [20] Olivé, A. "Conceptual Modeling of Information Systems", Springer, 2007
- [21] S. Ambler. "Agile Model-Driven Development with UML 2.0", Cambridge. 2004.
- [22] P. Loucopoulos, R. Zicari, "Conceptual Modeling, Databases and CASE: An Integrated View of Information System Development", Wiley, 1992.
- [23] M. Fowler. "Patterns of Enterprise Application Architecture", Ad. Wesley, 2002.
- [24] T. Buzan, "The Mind Map Book", BBC Active, 2003.
- [25] R. Downs, D. Stea "Image & Envi: Cognitive Mapping and Spatial Behavior", 1973.
- [26] R. M. Kitchin. "Maps Cognitive: What are and why study them?" Psy.Journal, 1994.
- [27] I. Mahmud "Mind-mapping: An Effective Technique to Facilitate Requirements Engineering in Agile Software Development" (ICCIT) 2011.
- [28] L. Kof, R. Gacitua, M. Rouncefield, P. & Sawyer. "Concept mapping as a means of requirements tracing" (MARK) 2010.
- [29] P. Laplante. "What Every Engineer Should Know about Soft. Engineering". 2007
- [30] J. Village, F. Salustri, P. Neumann. "Cognitive mapping: Revealing the links between human factors and strategic goals in organizations" Journal Ergonomics. 2013.
- [31] A. Sutcliffe, S. Thew, P. Jarvis. "Experience with user-centred requirements engineering". Requirements Engineering Journal vol. 16 - Springer Editions. 2011.
- [32] D. L. Moody, P. Heymans, R. Matulevicius, "Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax." 2009.
- [33] M. J. Davidson, et al., "Mental models and usability", Depaul Univ., Chicago, 1999.
- [34] M. Fowler "Domain-Specific Languages" Addison Wiley, 2010.
- [35] M. Voelter "DSL Engineering" (<http://dslbook.org/>), 2013.
- [36] D'Souza, D.F. Wills, "Objects, Components and Frameworks with UML: The Catalysis Approach", Addison-Wesley, 1999
- [37] M. Gogolla, J. Bohling, and M. Richters, "Validating UML and OCL models in USE by automatic snapshot generation," Software & Systems Modeling, vol. 4. 2005.
- [38] K. Bødker, F. Kensing, J. Simonsen, "Participatory IT design: Designing for business and workplace realities." Cambridge, USA: MIT Press. 2004.
- [39] D. L. Moody, "The 'Physics' of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," IEEE Transactions. 2009.
- [40] C. Heitmeyer, J. Kirby, B. Labaw, R. Bharadwaj, "SCR: A toolset for specifying and analyzing software requirements" 1998.
- [41] M. Glinz, "A lightweight approach to consis. of scenarios and class models," 2000.
- [42] T. D. Trong, S. Ghosh, R. B. France, M. Hamilton, and B. Wilkins, "UMLAnT: an Eclipse plugin for animating and testing UML designs," 2005.
- [43] L. Yu, R. France, I. Ray, W. Sun, "Systematic scenario-based analysis of uml design class models," 2012.
- [44] T. Clark, "Model based functional testing using pattern directed filmstrips," 2009.
- [45] K. Bak, D. Zayan, K. Czarnecki, M. Antkiewicz, Z. Diskin, A. Wasowski, and D. Rayside. Example-driven modeling: model = abstractions + examples. In ICSE. 2013
- [46] López-Fernández, Sánchez Cuadrado, Guerra, de Lara. 2013. "Example-driven meta-model development". Software and Systems Modeling (Springer). In press.
- [47] Sánchez Cuadrado, de Lara, Guerra: Bottom-Up Meta Modelling: An Interactive Approach. MODELS 2012.
- [48] D. Wüest and M. Glinz. Flexible sketch-based requirements modeling. In REFSQ, volume 6606 of LNCS. Springer, 2011.