Universidade Federal do Espírito Santo

Bernardo Ferreira Bastos Braga

Modeling Stories for Conceptual Model Validation

Vitória - ES, Brazil
March, 2016

Universidade Federal do Espírito Santo

Bernardo Ferreira Bastos Braga

Modeling Stories for Conceptual Model Validation

Dissertação apresentada ao curso de Mestrado em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Mestre em Informática. Orientador: Prof. Dr. João Paulo Andrade Almeida

Vitória - ES, Brazil
March, 2016

Acknowledgements

First of all, I'd like to thank my family for all the love and support they gave me. My mother Ruth and her husband João, my father Antonio Carlos and his wife Rozeli and my brother Pedro. They have been with me most of my life and their involvement in my upbringing was essential to defining my character, specially my love for all types of information and knowledge. Likewise, my friends have immensely helped me overcoming my troubles and limitations and will probably never realize how important they have been and how much they have affected me. I've had the pleasure of loving large number of friends through my life and listing the names of a few seems unfair to me.

I'm also grateful to my peers in NEMO, friends and teachers from the Federal University of Espírito Santo for their participation in my academic experience. In particular, I'd like to thank my advisor and friend João Paulo Andrade Almeida for everything we've been through. He is a standard for excellence and competence and my role model for being a competent researcher. I also must mention Giancarlo Guizzardi for enlightening me with many fruitful discussions, his work and for being an endless source of related work.

Finally, I thank and dedicate this work to my loving fiancée Nayara Tognere. She was there for me through the worst in the journey of writing this thesis; through the stress and despair, she supported me and kept me going.

This work was supported through a grant by the Brazilian Research Funding Agency CNPq.

Table of Contents

Table of figures

Abstract

Conceptual modeling is a challenging activity and assessing the quality of conceptual models is key to ensure that they may be used effectively as a basis for understanding, agreement and construction of information systems.

A model can be assessed for different types of model quality and in this work we focus on the *accuracy* of an ontology-based conceptual model in characterizing the conceptualization it is supposed to represent. Validating the accuracy of a model involves understanding the admissible worlds states implied by the model and comparing that to the world states deemed admissible in the domain conceptualization.

Previous efforts towards ontology-based conceptual model validation have created a model simulator that allows modelers to be confronted with the consequences of their modeling decisions. The model simulator generates sequences of snapshots of model instances, revealing the dynamics of object creation, change and destruction. Even though these efforts contribute to model assessment, they can be hard to understand and use and this work improves the approach using a mix of informal and formal storytelling.

Stories have always been used as means of communicating complex affairs and we argue that they may be used effectively to assess models and reveal modeling decisions. This dissertation proposes an approach to assess conceptual models by creating narratives about a subject domain. These narratives exemplify how concepts of the conceptual model are employed in context. To use them in the existing model simulator, the natural language narratives are formalized as abstract stories using a specification language we define. These abstract stories are then used to guide the model simulation, generating instance diagrams.

The natural language narrative is used to provide an intuitive understanding of the meaning of concepts. Comparing Natural Language Narratives to object diagrams that show the instantiation of the formal model allows one to understand how concepts are formalized. Contrasting these guided simulations with the intended conceptualization is the basis for model assessment in this approach.

Resumo

Modelagem Conceitual é uma atividade desafiadora e avaliar a qualidade de modelos conceituais é chave para garantir que possam ser usados efetivamente como base para compreensão, acordo e desenvolvimento de sistemas de informação. Modelos podem ser avaliados com relação a diferentes critérios de qualidade e neste trabalho focamos na acurácia de modelos conceituais baseados em ontologias em caracterizar as conceituações que visam representar. Validar a acurácia de um modelo envolve entender os mundos admissíveis que são implícitos a ele e sua correspondência com os mundos admissíveis de acordo com uma conceituação de domínio.

Esforços anteriores para validação de modelos conceituais baseados em ontologia deram origem a um simulador de modelos que permite a modeladores ser confrontado com as consequências de suas decisões de modelagem. Esse simulador de modelos gera sequências de snapshots da instanciação de modelos, revelando a dinâmica da criação, mudança e destruição de objetos. Ainda que esses esforços contribuam para avaliação de modelos, eles podem ser difíceis de compreender e usar e este trabalho melhora a abordagem existente usando um misto de histórias formais e informais.

Histórias sempre foram usadas como meio de comunicar ideias complexas e nós argumentamos que podem ser usadas efetivamente para avaliar modelos e revelar escolhas de modelagem. Esta dissertação propõe uma abordagem para avaliar modelos conceituais criando narrativas a respeito de um domínio de discurso. Essas narrativas exemplificam como conceitos de um modelo conceitual são empregados em seu contexto real. Para usá-las no simulador de modelos existente, as narrativas em linguagem natural são formalizadas como histórias abstratas usando a linguagem de especificação que definimos e, então, usadas para restringir a simulação de modelos, guiando o simulador para que gere diagramas de instância que correspondem à narrativa.

A narrativa em linguagem natural permite um entendimento intuitivo do significado dos conceitos. Comparar essas narrativas a diagramas de objeto que mostram a instanciação do modelo formal permite compreender como os conceitos são formalizados. Contrastar essas simulações com as conceituações pretendidas é a base da avaliação de modelos nessa abordagem.

1. Introduction

"You can't do much carpentry with your bare hands
and you can't do much thinking with your bare brain."
-Bo Dahlbom

1.1. Motivation

In his 1972 ACM Turing Award Lecture entitled "The Humble Programmer", E. W. Dijkstra [27] discussed the sheer complexity one has to deal with when programming large computer systems. His article argues that the increase in computer processing power leads to an increase in the expectation of the use of such power; which leads to an increase in the complexity of the programming task to meet such expectations.

According to Dijkstra, we as computer scientists should take a humble position towards such complexity, accounting for human's limited cognitive capacities and using whatever possible resources to deal with such complexity. He opened the eyes of theorists and practitioners to the fact that programming computers is an extremely complex task which should not be taken lightly. Although his lecture was explicitly addressed to the act of computer programming, we may read into his words more broadly and see that the act of codification, or the representation of ideas is generally a very difficult task. So, his plea to humility can be applied in many different levels when we regard transferring information and transforming it into physical or symbolic out-of-mind representation. In particular, we believe his plea to humility should also be applied to the task of *conceptual modeling*.

In a broad perspective, conceptual modeling has been characterized as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication" [49]. The resulting formal descriptions are called conceptual models and are built using artificial modeling languages. The quality of these formal descriptions is the main object of this work.

We are particularly interested in assessing the correspondence between the conceptual model and the subject domain it intends to represent, which we call here *accuracy* (following Guarino [42]). Accuracy is particularly important if conceptual models are to be used as a basis for the construction of an information system or for the definition of controlled vocabularies for semantic interoperability.

As argued in [49], the quality of a conceptual model depends partly on the support provided by the modeling language in which it is defined. This concern has justified the revision of a portion of the UML into the OntoUML conceptual modeling language [49]. This revision enables modelers to make finer-grained distinctions between, among other things, different types of classes according to the UFO foundational ontology [49], leading to what we call here ontology-based conceptual modeling. The objective of ontology-based conceptual modeling is to better represent a conceptualization of a subject matter.

While the quality of the conceptual modeling language employed is important, conceptual modeling itself remains subject to human error and the modeler's intention may not be properly reflected in the models. As Dijkstra proclaimed, this act of externalizing knowledge is hard and should not be taken lightly. This means that models should be subject to assessment, to ensure they may be effectively put to use. Assessing model quality is a challenging activity in itself, in particular assessing whether the model corresponds to the modeler's original intention, and whether it reflects accurately the conceptualization of a subject matter expert. Since subject matter experts often do not know the modeling language and modelers often know little or nothing beforehand about the subject matter, model assessment typically involves communication between modelers and subject matter experts. Their different backgrounds create an important communication gap that needs to be addressed during model assessment.

Conceptual model assessment can be approached from various perspectives, which motivated efforts into building tools [5, 8, 10, 45] and techniques [9, 70] for this purpose. These include automatic syntax verification, anti-pattern detection, designing cognitively efficient diagrams and model simulation for OntoUML models.

Each of these approaches contributes to model quality in a different way. Syntax verification [6,18] can show if the language's syntactic rules are obeyed and point to where they have been violated, but is not suitable to show whether the intentions of the modeler are correctly represented. Anti-pattern detection [68, 69] scans the model for configurations that are error-prone and offers a wizard-type interface to help decide if the model is correct or if it should be changed. In the latter case, it offers automatic correction of the model, including OCL [46] rules. While that helps detect errors and validate the model, the errors found are structural in nature i.e. with a focus on the use of the language constructs. Also, they do not help in improving coverage. Cognitively effective diagram design [9] helps people use the diagrams, improving their perception of

the elements involved but is neutral with regard to content. Model simulation [8, 10, 45, 46, 68] allows the observation of sequences of snapshots of model instances, revealing the dynamics of object creation, classification, association and destruction. This confronts the modeler with the implications of modeling choices and allows them to uncover mistakes or gain confidence in the quality of conceptual models. Model simulation is the approach we address and improve in this work.

In our previous approach for model simulation [8, 10, 45, 46, 68], an ontology-based conceptual model is translated to the Alloy logic-based language [54]. The resulting Alloy specification is fed into the Alloy Analyzer which then presents valid instances of the model (amounting to what could be considered a model "simulator"). While this approach has shown to be valuable in model assessment, the generation of model instances in this approach has so far been based purely on a random strategy, which is internal to the Alloy Analyzer. This means that the modeler cannot control the validation process. Even though this is useful to detect problems in the conceptual model (e.g., "edge cases" [88]), the simulation still has an overwhelmingly large number of possible instantiations. In order to control the model assessment process, we explore in this work a technique that allows the modeler to guide the simulation through storytelling. With this approach, we expect to help validation activities by providing structure to justify modeling decisions and act as a medium for communication between modelers and domain experts in validation sessions. This addresses both the communication gap between subject matter experts that do not understand the modeling language and the limitations of the current approach.

1.2. Objectives

This work has the objective of facilitating ontology-based conceptual model validation with the use of storytelling. We define a method of creating and using stories and narratives, both in natural language and using formal specifications. The stories provide structure to justify modeling decisions and act as a medium for communication between modelers and domain experts.

The following specific objectives are pursued:

- the definition of a specification language to capture stories formally;
- the extension of the current model simulation approach, in order to allow the use of story specifications to guide the simulation;

- the development of a tool for story specification and transformation; and,
- the demonstration of the technique in the assessment of OntoUML models.

1.3. Approach

In our approach we consider stories and conceptual models as complementary communication artifacts. Our approach supports model validation by using informal *Natural Language Narratives* that exemplify possible instantiations of the model and *Formal Story Specifications* that constrain the model simulation. Natural Language Narratives can be understood regardless of modeling language expertise and can be used to illustrate a model simulation in terms that are easy to understand. Formal Story Specifications bridge the informal Natural Language Narrative to the formal elements of a model, constraining the simulation to be the formal counterpart of the Natural Language Narrative. An instance diagram resulting from a simulation that fits such criteria is called here a *Formal Narrative*.

According to [26], "there is little doubt that narrative thought developed earlier in human history than scientific and logical thought". The ability to narrate gives us the possibility to reenact real-world events eliciting the imagination of the listeners, giving them experiences that they never had themselves. Early in the history of mankind, oral storytelling culture produced collective, standardized narrative versions of reality, particularly of past events; having become what we call the dominant "myths" of a society. Myths reflect the earliest form of integrative thought. In contrast with myths, theories are "very large, externally nested cultural products" which only emerged much later, as our culture allowed the externalization of memory [26] (e.g. writing).

Like storytelling, conceptual modeling is also used for transferring knowledge. Nevertheless, the concrete representation of this knowledge takes a very different form. Although a conceptual model also represents a view of some subject matter, it does so in a very structured manner, using a formal language to describe the categories of entities that are assumed to exist in a subject matter and how these entities relate to each other.

Our approach helps to validate conceptual models combining these two complementary means of communication (storytelling and conceptual modeling). We aim to leverage the value of storytelling as means for recording and transferring knowledge, not substituting but enriching ontology-based conceptual modeling. This approach builds on existing infrastructure (i.e. extends the OntoUML2Alloy model assessment approach)

by constraining the simulations using Formal Story Specifications (roughly, a simulation specification). This allows the user to have a finer control over the simulation, turning it into a tool to conduct intentional investigations in validation activities, instead of relying on random simulations.

Notice there are two facets of our approach. One is centered on the artifacts: manipulating the elements of the conceptual model, creating specifications and narratives. The other is mental: the stories act as tools for thinking, reducing the cognitive effort of mentally manipulating the symbolic elements of the conceptualization.

In **Fig. 1** we summarize our approach, showing three of its elements: (i) the Natural Language Narrative, (ii) the Formal Story Specification, which uses elements of the conceptual model and (iii) the Formal Narratives (roughly a simulated story).

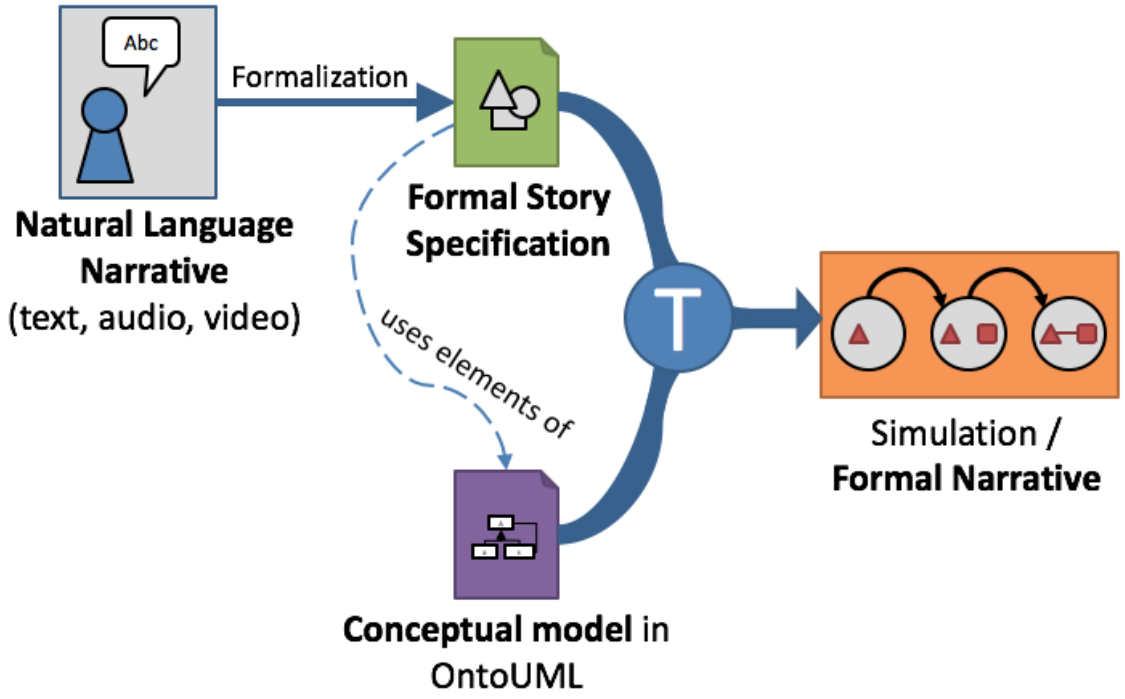

Fig. 1. An overview of the approach

In the first step, natural language narratives about the subject matter are recorded. **Natural Language Narratives** are the things that we typically refer to when using the words “narrative” or “story” in an informal context: a piece of text, a narration or a movie, for example. To create natural language narratives, we can draw from the literature about patterns for plots and narrative structure (reviewed in detail in chapter 3) and thought experiments in Science and Philosophy in general. In this approach, subject matter experts and modelers create Natural Language Narratives using the concepts that appear in the conceptual model.

In the second step of the approach, the modeler translates these Natural Language Narratives into **Formal Story Specifications** using a specification language we defined specifically for this purpose. In a Formal Story Specification, elements from the Natural Language Narratives (such as characters and relationships) are partially formalized regarding their semantic content, including the specification of which classes of the conceptual model they instantiate.

In the third step of the approach, these Formal Story Specifications (which partially define valid instantiations of the model) are translated to Alloy predicates, which are used to constrain the Alloy model generated by the OntoUML2Alloy model transformation (an Alloy specification that corresponds to the OntoUML conceptual model). Running such model results in what we call a **Formal Narrative** (a.k.a. model simulation).

By complementing a Natural Language Narrative with a Formal Narrative, one can exemplify how the domain was modeled. That means modelers may assess whether their intentions were correctly expressed in the model by exemplifying model features to validate them. Also, the presentation of a Formal Narrative along with a Natural Language Narrative allows the audience to assess the content of a model regardless of their knowledge of the modeling language: relating which elements of a Natural Language Narrative correspond to formalized knowledge. In particular, we argue that this helps reveal to subject matter experts the consequences of a theory specified in a conceptual model, lifting the burden of learning the conceptual modeling language. In other words, this helps to bridge the communication between modelers and subject matter experts.

We focus on *a posteriori* assessment of conceptual models, i.e., we assume the assessment approach is applied into existing OntoUML models. In order to demonstrate the applicability of the approach, we assess some previously published OntoUML models. In Chapters 4, 5 and 6 we use a single model to explain the approach. We show the additional application of the method to other models in Appendix C, Appendix D and Appendix E.

1.4. Technical arrangement

We assume models are defined using the ontologically well-founded OntoUML profile [47], which provides a clear semantics for a fragment of UML class diagrams, and

is supported by modeling tools such as OLED [47] and Menthor Editor[1] . Using the existing functionality in these tools, OntoUML models can be transformed into Alloy specifications, which are in turn fed to the Alloy Analyzer to generate simulations. This arrangement of tools is shown in Fig. 2.

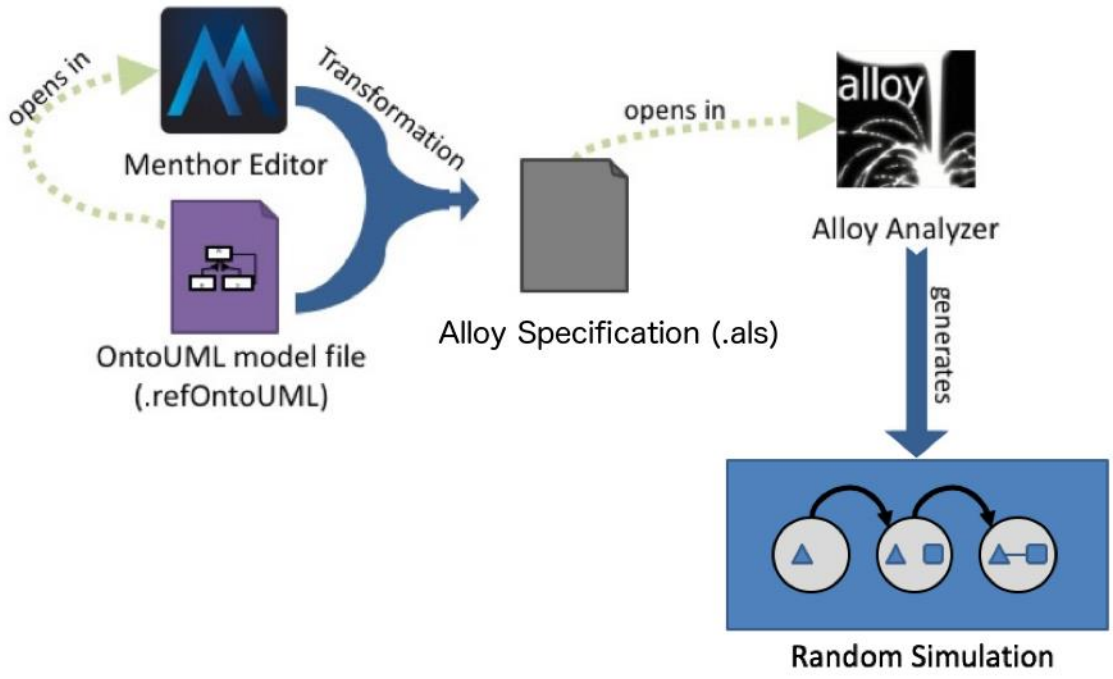

Fig. 2. Technical arrangement of the previous approach

The approach defined in this work extends this arrangement (Fig. 3), introducing the Story Modeler application that manipulates Formal Story Specifications. A meta-model for the Formal Story Specification Language is provided using the Eclipse Modeling Framework (EMF), capturing the language's abstract syntax. The meta-model references elements of the OntoUML meta-model; therefore a Formal Story Specification is defined referring to an OntoUML model, which means the application depends on a predefined OntoUML model (see Fig. 3). The Story Modeler provides an interface to create and manipulate formal stories in the Formal Story Specification Language, serializing them into a Formal Story file (Fig. 3). Once a Formal Story Specification is defined, the application may generate Alloy predicates that complement the Alloy models generated by the Menthor Editor. These predicates constrain the Alloy Analyzer to generate Formal Narratives that respect the specified Formal Story Specification.

[1] www.menthor.net

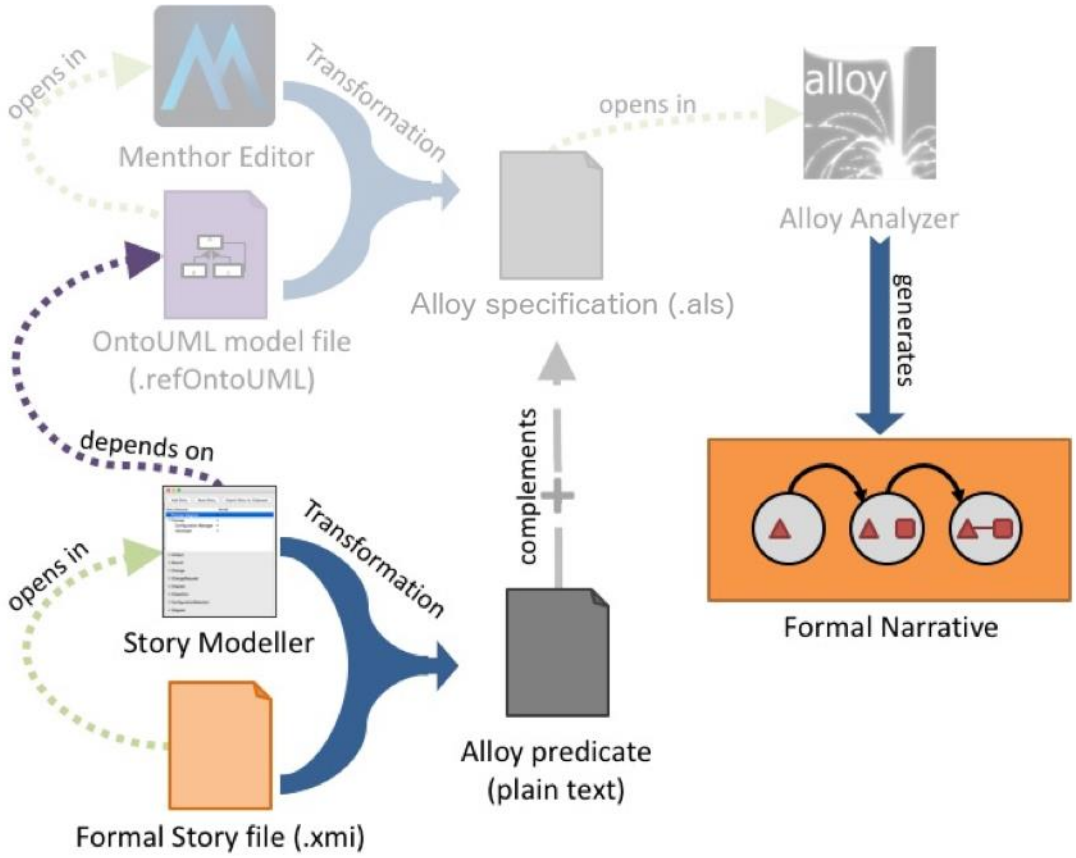

Fig. 3. Technical arrangement of the current approach

1.5. Structure

The rest of this work is organized as follows: in chapter 2, we review Conceptual Modeling, OntoUML (the conceptual model language we have adopted for this work), and the state of the art of OntoUML conceptual model assessment. In chapter 3, we review storytelling and how it can be used to transfer knowledge about reality. In chapter 4, we discuss the development of Natural Language Narratives for our approach, presenting our running example and providing a narrative for it. In chapter 5, we present our approach to creating Formal Story Specifications, building up on our running example using the Natural Language Narrative we developed in the previous section as a basis to create a Formal Story Specification. We also discuss in chapter 5 our Story Specification Language and how we may transform models in this language to Alloy. In chapter 6, we introduce an iterative assessment technique and exemplify it by simulating our running example model and modifying it. In chapter 7 we discuss some related work, and, finally, in chapter 8, we present conclusions and topics for further investigation.

Some additional content is included in annexes for further reference. Appendix A offers a minimal review of Alloy for those unfamiliar with the language, covering only the aspects of the language relevant to this work. Appendix B details how the existing model transformation from OntoUML to Alloy can be modified for better performance. Appendix C presents some existing thought experiments used to discuss conceptualizations and their codifications in OntoUML models. Appendixes C-F discuss further applications of the method on different models. Appendix C is an application of the method to a Bank model developed by a student in an undergraduate course assignment. Appendix D is an application of the method to a fragment of an Inventory Management model developed by a professional for a company. Appendix E is an application of the method to a published model for the generation of Emergency Plans, exemplifying the benefits of using Natural Language Narratives alone.

2. Conceptual Modeling background

"We are in sum, incomplete or unfinished animals who complete or finish ourselves through culture."
-Clifford Geertz (1973)

Since this work builds up on previous developments on ontology-based conceptual modeling, we review in this chapter the aspects of these developments that are important to understand the rest of this work, including key notions, the adopted modeling language and existing validation approaches. We also identify the opportunities for improvements which are the object of this work.

2.1. Introduction

A Conceptual Model is a communication artifact. It holds information about the way someone or a community understands a subject matter. In other words, Conceptual Models are symbolic representations of *domain conceptualizations* of real-world phenomena[2] (Fig. 4). A domain conceptualization is the set of concepts used to *articulate about a phenomenon*. For example, in the domain of genealogy, the domain conceptualization could include the concepts of Person, Father, Mother, Offspring, being the father of someone and being the mother of someone. An articulation about a phenomenon is called a domain abstraction, which is in accordance to (i.e. uses the concepts from) a domain conceptualization (Fig. 4) e.g. "John and Mary had a daughter called Joanna" is a domain abstraction that uses the domain conceptualization mentioned above. Domain conceptualizations and domain abstractions are entities that exist only in the mind of someone who perceives and understands reality in some way. In order to externalize this, modelers represent the *invariants* of domain abstraction using a formal language resulting in a conceptual model. The objective is to describe the categories of entities that are assumed to exist in the subject domain and how these entities relate to each other.

[2] . Naturally, a conceptual model can never be perfect. First of all, the person is unable to perceive the phenomena in its entirety, and therefore his/her abstractions are incomplete. Second, the expressivity of the conceptual model is constrained by the conceptual modeling language used to create it and by the ability of the modeler using it. Therefore, the conceptual model is an incomplete (but nevertheless useful) representation of a domain abstraction, which in turn is an incomplete understanding of a phenomena.

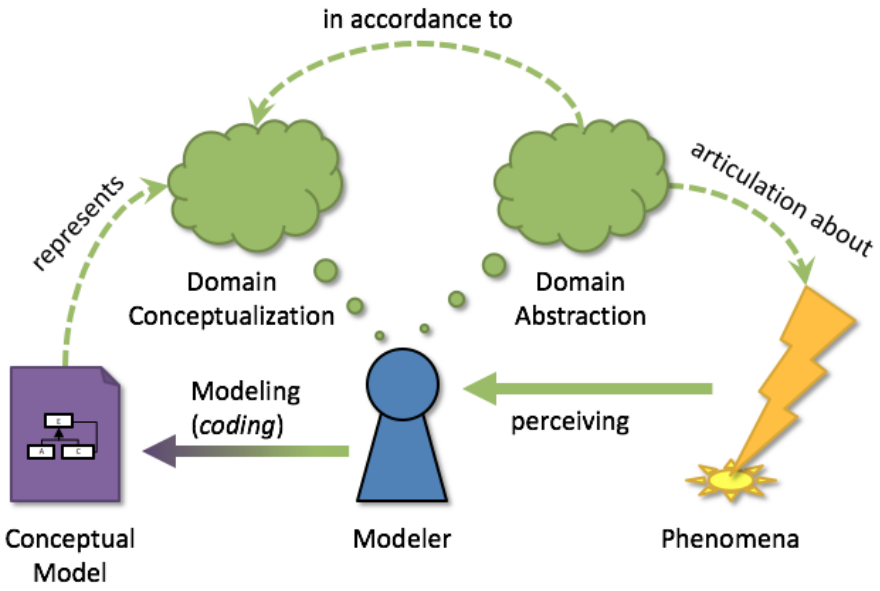

Fig. 4 A Conceptual Model represents a Domain Conceptualization

In this work we are particularly interested in the *accuracy* [42] of conceptual models. As argued by Guarino in [42], there are "two possible ways an ontology can get closer to a conceptualization: by developing a richer axiomatization, and by adopting a richer domain and/or a richer set of relevant conceptual relations". In other words, a model can have improved accuracy by improving its *precision* or its *coverage* of the domain.

One way of developing an *accurate* model is by choosing an appropriate modeling language. As argued in [49], the quality of a conceptual model depends partly on the support provided by the modeling language in which it is defined. Therefore, in section 2.2 we review a language that is ontologically well-founded, and therefore suited to faithfully representing domain abstractions. That means the *building blocks* used to *code* the domain conceptualization (Fig. 4) are closely related to the notions humans use to conceptualize about reality.

While the quality of the conceptual modeling language employed is important, conceptual modeling itself remains subject to human error and the modeler's intention may not be properly reflected in the models. Validation is "the process of determining the degree to which a model is an accurate representation of the real-world from the perspective of the intended uses of the model" [25]. We discuss model validation in section 2.3. In section 2.4 we discuss validation for OntoUML models in particular. Finally in section 2.5 we discuss challenges in conceptual model validation that can be addressed by incorporating storytelling in the existing OntoUML model validation approach.

2.2. OntoUML

In this section we review OntoUML, an ontologically well founded language to create conceptual models that are suited to represent domain conceptualizations accurately. The section is elaborated to overview a fragment of OntoUML language to those unfamiliar with the language. Detailed definitions of OntoUML can be found elsewhere, such as in [5], [6], [49] and [86] (in Portuguese).

We are motivated to use this language because of the arguments put forward in [49] that the language *is ontologically well-founded* and therefore adequate to truthfully representing real-world phenomena. The OntoUML conceptual modeling language is an extension of UML with specific concerns for the semantics behind the modeling constructs. UML is the *de facto* standard for conceptual modeling activities and such widespread use has revealed that the language lacks a precise definition of its formal semantics [48]. In [49] Guizzardi argues that in order to model conceptualizations of reality, a language should be based on *foundational ontologies* [48]. Foundational ontologies describe very general concepts like "space, time, matter, object, event, action, etc., which are independent of a particular problem or domain" [42]. In particular, OntoUML it is founded on the Unified Foundational Ontology (UFO) [49]. The stereotypes offered in OntoUML allow the construction of ontologically well-founded conceptual models, meaning they express precise semantics corresponding to the categories in the foundational ontology (UFO). Fig. 5. shows (in gray) some ontological distinctions of UFO that are realized as stereotypes for classes in OntoUML. Stereotyped classes are the building blocks of an OntoUML model.

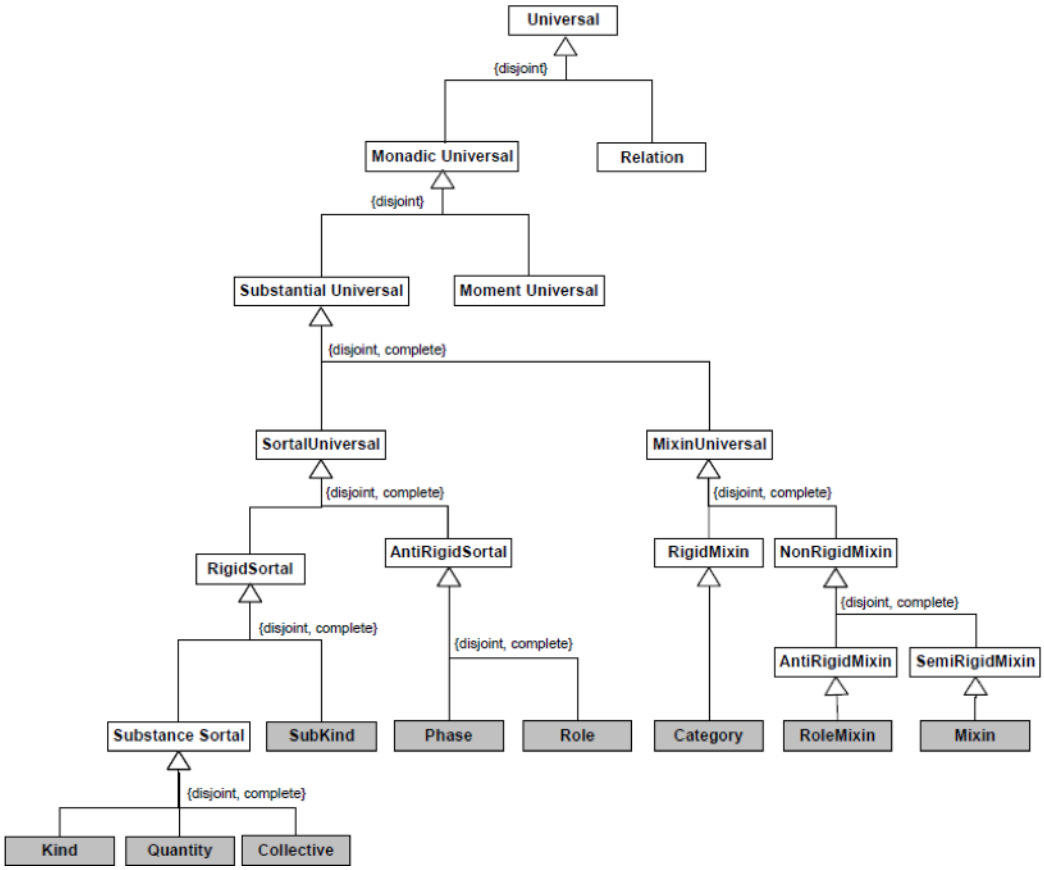

Fig. 5. A fragment of UFO showing ontological distinctions among *Substantial Universals*

Fig. 6 exemplifies a model specification using OntoUML. This model will be used later on this chapter to discuss the distinctions underlying the language. The model defines Enrollments between Students and Schools, as well as some details about the Organizations that can play the role of School and the Persons who can play the role of Students. Both Persons and Organizations are Agents, but while any Organization is Insurable, only Living Persons are Insurable. Naturally, any Person which is not Living is Deceased. Finally, any Person is either a Child, Teenager or Adult.

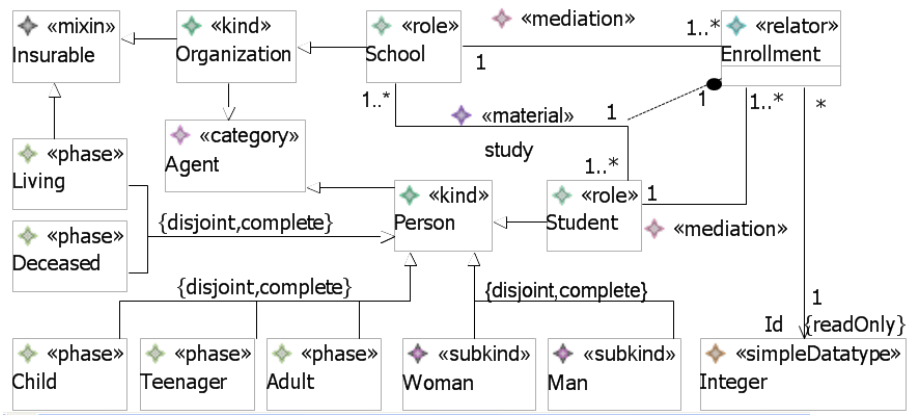

Fig. 6. Example OntoUML model

2.2.1. Individuals and Universals

In UFO, there is a distinction between Universals and Individuals (roughly, types and instances). Universals are *coded* in a conceptual model, while the Individuals that are classified by them are not. While the conceptual model represents a domain conceptualization, the possible configurations of individuals (instances of such conceptual model) are called *World States*. World States are, therefore, the symbolic counterpart of domain abstractions.

Individuals are entities that exist in reality possessing a unique identity, and Universals, conversely, are space-time independent pattern of features, which can be realized in a number of different Individuals [50]. For example, a particular person (such Barack Obama or Dilma Rousseff), would be an Individual, while the concept of Person would be a Universal.

World States consist of *individuals (instances of monadic universals)* and the relations between them; each individual classified by some (one or more) *Universals.*

For example, consider the World State depicted in Fig. 7 (an informal representation of a possible instantiation of the model presented in Fig. 6) where a person, Bernardo, studies at Móbile, a school. Bernardo and Móbile are *individuals* and instantiate *Monadic Universals* such as Person, Man, Student, Organization and School. Also, a *relator* holds between them and such *relator* is also classified by (instantiates) *Universals*, such as Enrollment.

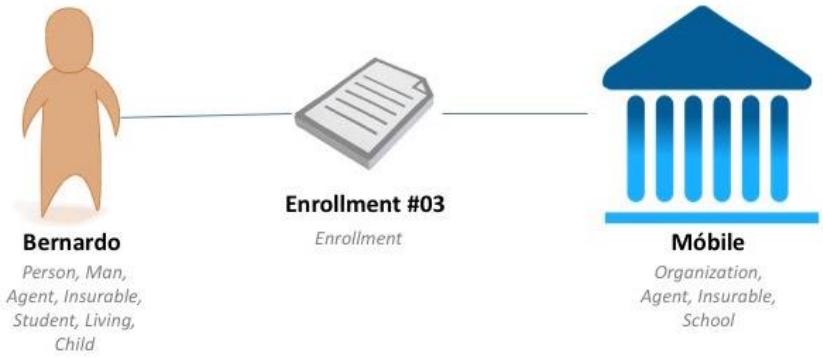

Fig. 7. An example instance diagram: Bernardo studies at Móbile

2.2.2. Dependence

Dependence is an extraordinarily common and varied phenomenon and OntoUML language offers constructs that reflect distinctions in types of dependence. For example, an enrollment depends on the student and on the school in a way which is not symmet-

rical: an enrollment can only exist as long as the participants exist, but the same cannot be said conversely. Moreover, for a person to be a student, he must be enrolled in (and therefore depends on) some school and not on a specific school, while an enrollment depends on specific individuals to exist: if you change the person or the school, it cannot be said to be the same enrollment (the identity of the enrollment is connected to the participants). This exposes a fundamental distinction between two types of dependency relation: generic dependence and specific dependence. The student depends *generically on some* school to be a student, a generic dependence; Enrollment#03 depends *specifically on* Bernardo and Móbile, constituting a specific dependence.

A particular yet very important type of specific dependence is existential dependency. Existential dependency means an individual depends necessarily (in a modal sense) on some specific individual to exist, e.g., Bernardo depends existentially on his brain; there is no situation where he can maintain his identity without having his specific brain as a part of him. Existentially-dependent relations include *characterization*, *mediation* and *essential* or *inseparable* parthood. [49] We refrain from detailing these types of relations for the sake of conciseness.

2.2.3. Moments and Moment Universals

A special kind of existentially-dependent individual is called a *moment*, derived from the German word *Momente* from the writings of Husserl and denotes what is called also called a “trope”, “abstract individual” or “property instance”. *Moments* can be understood as objectified properties of an individual and are *inherent* to them [86]. For example, the color of an apple is a *moment* inhering on the apple; John’s headache is inherent to John, as the intensity of the headache is inherent to the headache.

Differently from moments, individuals that do not inhere in other individuals are called *substantial individuals*. *Universals* that describe *substantial individuals* are called *Substantial Universals*; likewise, universals that describe *moments* are called *Moment Universals*; both are specializations of *Monadic Universal*.

2.2.4. Rigidity

The rigidity of a Universal determines how the universal may be applied to instances. A rigid universal applies necessarily to its instances, while an anti-rigid universal applies contingently. For example, consider the model in Fig. 6 where Person is a rigid

class and Student is an anti-rigid class; according to this model, any individual Person cannot cease to be a Person but may become and cease to be a Student.

The set of rigid universals in any conceptual model defines its backbone taxonomy, the most important features of an ontology [43]. The term refers to the structural role these concepts perform in conceptual modeling: considering only the elements of the backbone gives someone a survey of the entire universe of possible instances [43]. These are divided in three types: Categories which do not carry identity, Kinds which supply identity and SubKinds which carry but do not supply identity. Universals that carry identity are called Sortals, while those that do not are called Mixins. Mixins classify individuals that obey different principles of identity (e.g., Agent in Fig. 6 which classifies different kinds of entities such as Persons and Organizations). Hence, Mixins are types which provide properties to (characterize) individuals which have already been individuated by sortal-supplied principles.

Anti-rigid universals describe the characteristics that apply contingently to individuals. In OntoUML there are two types of anti-rigid universals: Roles and Phases which are differentiated with regard to their specialization conditions. For the case of Phases, the specialization condition is always an intrinsic one. For instance, in Fig. 6, a Child is a Person within a certain age. For Roles, in contrast, their specialization condition is a relational one: a Student is a Person who is enrolled in (has a study relation to) a School. If the relation no longer exists, the person ceases to be a student; i.e. the class instantiation depends on the relation to another individual. Formally speaking, this distinction is based on a meta-property named Relational Dependence.

Additionally, as discussed in [49], Phases (in contrast to Roles) are always defined in a partition set. For instance, in Fig. 6, the universals Child, Teenager and Adult define a phase partition for the Kind Person. As consequence, we have that in on each world *w*, every Person is either a Child, a Teenager or an Adult in *w* and never more than one of these.

2.3. Model Validation

Model assessment is an important part of conceptual modeling in which one inspects the model for correctness and adequacy to its purposes. Model development is a human centered activity; more specifically, it can be seen as a communication activity. Human activities are naturally error-prone but communication activities are even worse; the

difficulty of proper communication appears even in ancient texts, such as in the etiological myth "The tower of Babel" in Genesis 11.

To illustrate the need for model assessment, imagine the activity of writing text in natural language. Be the text long or short, the writer usually feels the need to read what was written, to see if he made any spelling mistakes. Additionally, revision allows him to assess the meaning of the words he wrote so he may evaluate if the message conveys his intentions, i.e., if it says what he meant. Models convey messages which must be assessed for correctness as well. Two key aspects are analyzed: syntax and semantics. We call the process of assessing the syntax *"model verification"*, and of assessing the semantics *"model validation"*[3]. While verification is important, this work is mainly concerned with validation[4]. Model validation involves understanding the possible configurations of elements implied by a model to attest their fidelity in representing the domain abstractions (Fig. 8).

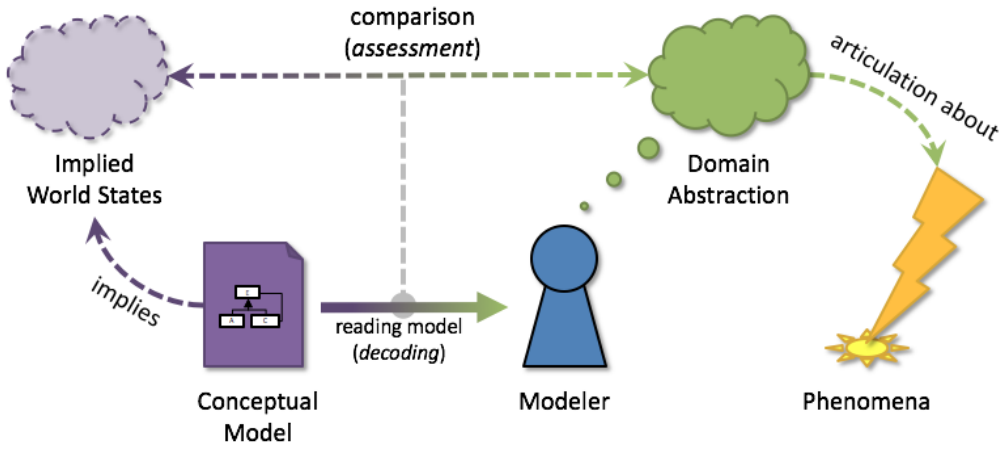

Fig. 8 Model assessment involves comparing the worlds states implied by a model to the domain abstractions in the mind of the person assessing them.

Naturally, to decipher a model and understand the World States implied by it, one must understand the modeling language (Fig. 9). This activity depends on the skill of the modeler in the modeling language. Skilled modelers are able to fully decode what is logically implied by the model, while unskilled modelers are not.

[3] Verification and validation are words with similar meaning; their usage to convey these specific meanings, as we do, is not completely agreed upon.

[4] In section 7.1 we include some discussion on OntoUML model verification.

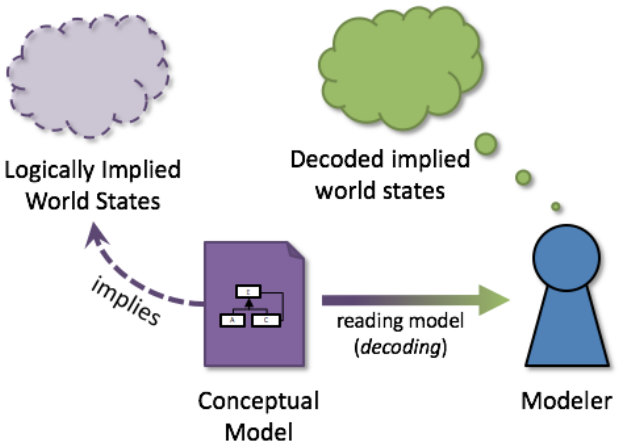

Fig. 9 Reading (decoding) a model allows one to understand what is implied by the model

Additionally, to attest that what is implied by the model is valid requires knowledge of the domain, i.e. requires having previous domain abstractions to have something to compare to (Fig. 10).

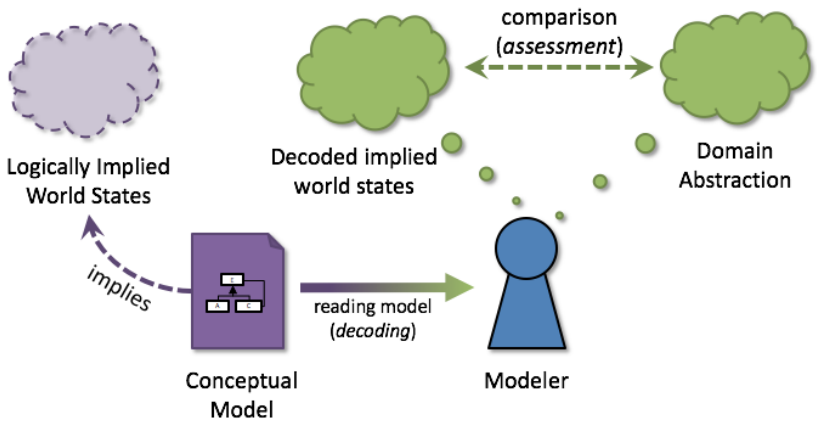

Fig. 10 A modeler assesses if the World States implied by the model are valid according to his domain abstractions

In the next section we describe an approach to validate conceptual models defined in the OntoUML language, which we will extend in this work. The approach facilitates model validation by simulating the model. The simulation is manifested in diagrams for some World States that exemplify class instantiation and can therefore help the modeler in understanding what is logically implied by the model.

2.4. OntoUML Conceptual Model validation: Previous approach and opportunities for improvement

This work builds on OntoUML2Alloy which was specified initially in [8,10], and later merged and improved, following considerations raised in [68]. OntoUML2Alloy is

a software that uses Model-Driven Development (MDD) techniques to automatically transform models in the OntoUML language to the logic-based language Alloy[5] [54]. The product of this transformation is an Alloy specification that can be fed into the Alloy Analyzer to generate a sequence of instance-level state-of-affairs which are valid according to the language axioms (Fig. 11). The analyzer may be further used to produce assertion counter-examples, i.e. to query possibilities within the model's constraints. OntoUML2Alloy supports validation by allowing the observation of sequences of snapshots of model instances. The visualization of instances confronts the modeler with the implications of modeling choices [10]. Should the instances reveal inadmissible states-of-affairs (or sequences thereof), the model may be analyzed to identify opportunities for correction in an iterative validation approach. Moreover, this can also be used as means to identify missing or over restrictive domain rules [10].

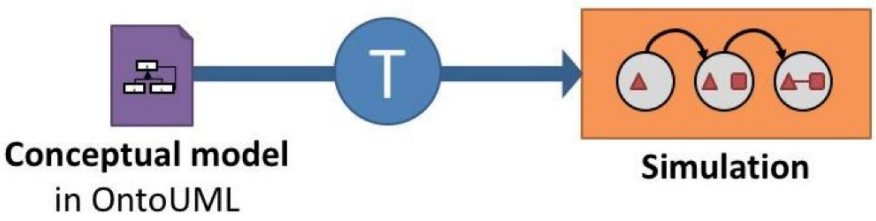

Fig. 11 OntoUML2Alloy: a model transformation (T) allows the simulation of the conceptual model

While OntoUML2Alloy is useful for conceptual model validation, understanding the diagrams generated automatically by the Alloy Analyzer can be difficult, which justified previous work in providing methodological guidance to designing cognitively effective diagrams [9]. Cognitive efficiency in [9] refers to taking advantage of the properties of human graphical information processing i.e. how human visual systems acquires graphical information and how that information is processed in our minds. To improve perception, *visual variables* of the diagram such as color, position, shape and texture should be used to increase visual discriminability of the information coded in the diagram.

The work developed in [9] sheds light on how to deal with the visual complexity of the diagrams. However, there is another problem regarding complexity which was not addressed; namely, the complexity of the content generated by random simulations. On one hand, having a complete random generation of instances for the simulation is positive, as it can find unexpected scenarios and therefore help discover modeling mistakes.

[5] For a revision of the Alloy language and its analyzer, please refer to Appendix A.

On the other hand, the random generation of instances often makes the simulation overwhelming. It can often include trivial and repetitive scenarios, making relevant scenarios hard to find. Additionally, the relevant scenarios can be polluted with irrelevant information. For example, consider Fig. 12 which depicts a simulation of the model in Fig. 6. In the first World Bernardo is a Man and a Child and Móbile is an Organization. In the second World, there is Bernardo, Mary, John and Fred, and Bernardo enrolls in Móbile, becoming a student. In the third World, there is only Bernardo, no longer a student. In that model simulation, Bernardo was used to exemplify the anti-rigidity of the student class and how an enrollment relationship can be established and later destroyed. To this effect, John, Mary and Fred acted as noise, drawing attention where attention was not needed.

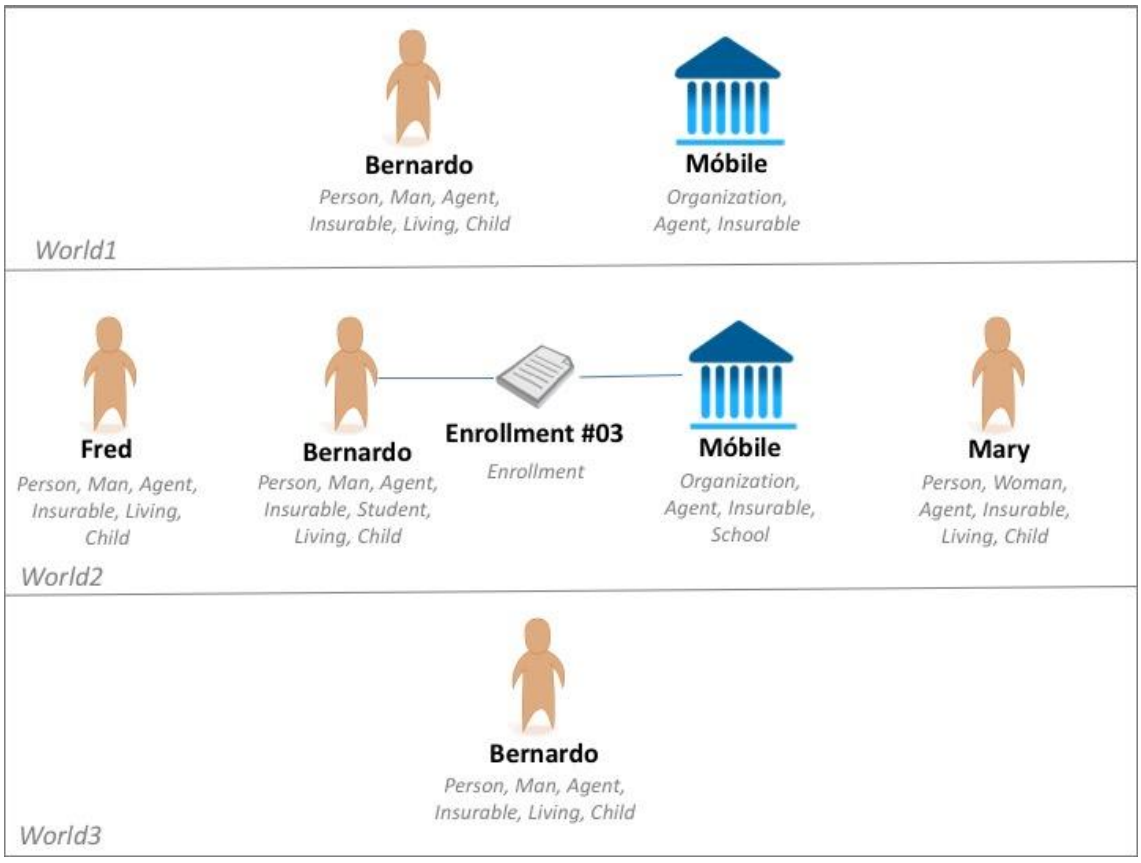

Fig. 12. Simulation that has irrelevant elements (Fred and Mary) in the second world

This work addresses such complexity and reduces the noise of the simulations by providing tool support for controlling the simulation generation. Using our approach one can specify the World States intended to be tested instead of manually generating and inspecting several random simulations to find the desired state of affairs. In our previous example, we could specify a story to find a sequence of snapshots that include a Person and an Organization in all of the snapshots, an enrollment relationship between

them in the second snapshot (but not on the first and last snapshot). The benefits of random simulation can still be leveraged, if desired. In chapter 5, the approach to constrain the simulation is explained in detail.

Controlling the simulation helps the modeler check if some World States he decoded as possible by reading the model are in fact possible (i.e. logically implied by the model) (Fig. 10). However, that activity only helps to confirm that the symbolic representation behaves as expected. The link between the symbolic representation and the Domain Abstractions still requires understanding of the modeling language. This is particularly relevant when discussing the quality of models with subject matter experts which typically have no knowledge of the modeling language.

Storytelling (discussed in depth in chapter 3) can help connect the formal specifications to situations in reality. Storytelling can also help structure the content of the simulation, creating nexus between the elements. This helps to avoid including elements that act as noise, as discussed previously.

2.5. Challenges for Conceptual Model validation

While the previous work provides means to assess conceptual models and reveal what is implied by the model, there are some challenges in model validation that have not been addressed in previous work. Model validation can be particularly challenging because, in realistic scenarios, the Subject Matter Expert (whose domain abstractions can be considered authoritative) and the Modeler (an expert in the modeling language who is able to understand what is implied by the model) are often different people. This is depicted in Fig. 13, which introduces some communication between subject-matter expert and modeler.

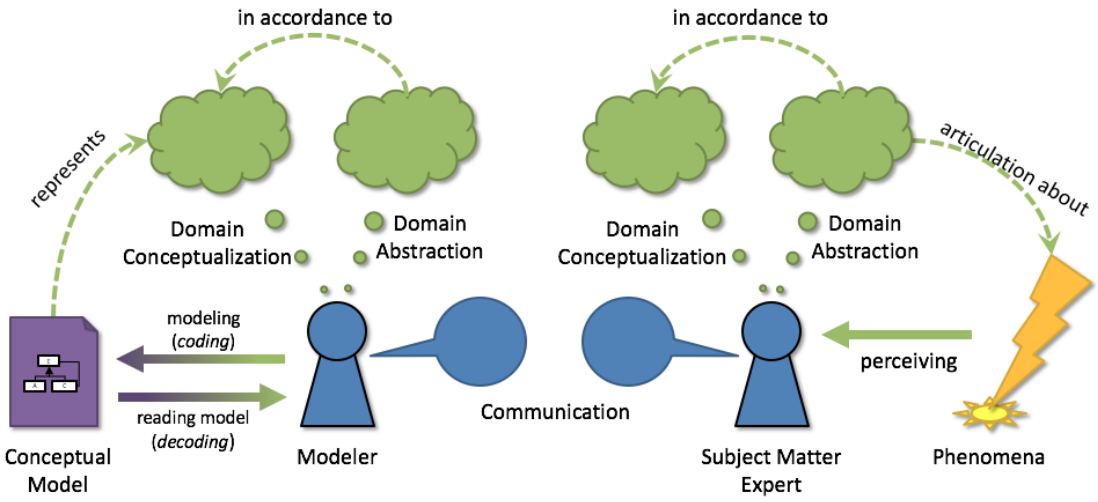

Fig. 13 Cooperation between Modeler and Domain Expert to validate a conceptual model

We assume the modeler cannot learn enough about the Phenomena to become a Subject Matter Expert himself and that the Subject Matter Expert cannot learn enough about the modeling language to become a Modeler himself. Therefore, the modeler has a limited access to Domain Abstractions and Domain Conceptualizations, while the Subject Matter Experts has a limited knowledge of what is implied by the model. In such situation, subject-matter expert and modeler must communicate to cooperate and assess the model together. Such cooperation can take the following formats:

(i) Subject-matter Expert communicates Domain Abstractions and Domain Conceptualizations to the Modeler: providing the modeler with documentation about the subject matter, showing the phenomena to the modeler, participating in interviews and answering questions.

(ii) Modeler decodes the model for the Domain Expert: guiding the Domain Expert through the models explaining what they mean or showing possible World States by instantiating the model via model simulation.

In this work, we will focus on (ii), using storytelling to help modelers communicate some admissible worlds implied by the model to subject matter experts. The previous approach is extended allowing modelers to create simulations derived from Natural Language Narratives. These narratives can be understood by the Subject Matter Expert regardless of their understanding of the modeling language. By comparing them with the Formal Narratives (i.e. diagrams generated by the guided simulation), they may better assess what is implied by the model. This effectively helps with decoding the model to the Subject Matter Expert and revealing what is possible to articulate using the Con-

ceptualization captured by the model. With the feedback of this activity, we expect (i) to be facilitated.

2.6. Concluding remarks

In this chapter we have discussed conceptual modeling, the modeling language we adopt in this work, i.e. OntoUML, and the validation of models in this language. We have shown that the current state of the art in OntoUML model validation has limitations in respect to the control modelers have over the simulation and motivated the contributions we offer in this work create mechanisms of controlling them. This helps reduce the noise in simulation, i.e. irrelevant content that hinders its understanding.

We argue that storytelling can help create nexus in the content of simulations and help communicate the World States that are logically implied by a conceptual model to Subject Matter Experts which do not understand the modeling language.

In the next chapters we will review some theory about storytelling that justifies this argument and introduce elements to the current simulation approach that allows a modeler to control it and present the results to Subject Matter Experts.

3. Storytelling

"We speak not only to tell other people what we think,
but to tell ourselves what we think.
Speech is a part of thought."
- Oliver Sacks – Seeing Voices

Our approach uses storytelling to improve model validation activities. This chapter examines the ways storytelling impacts human communication and culture, serving as a foundation for the chapters that follow. The chapter constitutes a interdisciplinary examination of the importance of stories in human communication, which motivates the adoption of stories as complementary to conceptual modeling and to conceptual modeling validation in particular.

3.1. Introduction

Stories, myths and religious text have always been used as means of transferring knowledge within society and its subsequent generations [15]. The word narrative itself stems from the Latin word *gnarus* which means 'knowing' [63]. According to [26], "there is little doubt that narrative thought developed earlier in human history than scientific and logical thought". The ability to narrate gives us the possibility to reenact real-world events eliciting the imagination of the listeners, giving them experiences that they never had themselves. Early in the history of mankind, oral storytelling culture produced collective, standardized narrative versions of reality, particularly of past events; having become what we call the dominant "myths" of a society. Myths reflect the earliest form of integrative thought. In contrast with myths, theories are "very large, externally nested cultural products" which only emerged much later, after our culture allowed the externalization of memory [26] (e.g. writing). We take ontology-based conceptual models to be a particular means to represent a theory about a subject domain, formally capturing admissible states of affairs [47] using invariants i.e. logical assertions or rules that are held to always be true.

This work defends the use of narratives to facilitate validation activities in conceptual modeling. In this context, the narrative reveals domain abstractions about a subject matter by exemplifying it in a real context (revealing therefore the real world semantics of elements of the domain conceptualization). Such exemplification reveals the theory

in a strong and intuitive way, providing insight into causality and dependence of the domain and serving as an anchor to the concepts. By using the narratives as context for the model validation, we may assess the model more accurately.

The rest of this Chapter is structured as follows: in section 3.2, we use a story to motivate the importance of storytelling in human culture, exemplifying how it can be used to transfer information about reality and plan ahead. In section 3.3, we discuss different theories that are used to analyze Stories, Narratives, Myths, their structure and importance and how they relate to human psyche and human culture. In section 3.4, we discuss how storytelling has been used in Science and Philosophy as an artifact to discuss conceptualizations and reduce cognitive effort. We argue that these may be used in a similar fashion to explain domain theories defined by conceptual models. Finally, in section 3.5, we offer concluding remarks.

3.2. An illustration of the role of storytelling in knowledge transfer

In this work we defend that storytelling can help explain theories; so in this section we use a story to intuitively show how storytelling works: we tell a story about someone telling a story, to show the importance of telling stories. In a way, this makes the section somehow recursive in nature. This section should illustrate use of storytelling as a tool to transfer knowledge, communicate complex states of affairs and motivate rules.

"A long, long time ago, in pre historic times…

Kaj arrives in his cave and finds his family. In a crude language, the family asks "where is Mak, your brother, that left with you to hunt? Hasn't he returned?" - The arriving man then says, with a sad voice - "No. Mak is no more". –the family gathered around in distress to hear Kaj - "We were crossing the mountain pass when the strongest of cats pounced on my brother. The cat dragged him and then it ate him on the mountain's edge".

Disaster! The family was desperate. Not only was the grief of losing a family member terrible, being one hunter short in the family was bad. Having the mountain pass blocked by such a dangerous animal

would make things worse, as the path lead to a small forest with many fruit trees, which the children would often go picking

Wala, mother of three said "The strongest of cats! A demon cat! Children! You will not go through the mountain pass anymore! It is dangerous! The cat may pounce on you, and you will be no more, as Mak is no more."

"We should attack him with stones" yelled a brave young boy

"No"- commanded Wala - "It got Mak, it will get you too. You will not get close to the mountain pass. You will stay in the cave".

What could they do? Not having access to the fruit trees and being a hunter short, they could face hunger. They needed a plan. (...)"

–The story of the strongest of cats

The story above shows some aspects of the use of storytelling in communicating. In the story, Kaj tells the story about his brother dying in the mountain pass. His family understands from this message a very complex scenario: the mountain pass is a dangerous place to go, others could die as Mak died. As the family listens to the story, they put themselves in the position of the characters and feel as they imagine the characters felt. This ability to empathize with the characters in a story is crucial to the mechanism of transferring knowledge with storytelling.

The family also used short narratives to plan ahead: throwing stones at the big cat is one plan (and in a way, a plan can be seen as a story too); that can be twisted and turned and re-structured. This illustrates another storytelling feature: stories can be adapted and changed. The child's reaction could have the following rationale: "Well, I would behave differently, given the same situation. Since I know it is there, I can attack it first, and kill it, like it did to Mak". In this adaptation, the child switched roles, where he was the killer, and the cat was killed instead of Mak. When he said "We should attack him with stones", in the mind of the rest of the family they could see a possible future, a possible story unfolding. In such story, the family attacks the cat.

In her mind, the mother again modified this new story imagined by the boy. In her mind, the story unfolded as "A cocky boy thinks he is very strong and attacks a demon cat with stones. The cat gets angry because he was stoned and attacks the boy, killing

him.". To avoid this dreadful ending, she stops the chain of action that would lead to it. By forbidding the young man to leave the cave, she arranges a different story (plan) in the mind of the family. "The boy stays in the cave and avoids the mountain pass, fearing he could be eaten by the demon cat."

Another way to plan ahead using a modified version of the story could be to offer food to the cat, as a means to satisfy his hunger, leading to a safe passage. If this strategy proves successful and becomes a tradition, as generations pass, they may end up forgetting why they always offer meat to the demon cat (or Cat God) the day before the fruit harvest to ensure a safe journey through the mountain pass and simply regard it as a rule that must be obeyed.

Stories were always used this way as means to communicate knowledge about the world to other people. In a simple way – easily – a broad and general concept is explained, without necessarily spelling it out explicitly. And frequently, the knowledge is justified, the reason why things are the way they are is better understood. If Kaj had simply commanded that "no one shall go through the mountain pass anymore", it probably would not be as effective as telling the story of his brother, who passed away crossing it. The story motivates the rule and clarifies it, i.e., explains why it is necessary.

In this section, we did a very informal motivation of how stories may be useful to humans. In the remainder of the chapter, we will discuss what the literature says about storytelling and narratives, how they were employed in human history and why.

3.3. The study of Storytelling: psychological aspects and story patterns

In traditions of oral storytelling prior to writing, narratives had no concrete representation and were modified in each enactment, with details told differently based on the reaction of the audience or the teller's memory/creativity. [59 apud 61, pg. 188] Regardless of the differences between narrations, the sum of related events and situations in some narratives can be recognized as being the same *story* [72]. In this work we use this terminology to differentiate between a single *story* and its many possible *narratives*. A story is a very abstract representation that has the property of identifying different (but similar) narratives as being, somehow, the same story. A single story e.g. Romeo and Juliet, may bring forth different narratives, such as a classical enactment of Shake-

speare's text in a theater or a movie adaptation of the text. In the case of the movie featuring Leonardo Di Caprio, the narrative takes place in modern times, with cars and guns, and while it is wildly different from the original text, it is still Romeo and Juliet. A narrative, roughly speaking, realizes a story.

The first type of theory we discuss is about the general structure that stories may have. Lots of work has been done in analyzing story patterns and identifying reoccurring structure. Story patterns that organize a plot can, in some cases, be called dramatic structure. The study of dramatic structure has begun with Aristotle in his Poetics (c. 335 BCE), where he proposes "to inquire into the structure of the plot as requisite to a good poem; into the number and nature of the parts of which a poem is composed; and similarly into whatever else falls within the same inquiry.". Aristotle described the dramatic structure (of a tragedy, in particular) to have 3 parts: "a beginning, a middle, and an end" [3] a beginning is where the plot is set up; the end is where the plot is resolved and the middle connects these two parts[6].

Since Aristotle, many variations emerged regarding how many parts there should be and their roles. Here, we will use Freytag's pyramid [32] as an example (Fig. 14). It starts on the left with the exposition, which introduces the characters and the setting, providing a description and background. Then, something happens and triggers the "Rising action", where the story builds and gets more exciting. The climax is the point of greatest tension in the story, often the most exciting event. It is the event that the rising action builds up to. The falling action is composed of the events that happen after the climax and that indicate that the story is coming to an end, up to a point where the main problem or conflict is solved. Thereafter, we have the Denouement, where any secrets or questions that remain are solved.

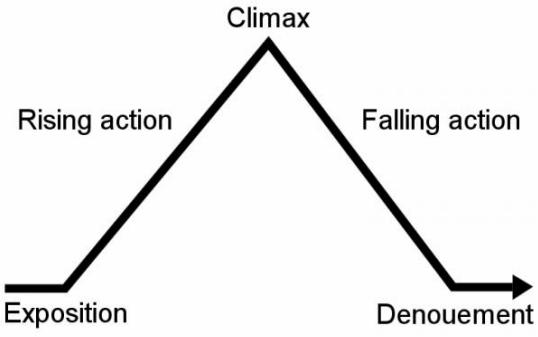

Fig. 14. Freytag's pyramid

[6] . Aristotle also discusses some narrative patterns such as the "recognition" or "reversal of the situation" and details parts of a plot, but we have chosen not to go further in detail on his analysis.

Narrative analysis is ancient. While its theoretical lineage can be traced back to Aristotle, a well established discipline only emerged much later, as part of a movement called “the Russian formalism”, where scholars found patterns that emerged from folklore. Propp's “Morphology of the Folktale” [67] assembled a set of story patterns, which he called narrative functions, that allegedly accounted for all Russian folklore i.e., each story could be composed of the set of patterns Propp introduced. Naturally, these patterns were of a very abstract nature, such as "*Departure*: Hero leaves home" or "*Receipt of a magical agent*: Hero acquires use of a magical agent (directly transferred, located, purchased, prepared, spontaneously appears, eaten/drunk, help offered by other characters)" [67] Stories could thus be broken down into their constituting elements, each with a *narrative function* i.e. their role in conveying a message.

Propp's work was criticized for leaving aside the details of the stories such as tone, mood or anything that makes a tale different from another. He was concerned about the similarities but not about the differences. Nevertheless, his work was seminal both to followers of his practice and to opposers that offered different approaches.

This field of study started by Propp is often called Narratology (but that is not a consensus). "Narratology examines the ways that narrative structures our perception of both cultural artifacts and the world around us. The study of narrative is particularly important since our ordering of time and space in narrative forms constitutes one of the primary ways we construct meaning in general" [29].

After the study of story patterns emerged, some scholars have investigated the psychological reasons why they seem to appear in any society and their role as a tool to transfer knowledge [15,75]. Like Propp, Joseph Campbell also extracted patterns from myths and religions from many different parts of the world [15]. Based on these patterns, Campbell coined “The monomyth” (also known as The Hero's Journey), which is a story that could be applied (fully or in parts) to uncountable stories. It is similar to Propp's morphology in the sense that it offers a structure that fits most stories in folklore, myths and religions.

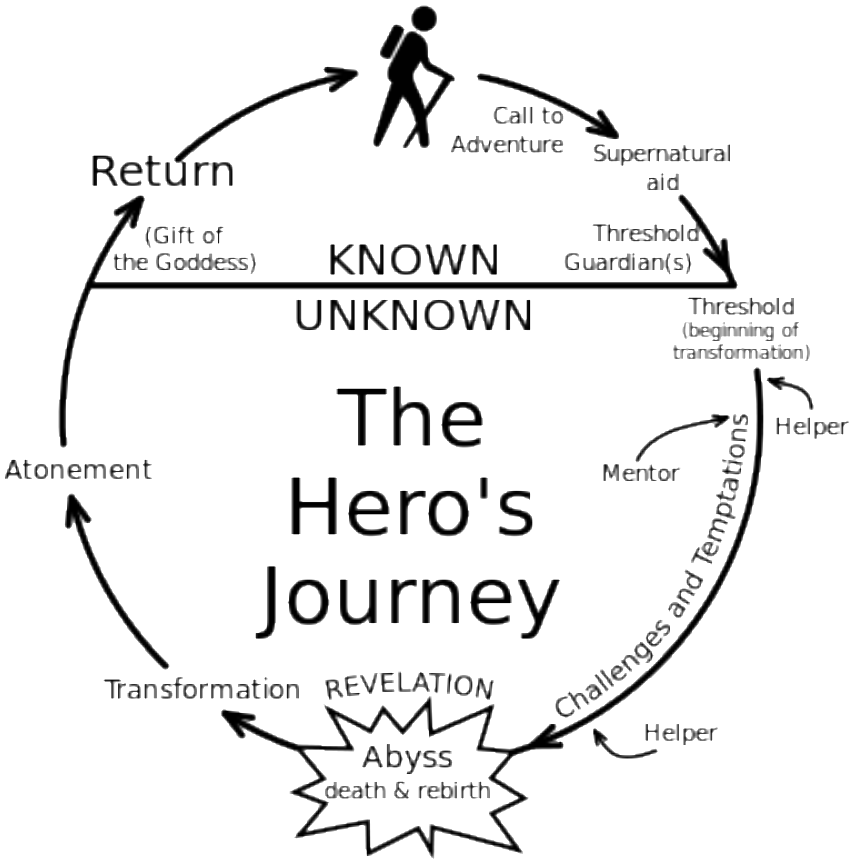

Fig. 15. The hero's journey [81]

Nevertheless, his approach was much more concerned with the psychological and anthropological reasons why these patterns seem to be cultural independent, that is, they appear in every society, regardless of cultural interchange. In Campbell's psychological analysis, human beings have a natural way of conducting themselves in societies and relating to each other. The myths serve to pass on knowledge about this social reality through generations.

Campbell built on ideas from Adolf Bastian "who recognized, in the course of his extensive travels, the uniformity of what he termed the "elementary ideas" (*Elementargedanke*) of mankind. Remarking also, however, that in the various provinces of human culture these ideas are differently articulated and elaborated, he coined the term "ethnic ideas" (*Völkergedanke*) for the actual, local manifestations of the universal forms. Nowhere, he noted, are the "elementary ideas" to be found in a pure state, abstracted from the locally conditioned "ethnic ideas" through which they are substantialized"[16] Campbell argues that the narrative patterns that occur in human culture (even in isolated societies) emerge from such "elementary ideas". He compares these ideas to trigger responses animals have, such as "Chicks with their eggshells still adhering to

their tails dart for cover when a hawk flies overhead, but not when the bird is a gull or duck, heron or pigeon." [16] It is as if human beings are hardwired to share their experiences in narrative form and as if something of the content of stories is hardwired too.

While Campbell and others have used psychoanalysis to analyze myths (socially constructed stories, and in this sense, with no individual author), psychoanalysis is traditionally used to analyze individual narratives. In psychoanalysis (starting with Freud) the patient usually narrates events that marked their life or dreams he/she had and is guided into abstracting patterns from these narratives; to extract the reoccurring logical structures of the way they relate to others.

The similarities between a particular narrative and an abstract story can go either way: one may find patterns on their own particular narratives to find they fit into a more abstract, general reoccurring story, as it is done in psychoanalysis; or take the other way around and recognize in an existing story something that matches to their own particular narratives.

When an author intentionally uses these story patterns to structure a new story, they introduce elements that the audience is prone to recognize as theirs, as part of their psyche, elements that resonate in them. The stories are transferred exactly because they touch the listeners in their intimacy, when they recognize themselves in the story told.

Because this works so well and creates such a powerful emotional experience, these narrative theories which were initially designed to analyze existing text (and better understand human culture), have also been used to design new stories. For example, George Lucas used Campbell's "Hero's journey" to structure Luke Skywalker's journey in "Star Wars" [17] as many other writers do. Also, recently, there is a trend in using storytelling in marketing and advertising.[58]

Following this trend, we can also use these theories and structures to guide our designs of stories for conceptual model validation purposes. By applying these elements in the stories, we may expect to cause empathy on the audience, that they will connect to the stories and recognize themselves in them. That is one of the reasons why we argue that using Natural Language Narratives and storytelling help conceptual model validation.

The knowledge transferred by myths, fables and novels usually explores human existence in a general way, not focusing on specific subject domains. Since we aim to use these stories as artifacts to help discuss narrow subjects in a specific activity (validation), these stories should not only be compelling, but also explore the conceptual issues

in the subject domain. In the next section we explore how Computer Science and Philosophy have intentionally created and used narratives to this effect.

3.4. Stories as *tools for thinking*

In this section we discuss briefly how stories and other similar devices (such as thought experiments) may be used as *tools for thinking*. "These handy prosthetic imagination-extenders and focus-holders permit us to think reliably and even gracefully about really hard questions." [23]

Used in Philosophy and Science, thought experiments are "devices of the imagination used to investigate the nature of things" [13] . Examples include Maxwell's demon, Einstein's elevator or Schrödinger's cat. [13] They are usually small stories embedded in larger arguments and serve as means to analyze phenomena in detail [23].

A taxonomy for thought experiments may divide them between "constructive vs. destructive" [12] As the names suggest, *constructive* thought experiments help the elaboration and defense of a theory or argument. They reveal the key features that should be taken in account while thinking about the subject matter and they make a case for theories [23], leading to "a very satisfying sense of understanding" [13]. They are persuaders [24], instruments to challenge each other with our ideas and engage in interpersonal thinking. Examples of constructive thought experiments include Newton's cannon to discuss orbits and Einstein's elevator to formulate the equivalence principle [12]. *Destructive* thought experiments, on the other hand, are made for arguing against theories, usually by drawing out a contradiction in a theory and thereby refuting it or by "showing the theory in question is in conflict with other beliefs that we hold". [13] Examples of destructive thought experiments include the famous Schrodinger's cat paradox to show a bizarre consequence of the theory of quantum mechanics and Galileo's free fall thought experiment.

As a matter of fact, Galileo's thought experiment is both constructive and destructive. "Galileo asks us to imagine a heavy ball attached by a string to a light ball (…). What would happen if they were released together? Reasoning in the Aristotelian fashion leads to an absurdity. The lighter ball would slow up the heavy one, so the speed of the combined balls would be slower than the heavy ball alone (i.e., $H + L < H$). However, since the combined balls are heavier than the heavy ball alone, the combined object should fall faster than the heavy one (i.e., $H < H + L$). We have a straightforward con-

tradiction; the old theory is destroyed. Moreover, the new theory is established; the question of which falls faster is obviously resolved by having all objects fall at the same speed." [12]

There can be other *tools for thinking* that are not necessarily thought experiments, such as analogies, metaphors and *labels*. A label is "a vivid name for something helps you keep track of it while you turn it around in your mind trying to understand it" [23]. For example, constructive thought experiments can be *labeled* as *intuition pumps* [23], referring to their disposition to *pump* an intuition about a given subject matter to our brains. While we may employ any of these *tools for thinking* in our validation activities, we will concentrate mainly on those that use narratives.

In computer science, in particular, stories and thought experiments have been used to describe algorithmic problems. "The use of pithy and classic anecdotes set in familiar design situations is an excellent means for abstracting general principles while at the same time providing unifying themes and useful lessons that will be remembered." [66]. A classical example is "The Dining Philosophers", a thought experiment initially elaborated by Dijkstra and published by Tony Hoare to discuss concurrency and deadlocks in computer programs sharing resources. The story was originally published as follows: "In ancient times, a wealthy philanthropist endowed a College to accommodate five eminent philosophers. Each philosopher had a room in which he could engage in his professional activity of thinking; there was also a common dining room, furnished with a circular table, surrounded by five chairs, each labelled by the name of the philosopher who was to sit in it. The names of the philosophers were PHIL0, PHIL1, PHIL2, PHIL3, PHIL4, and they were disposed in this order anticlockwise around the table. To the left of each philosopher there was laid a golden fork, and in the centre stood a large bowl of spaghetti, which was constantly replenished. A philosopher was expected to spend most of his time thinking; but when he felt hungry, he went to the dining room, sat down in his own chair, picked up his own fork on his left, and plunged it into the spaghetti. But such is the tangled nature of spaghetti that a second fork is required to carry it to the mouth. The philosopher therefore had also to pick up the fork on his right. When he was finished he would put down both his forks, get up from his chair, and continue thinking. Of course, a fork can be used by only one philosopher at a time. If the other philosopher wants it, he just has to wait until the fork is available again. (…) Suppose all the philosophers get hungry at about the same time; they all sit down; they all pick up their own forks; and they all reach out for the other fork—which isn't there. In this undignified

situation, they will all inevitably starve. Although each actor is capable of further action, there is no action which any pair of them can agree to do next. However, our story does not end so sadly. Once the danger was detected, there were suggested many ways to avert it. For example, one of the philosophers could always pick up the wrong fork first—if only they could agree which one it should be! The purchase of a single additional fork was ruled out for similar reasons, whereas the purchase of five more forks was much too expensive. The solution finally adopted was the appointment of a footman, whose duty it was to assist each philosopher into and out of his chair." [53] In this thought experiments the philosophers are a methaphor to computer programs, which should "think" (compute) most of the time but need to use some shared resource every now and then (in this case the shared resources are the forks). He uses the metaphor to discuss means how we should approach concurrency problems. Other famous thought experiments in Computer Science include Searle's Chinese room [73], the Two Generals' Problem, the Travelling salesman problem, Yale shooting problem, among many others.

Conceptual Modeling is a branch of Computer Science which has its own thought experiments, although not as well documented. In [49] some thought experiments (such as the Counting Problem and The Color of the Rose) are used to defend some decisions for the Unified Foundational Ontology. In this context, thought experiments have been used to discuss a class of problems which are covered by the modeling language. To give an example, we introduce the following thought experiment, which has been adapted from [49]. This philosophical problem is known as *The Counting Problem* and has been an old discussion in conceptual modeling, according to [49]. Consider the following narrative:

"John boarded flight KL124 on April 22nd, 2004 from London to Amsterdam and boarded flight KL121 on November 19th, 2004 from Amsterdam back to London".

In the statement "KLM served four thousand passengers in 2004" would John count as one or as two passengers in the total amount? In other words, are we counting how many plane tickets were sold by KLM (and actually used to board a plane) or how many distinct people travelled with the company?

The argument put forth is that when counting how many Passengers flew with KLM in 2004, we could count the *qua-individuals*, not the people. In our example, John there are two different *John-qua-Passenger* entities, one for each flight.

In this case, the word “passenger” was used to refer to qua-individuals, but the word could also be used to refer to the person. For example, consider the following story: “In June 1943 a KLM operated airplane was shot down by the Luftwaffe resulting in the death of the 17 passengers on board”. In this case, it is not the *person-qua-passenger* that died, rather the person died.

3.5. Concluding remarks

In this chapter we have reviewed some ways humans may use stories to transfer knowledge and discuss conceptualizations and abstractions. Some stories, such as myths, may or may not have been authored with a clear intention of functioning as a tool, while thought experiments, on the other hand, were. We argue that we may draw from storytelling culture and existing thought experiments to author our own stories in our model validation efforts. Using such stories in validation activities may constitute thought experiments.

In our previous approach one could examine the simulations and compare them to the intended models; but to really validate a model, one must compare these simulations to the situations in reality, which can be problematic as discussed in Chapter 2. We argue that modelers may author stories to reveal and discuss the possible world structures implied by the conceptual model specification. In particular, this can be an effective means to bridge the gap in communication between modelers and subject matter experts. This is analogous to how philosophers and scientists create thought experiments to discuss theories with their peers.

In the following chapter, we discuss how we may design Natural Language Narratives with such specific purpose, when possible using the overarching structures discussed in section 3.3 as guidelines.

4. Authoring Natural Language Narratives for Conceptual Model validation

"A well-thought-out story doesn't need to resemble real life.
Life itself tries with all its might to resemble a well-crafted story."

— Isaac Babel

In this chapter, the argument of using Storytelling for validation defended in previous chapters is applied. Here, we introduce the concept of Natural Language Narrative, discussing how to author them. Natural Language Narratives here have both applications on their own and will be used as basis for the elaboration of Formal Story Specifications and Formal Narratives in the next chapters.

4.1. Introduction

A Natural Language Narrative is the thing that would probably come to mind when hearing the word "story" or "narrative", e.g., the contents of a fiction book is a Natural Language Narrative.

While narratives in natural language could serve many different purposes in general, such as entertainment, for example, here we use the term to designate the narratives elaborated with the sole purpose of helping to validate OntoUML Conceptual Models. The model validation approach defended in this work uses Natural Language Narratives as the first step. Natural Language Narratives serve as basis for the elaboration of Formal Story Specifications which are, in turn, used to generate Formal Narratives (i.e. simulations) that exemplify how the Natural Language Narrative can be realized in formal terms according to the conceptual model. Formal Narratives are diagrams that illustrate the Natural Language Narrative in a formal fashion.

In this context, Natural Language Narratives act as a *tool for thinking* in validation activities. They have two main purposes: reduce cognitive effort of the people involved in validating models and help in communication activities between modelers and Subject Matter Experts.

As we have discussed in Chapter 2, the validation of a conceptual model involves assessing whether the model captures accurately the conceptualization of a subject do-

main. This typically involves the communication between subject matter experts (authorities in the conceptualization of the subject domain) and modelers (who have the duty to capture the conceptualization faithfully). The role of the Natural Language Narratives in this setting facilitates this communication by creating some cohesion between the informal knowledge in the minds of the communicating stakeholders to the formalized knowledge in the conceptual model.

In section 4.2 we discuss some scenarios where authoring Natural Language Narratives can help validate conceptual models. In section 4.3 we introduce a running example that will be used to create a Natural Language Narrative in section 4.4 and which will also be used in the next chapters, to create a Formal Story Specification and generate Formal Narratives. Finally, in section 4.5 we offer concluding remarks on the subject.

4.2. Scenarios for Natural Language Narrative authoring

We envision three main scenarios for Natural Language Narrative authoring. (i) A modeler by himself creating Natural Language Narratives as a *tool for thinking* about the problem at hand (ii) a modeler creating Natural Language Narratives to help communicating with stakeholders in validation activities either illustrating (constructive thought experiment) or refuting (destructive thought experiment) a model and (iii) the Subject Domain Expert telling a story about his experiences in the domain, which helps modelers understand domain abstractions.

Either way, the plot of the narrative is closely related to its role in the validation activity. The narrative should cover a significant portion of concepts in the model to be assessed. By exercising the dynamics of object creation, role playing, phase changing etc. the interaction between the concepts in the model is revealed.

The first case of modeler-authored narratives involves only the modeler, where the Natural Language Narrative serves as a sort of *informal simulation*: the narrative acts as an articulation of elements of the conceptual model and therefore exercises domain abstractions in the mind of the modeler. This either increases the confidence of the modeler in the conceptual model or reveals some fraction of the model which he believes

should be clarified or improved. This can motivate some simulations to find errors or reveal a case which must be discussed with the Subject Matter Experts.

The second case of modeler-authored narratives involves both the modeler and the Subject Matter Expert where the Natural Language Narrative serves to improve the communication between the parties. In this case, the modeler may exercise fragments of the model he/she suspects may be incorrect or are hard to understand and could be clarified. He then creates simulations that stress these fragments or reveal the problems found. Next, the modeler authors a Natural Language Narrative to illustrate these simulations. The Natural Language Narrative connects the formal elements of the simulation to the domain abstractions in the mind of the Subject Matter Expert, improving the communication between them. To make the narratives interesting and engaging, the modeler may apply story structures such as those discussed in the previous chapter (e.g. Campbell's Hero's Journey) or resort to online material on plot such as the TV Tropes [77], which offers a large catalogue of story patterns to draw from.

Last, but not least, in the case of narratives authored by subject matter experts, a subject matter expert narrates real life events about a fragment of the model requested by a modeler or narrates situations in reality about concepts modelers find hard to understand. The narratives help to understand how these concepts are exercised in their real context.

4.3. Running Example: Software Configuration Model

In order to demonstrate the application of the technique, we introduce a running example in the domain of Software Configuration Management. "*Software Configuration Management (SCM) is a fundamental process in the development of complex products. It provides technical and administrative guidelines to manage a product's lifecycle. SCM guides and controls the evolution of a product's configuration, promoting means to prevent disorder in its development. This control occurs through a process that identifies and defines a products configuration items, controls the modifications on the configuration items during its lifecycle, registers and reports their states and verifies the consistency of those items.*" [14] The model is presented briefly below and will be used throughout the work as an example.

A key notion of Software Confirmation Management is the notion of **Item**. An item is a generic term used to represent parts of a product or information generated in their

development. The diagram in Fig. 16 specifies different kinds of **Items** that can be versioned: **Software Tools** and **Artifacts** such as **Source Code**, **Document** and **Diagram**. **Item** and **Artifact** are defined in this model as Categories, and as such are rigid classes (i.e., they apply necessarily to their instances). **Software Tools, Source Code, Document and Diagram** are stereotyped as Kind and therefore are also rigid but they define a principle of identity for their instances, while Categories are classes that subsume instances with different principles of identity. An Item that has been selected by a **Configuration Manager** assumes the role of a **Configuration Item**. Configuration Manager is the role a **Person** assumes in the context of that selection. (The Person class is omitted from the diagram and appears in italics on the top classes that specialize it.) Roles are Anti-Rigid (a.k.a. dynamic) classes i.e. they apply contingently to their instances; thus, it is not necessary for persons to play the role of configuration manager. The relationship between the Configuration Manager and the Configuration Item is reified as a **Configuration Selection.**

Each Configuration Item is characterized by some **Version.** Version is stereotyped as Mode, meaning they are existentially dependent and inhere in the thing they characterize. In this case, Versions can only exist in Configuration Items. Versions are part of some **Branch.** Branches, on the other hand are part of some **Repository.** Stereotyped as Collectives, their instances are collections formed by uniform parts. Versions can be **submitted for change**, when **requested**. These **Change Requests** can be **Evaluated** by a **Configuration Manager**, which is an **Evaluator**[7] in this context. A **Developer** is a **Person** that may **Check Out** versions, **modify** them and **Check In Modifications** (a checked-in modification is called a **Registered Modification**)**.** Versions that are checked out are **Checked-Out Versions** and generate **Copies.** A **Copy** that has been modified assumes the role of **Modified Copy**, and when checked-in, is **consumed** and makes the requested **change implemented**. A **Verifier** may assess an **implemented change**, making it **verified**.

[7] Both Evaluator and Verifier are subclasses of Configuration Manager, even though the generalization is not explicit in this diagram.

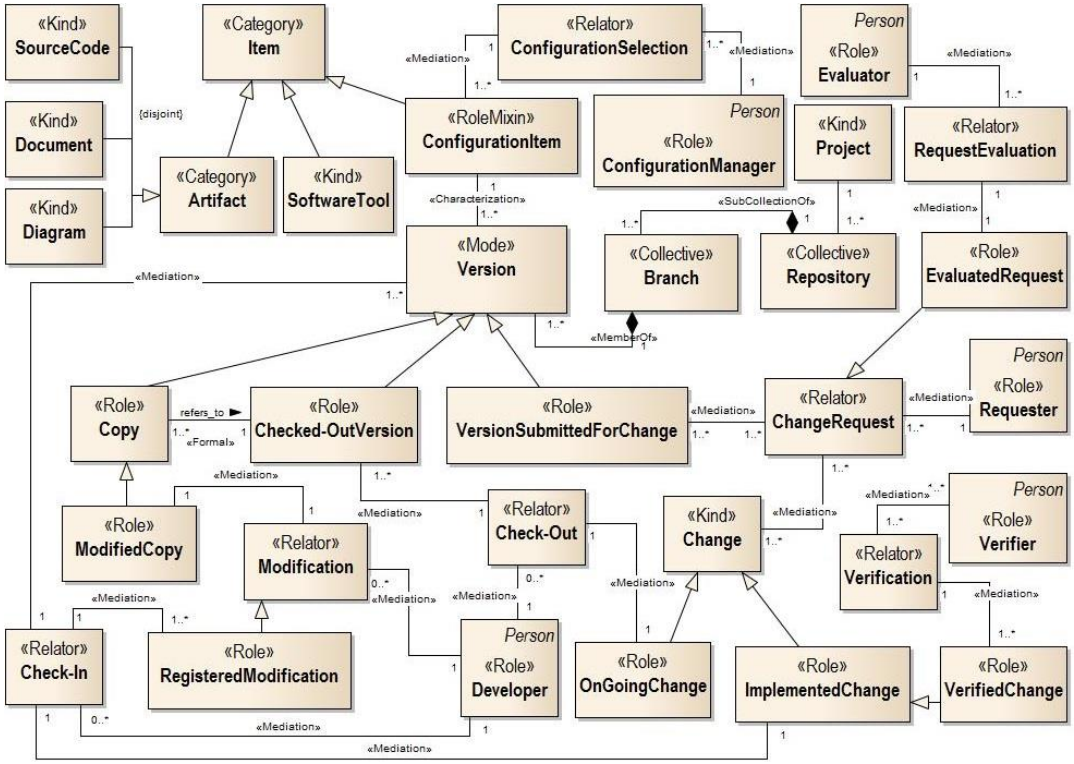

Fig. 16. A model for Software Configuration Management extracted from [14]

4.4. Natural Language Narrative for the Software Configuration Management model

Drawing from our running example, we produced the following Natural Language Narrative. It exercises most of the classes of the model presented in Fig. 16 (the model in our running example). Whenever a class is used in the narrative, it is highlighted in bold. Some concepts are used in an informal manner (e.g. version-controlled, branch the project). This is intentional: forcing the Natural Language Narrative to use the vocabulary exactly as defined by the Conceptual Model can make the narrative artificial. Additionally, this exercise may show domain abstractions which are not captured and therefore reveal opportunities for extending the scope of the model. This narrative is the basis for the Formal Story Specification presented in Chapter 5, which will in its turn be used to generate Formal Narratives (simulations) in Chapter 6.

"John, Mary, Fred and Thomas work at OntoSoft company as ***developers****. They are working on an information system for a bakery to manage its finances and supply-chain processes. The system they*

*are producing already manages the financial aspects, and currently they are developing new **artifacts** (such as **diagrams**, **documents** and **source code**) to manage the supply-chain processes. Thomas is the **Configuration Manager** and he **selects** some of the **artifacts** they created to be part the project's repository, where they are **version**-controlled.*

*As the team focuses efforts on the bakery's supply-chain processes, Fred finds a deadlock in a process diagram for buying raw materials and files a **change request** for it, describing the problem he found and the **change** that should be implemented. Mary **evaluates** the request and asks John to **check out** the **diagram** in the version control system to **modify** it. After doing the necessary adjustments, he **checks in** the **modified version** and Mary is assigned to **verify** whether John has met the **change request**.*

*Mary **verifies** the **diagram** and notices that John's **modifications** introduced bugs in the already-approved finance processes. These **changes** have a deep impact in the approved parts of the software so Mary rejects the **version** and asks John to **branch** the **project** and try again from a different angle."*

The narrative above features a sequence of events that show how the different classes interact in the instance level and how the instances of a class may change. From a technical point of view, the classes of the conceptual model are exemplified in a context and this activity facilitates model comprehension.

For example, consider how in the first paragraph the narrative navigates the model showing, for example, how a Diagram is an Artifact (Diagram specializes Artifact); which is an Item (Artifact specializes Item); which can be a Configuration Item (Configuration Item specializes Item) in the context of a Configuration Selection. While navigating the model to understand that the elements interact this way can require a certain expertise in the OntoUML language, reading the Natural Language Narrative reveals this possibility in an intuitive way.

In the second paragraph, we can see the difference between the Change Request, which is a communication artifact (a request from Fred) and the Change itself, which is a specification of the problem to be solved (remove deadlock).

We can also use the narrative to discuss the coverage of the model. For example, we mentioned that Mary assigned John to deal with Fred's request but there is no concept capturing the assignment. Also, in the end, Mary rejects the version. There is no concept of rejecting a version or rolling back. Is that what it means for a version to be destroyed? If we consider what could happen after the end of the story, when John eventually branches the model, what would be the relation between the new branch and the old one? Could John commit his modifications to this new branch? Can John branch the bugged project to fix it and Mary carry on working in the master branch after rolling back on Johns modifications?

Finally, from a narrative point of view, we exercised some dramatic elements as well: the first paragraph sets the scenario and acts as the *exposition* in Freytag's pyramid. On the second paragraph, we have the *rising action*, when Fred finds a deadlock in the process and John is called to action. He proceeds to checkout the diagram, modify it and on the *climax*, he checks in the modified version and Mary inspects it and rejects it. The *falling action* then throws John back to the start.

4.5. Concluding remarks

Natural Language Narratives should be motivated by needs of validating and clarifying the model and therefore the plot of the Narrative should stem from those needs. When using these narratives to communicate with Subject Matter Experts, some plot devices and narrative design patterns which have been discussed in previous chapters can be employed to create engaging narratives. While these patterns and devices can help a narrator, it is outside of the scope of this work to give methodological means for creating the Natural Language Narratives. They are simply too dependent on the domain of discourse and creating such narratives is a creative effort which requires practice to master.

In the following chapters we will use the narrative elaborated in this chapter as basis for creating a Formal Story Specification (Chapter 5) that will be used to generate Formal Narratives (Chapter 6). While in this chapter we have only exemplified how Natural Language Narratives may work as constructive thought experiments, in the context of

Iterative Validation in Chapter 6 we will exemplify how they may be used as destructive thought experiments as well, refuting some modeling choices and motivating change.

5. Formal Story Specification and transformation

"Out of clutter, find simplicity."

— Albert Einstein

In this chapter, we introduce a Formal Story Specification language and exemplify its application using our running example. The transformation from Formal Story Specifications to Alloy is also specified.

5.1. Introduction

A Formal Story Specification is a formal representation of the semantic aspects of a story. Formal Story Specifications have a practical purpose: they are used as basis for the generation of Formal Narratives (i.e. simulations in conformance with the Formal Story Specification), which in turn will be used to assess the model. In this sense, they create a correspondence between an informal Natural Language Narrative and a Formal Narrative (a simulation of the conceptual model in the instance level).

The name we choose for the artifact, "Formal Story Specification", reveals something about its nature. We use the term "story" because it is an artifact that has the property of defining many different narratives, as regular stories do. However, stories are abstract entities that can only exist in the mind of people, therefore we call the artifact we produce "specifications". Finally, it is "formal" because the artifact is only concerned with specifying the formal semantic aspects of the story.

The rest of this chapter is structured as follows: in section 5.2, we discuss different scenarios that justify the creation of Formal Story Specifications; in section 5.3, we define the Formal Story Specification Language; in section 5.4, we revisit our running example, and exemplify the use of the language defined in this chapter; in section 5.5, we present a prototype application we developed named Story Modeler designed to elaborate Formal Story Specifications; in section 5.6, we discuss the transformation from Formal Story Specifications to Alloy; in section 5.7 we discuss how to use the Alloy Analyzer to generate Formal Narratives based on Formal Story Specifications, dis-

cussing the problems that arise in this activity; finally, in section 5.8, we offer concluding remarks about the chapter.

5.2. Scenarios for creating Formal Story Specifications

We have identified two main scenarios to creating Formal Story Specifications. In the first case, they are based on an existing Natural Language Narrative. In this alternative, the modeler captures what happens in the narrative by specifying individuals and their relationships using the concepts present in the conceptual model. When formalizing an existing Natural Language Narrative much detail is lost since Formal Story Specifications only contain semantic aspects of the narrative that are relevant to the conceptual model. However, this process may create information that is more precise than their natural language counterparts. Inconsistencies, ambiguities and suppositions are removed in this stage, making the modeler commit to a certain interpretation, fixing part of the story's semantic content. The Formal Story Specification acts like a sort of explanation, revealing the elements involved in the story and its unfolding.

In the second case, a modeler may take the reverse approach: first create a Formal Story Specification and later elaborate a Natural Language Narrative based on it. One reason to use this approach is to validate the modeler's understanding of the domain: he/she can narrate this story to a subject matter expert and assess if he/she understood correctly what was implied by the model. Another reason to use this approach is to motivate and justify changes in the model. If the model validation finds an inconsistency, the modeler may create a narrative that explores the inconsistency to expose it to others.

Either way, the elaboration of Formal Story Specifications should be focused on its practical validation purposes. It is an instrument to generate Formal Narratives and concerns with computation aspects apply. For example, in the Natural Language Narrative for our running example many items are mentioned (diagrams, documents and source code). However, only the process diagram takes a central role in the story and the other elements could therefore be omitted from the Formal Story Specification to save computational resources in the Formal Narrative generation.

5.3. Formal Story Specification Language

Formal Story Specifications are model instances of our special-purpose language, whose meta-model is presented in Fig. 17. In this language, the user may specify *nodes* and *links* between nodes. The elements make explicit connections to the conceptual model, specifying which classes they instantiate. Each node may be assigned to instantiate some Rigid classes from the conceptual model, while links connect nodes and can be assigned to instantiate Associations. Individuals (nodes and links) can be present in *worlds* and a world sequence represents the unfolding of the story (the world sequence is represented using "next" and "previous" relations). A world is a snapshot of the story, capturing the state of things in a particular point in the story. As the story progresses, elements may be created, changed or destroyed. Change is represented as classification statements that may be made about the nodes, which specify contingent characteristics of it, i.e., the Non-Rigid (Anti Rigid or Semi Rigid) classes a node instantiates.

The language takes an *open world assumption*, meaning the specification is partial and what is not specified cannot be assumed to be false. In other words, the relations in this model capture the "facts" that the modeler asserts about the story. The modeler can assert a fact (e.g., "John" is an instance of "Developer") or assert its negation (e.g., "Mary" is not an instance of "Developer") by using the appropriate relations. Whenever the model is silent with respect to a particular choice, e.g., when nothing is said about whether "John" is a developer, the simulator will allow both options, meaning either case can appear in a Formal Narrative of such story. This is useful to partially formalize a story and simulate to see the possible rearrangements of states of affairs generated by the simulator. As a consequence, a single Formal Story Specification can define many different Formal Narratives.

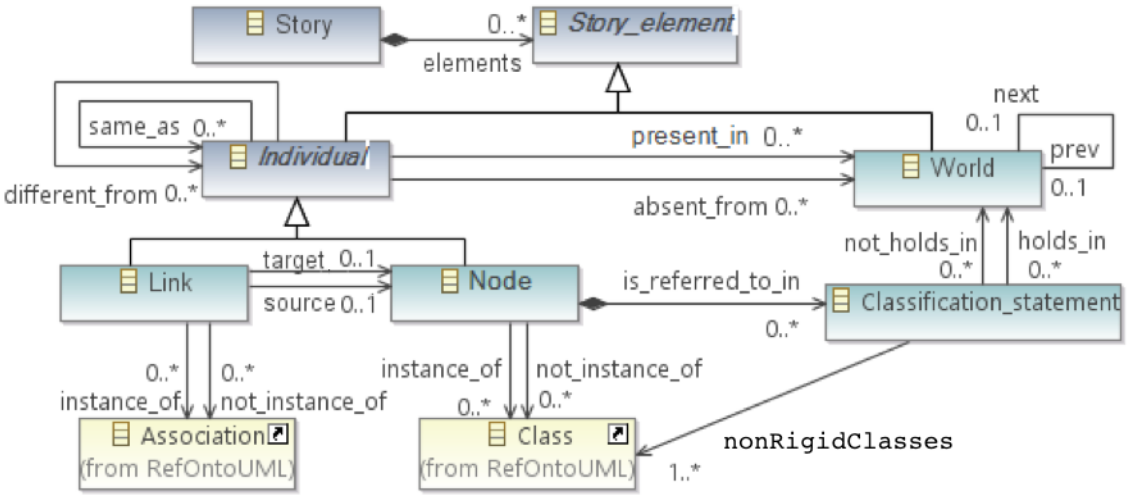

Fig. 17. Meta-model of the Formal Story Specification language

5.4. Running Example

To exemplify the creation of a Formal Story Specification, we will build on our running example by formalizing the Natural Language Narrative presented in section 4. The table below shows fragments of the Natural Language Narrative on the left column and the corresponding Formal Story Specification on the right column. We use an informal textual syntax to exemplify the use of the Formal Story Specification Language.

Natural Language Narrative Fragments	Elements of the Formal Story Specification
John, Mary, Fred and Thomas work at OntoSoft company as developers.	John is Node Mary is Node Fred is Node Thomas is Node John different_from Mary John different_from Thomas John different_from Fred Thomas different_from Mary Thomas different_from John John different_from Mary John instance_of Person Mary instance_of Person Fred instance_of Person Thomas instance_of Person

	W1 is World
	W2 is World
	W3 is World
	W4 is World
	W5 is World
	W6 is World
	W1 next W2
	W2 next W3
	W3 next W4
	W4 next W5
	W5 next W6
	
	John present_in W1, W2, W3, W4, W5, W6
	Mary present_in W1, W2, W3, W4, W5, W6
	Fred present_in W1, W2, W3, W4, W5, W6
	Thomas present_in W1, W2, W3, W4, W5, W6
	
	John is_referred_to_in {
	nonRigidClasses = Developer
	} holds_in W1, W2, W3, W4, W5, W6
	Mary is_referred_to_in {
	nonRigidClasses = Developer
	} holds_in W1, W2, W3, W4, W5, W6
	Fred is_referred_to_in {
	nonRigidClasses = Developer
	} holds_in W1, W2, W3, W4, W5, W6
	Thomas is_referred_to_in {
	nonRigidClasses = Developer
	} holds_in W1, W2, W3, W4, W5, W6

They are working on an information system for a bakery to manage its finances and supply-chain processes. The system they are producing already manages the finance aspects, and currently they are developing new artifacts (such as diagrams, documents and source code) to manage the supply-chain processes.	ProcessDiagram is Node ProcessDiagram instance_of Diagram ProcessDiagram present_in W1, W2, W3, W4, W5, W6
Thomas is the Configuration Manager and he selects some of the artifacts they created to be part the project's repository, where they are version-	Thomas is_referred_to_in { nonRigidClasses = ConfigurationManager } holds_in W1, W2, W3, W4, W5, W6

controlled.	John is_referred_to_in {
	nonRigidClasses = ConfigurationManager
	} not_holds_in W1, W2, W3, W4, W5, W6
	Fred is_referred_to_in {
	nonRigidClasses = ConfigurationManager
	} not_holds_in W1, W2, W3, W4, W5, W6
	Mary is_referred_to_in {
	nonRigidClasses = ConfigurationManager
	} holds_in W1, W2, W3, W4, W5, W6

As the team focuses efforts on the bakery's supply-chain processes, Fred finds a deadlock in a process diagram for buying raw materials and files a change request for it, describing the problem he found and the change that should be implemented.	ChangeRequest001 is Node ChangeRequest001 instance_of ChangeRequest ChangeRequest001 present_in W2, W3, W4, W5, W6 ChangeRequest001 not_present_in W1 Link001 is Link Link001 source Fred Link001 target ChangeRequest001 Link001 present_in W2, W3, W4, W5, W6 Link001 not_present_in W1 Change001 is Node Change001 instance_of Change Change001 present_in W2, W3, W4, W5, W6 Change001 not_present_in W1 Link002 is Link Link002 source Change001 Link002 target ChangeRequest001 Link002 present_in W2, W3, W4, W5, W6 Link002 not_present_in W1
Mary evaluates the request and asks John to check out the diagram in the version control system to modify it.	Evaluation001 is Node Evaluation001 instance_of RequestEvaluation Evaluation001 present_in W3, W4, W5, W6 Evaluation001 not_present_in W1, W2 Link003 is Link Link003 source Evaluation001 Link003 target ChangeRequest001 Link003 present_in W3, W4, W5, W6

	Link003 not_present_in W1, W2 CheckOut001 is Node CheckOut001 instance_of CheckOut CheckOut001 present_in W3, W4, W5, W6 CheckOut001 not_present_in W1, W2 Link004 is Link Link004 source John Link004 target CheckOut001 Link004 present_in W3, W4, W5, W6 Link004 not_present_in W1, W2 Link005 is Link Link005 source CheckOut001 Link005 target Change001 Link005 present_in W3, W4, W5, W6 Link005 not_present_in W1, W2
After doing the necessary adjustments, he checks in the modified version	Modification001 is Node Modification001 instance_of Modification Modification001 present_in W4, W5, W6 Modification001 not_present_in W1, W2, W3 Link006 is Link Link006 source Modification001 Link006 target John Link006 present_in W4, W5, W6 Link006 not_present_in W1, W2, W3 CheckIn001 is Node CheckIn001 instance_of CheckIn CheckIn001 present_in W5, W6 CheckIn001 not_present_in W1, W2, W3, W4 Link007 is Link Link007 source CheckIn001 Link007 target Modification001 Link007 present_in W5, W6 Link007 not_present_in W1, W2, W3, W4
and Mary is assigned to verify whether John has	Verification001 is Node

met the change request. Mary verifies the diagram and notices that John's modifications introduced bugs in the already-approved finance processes. These changes have a deep impact in the approved parts of the software so Mary rejects the version and asks John to branch the project and try again from a different angle.	Verification001 instance_of Verification Verification001 present_in W6 Verification001 not_present_in W1, W2, W3, W4, W5 Link008 is Link Link008 source Verification001 Link008 target Mary Link008 present_in W6 Link008 not_present_in W1, W2, W3, W4, W5 Link009 is Link Link009 source Verification001 Link009 target Change001 Link009 present_in W6 Link009 not_present_in W1, W2, W3, W4, W5

5.5. Story Modeler

We have implemented a prototype software application named Story Modeler to create Formal Story Specifications. A screenshot of it is available in Fig. 18. The tool represents the Formal Story Specification internally as an instance of the abstract syntax meta-model presented in Fig. 17 (using code generated with the Eclipse Modeling Framework). The tree table specifies the Individuals (Nodes and Links) in each row and the Worlds in the columns. Each field determines if the element exists (a checkmark), does not exist (a red 'x'), or if it is left unspecified (an empty box); for each world column. The classes each story element instantiates, as well as the non-rigid classes for the classification statements, can be defined in the list below the story elements panel.

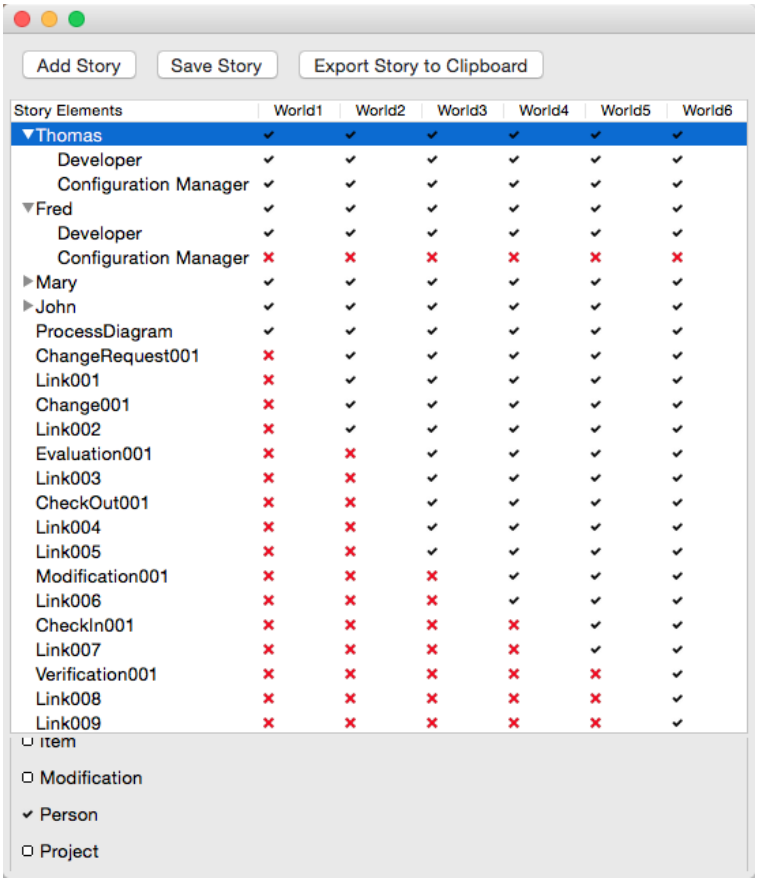

Fig. 18. The Formal Story Specification interface

5.6. Formal Story Specification transformation to Alloy

Formal Story Specifications may be transformed to Alloy using the Story Modeler application. For an overview of the Alloy language refer to Appendix A. The Story Modeler generates a predicate that can be combined with the Alloy model generated by OntoUML2Alloy transformation, available in the Menthor Editor. Running the Alloy Analyzer using such generated predicate constraints the simulation to behave as specified by the Formal Story Specification. (Fig. 3)

The model transformation OntoUML2Alloy has been specified elsewhere and it generates a signature structure that corresponds to the classes, its relationships and constraints that are implied by the OntoUML syntax. A sequence of Worlds represents state-of-affairs. The transformation can be roughly summarized as follows.

The three[8] most relevant signatures are Object, Property and World. In the World signature, a field exists represents which Object or Property exists in each world. Classes of the conceptual model are represented as a fields in the World signature. There-

[8] Datatypes are also translated to signatures but have been left out of scope for this work.

fore, if an Atom ‘x’ is part of a field ‘C’, for a given world ‘w’, that means ‘x’ is an instance of ‘C’ in ‘w’. The set w.C represents the extension of a class C in the World w. By running a model (i.e. simulating a model), the Alloy Analyzer will find structures that satisfy the model’s constraints and represent a model instantiation.

The model transformation for Formal Story Specifications defined in this work takes as input a model specified using the Formal Story Specification language presented in the previous section. Each Node, Link or World becomes a variable of the story predicate and the constraints are made in terms of these variables. By defining when they exist, how they are related to each other and which classes they instantiate, we define the story. Below, we present an example story predicate that defines a sequence of three worlds and John, a Person. In the case of worlds, we specify the type to be the World signature but nodes are specified to be in the set univ - World[9], which is basically an atom of any signature, except World.

```
pred story [World0:one World,
        World1:one World,
        World2:one World,
        John:one univ-World]
{
   John in World.Person
   John in World0.exists
   World0.next = World1
   World1.next = World2
}
```

First of all, we discuss how to represent class instantiation. That includes static classification for nodes and links and dynamic classification for nodes, in the form of Classification_statements. As mentioned before, in the OntoUML2Alloy transformation each class is defined as a field in the World signature and to be instance of a class “ClassName” in a World w means to be included in the set defined by w.ClassName. Since an instance of a rigid class is necessarily an instance of it in every possible world we can use the signature World (which is the set of all worlds) instead. By defining an Individual to be a member of the set World.ClassName, we specify it to be a in the ex-

[9] To cover datatypes, one would have to exclude them here, in the same way we exclude World.

tension of the class ClassName in some World[10]. For example, a node "n" with an instance_of relation to a class "ClassName" would be translated to the following statement:

n in World.ClassName

In the case of not_instance_of, we may simply insert the "not" keyword in the statement to prevent the node from being an instance of the class.

n not in World.ClassName

Dynamic classification is defined using Classification_statements. That way, we may specify in which Worlds a node instantiates a Non Rigid class. The classification statement itself is not represented in the Alloy transformation, it is a syntactical element used only to define in which worlds a given node instantiates a class. The node's dynamic classification is represented, instead. The generated constraint is very similar to the rigid classification statement, but it is instead targeted to a specific world "w".

n in w.ClassName

n not in w.ClassName

To specify the existence of a node in a given world w, we may use the "exists" field of the World signature. Being included in such property in a given world means to exist in that World. Thus, if the node n is present_in w1 and absent_from w2, the following statements would hold

n in w1.exists

n not in w2.exists

Regarding the identity of a node (defined by their same_as and different_from relations), we may simply use the "=" symbol. Therefore, "x = y" is generated for every same_as relation and "x not = y" for every different_from relation.[11]

Links are represented as a tuple between two atoms. A link between "x" and "y" instance of the Association "Assoc" in a World "w" is defined as

x->y in w.Assoc

In the case of undirected binary relationships, we may specify a tuple and its inverse and constrain their union to have an intersection with the set that defines the association, that way the direction of the link is not relevant.

some (x->y + y->x) & w.Assoc

---

[10] This alone would only guarantee that the Individual would be an instance of the class in some world, not in every world. However, constraints generated by the OntoUML2Alloy transformation guarantee that if the Individual is an instance of the set in some world, it will be in every possible world.

[11] . While the meta model defines these relations, the current version of the software does not use them yet and every node is defined as different_from each other by default.

If a Link is defined without specifying the Association it instantiates, an auxiliary predicate is used. The auxiliary predicate is defined as part of the translation and essentially creates a set of the union of every Association. If the tuple (or its inverse) is part of such set, it exists as an association of some (undefined) type in the world w.

```
pred direct_rel_in_w[x1,x2: univ , w:World]{
    some (x1->x2 + x2->x1)
    & (w.Association1+
    w.Association2+
(…)
    w.AssociationN)
}
```

If the link is defined, but there is no specification of which Worlds the link is *present_in*, it may (or may not) exist in any possible world. In this case we may use the following predicate as shorthand to using the previously defined predicate with World signature set as the *w* variable.

```
pred direct_rel[x1,x2: univ-World]{
    direct_rel_in_w[x1,x2,World]
}
```

If the association that doesn't have an Individual defined as either the target or the source, the transformation uses the (univ – World) set, i.e. "any" node.

Last but not least, our application assumes Worlds are ordered (using the .next field of the World signature) and their order is defined by their position in the Story Modeler interface (from left to right). For example:

```
World1.next = World2
```

The table below shows how each of the elements in the Formal Story Specification is translated to Alloy.

Elements of the Formal Story Specification	**Alloy Predicate**
	pred story [
W1 is World	World1:one World,
W2 is World	World2:one World,
W3 is World	World3:one World,
W4 is World	World4:one World,
W5 is World	World5:one World,

W6 is World	World6:one World,
Thomas is Node	Thomas:one univ-World,
Fred is Node	Fred:one univ-World,
Mary is Node	Mary:one univ-World,
John is Node	John:one univ-World,
ProcessDiagram is Node	ProcessDiagram:one univ-World,
ChangeRequest001 is Node	ChangeRequest001:one univ-World,
Change001 is Node	Change001:one univ-World,
Evaluation001 is Node	Evaluation001:one univ-World,
CheckOut001 is Node	CheckOut001:one univ-World,
Modification001 is Node	Modification001:one univ-World,
CheckIn001 is Node	CheckIn001:one univ-World,
Verification001 is Node	Verification001:one univ-World,]{
John different_from Mary	disj[John,Mary]
John different_from Thomas	disj[John,Thomas]
John different_from Fred	disj[John,Fred]
Thomas different_from Mary	dijs[Thomas,Mary]
Thomas different_from Fred	disj[Thomas,John]
Fred different_from Mary	disj[Fred,Mary]
John instance_of Person	John in World.Person
Mary instance_of Person	Thomas in World.Person
Fred instance_of Person	Fred in World.Person
Thomas instance_of Person	Mary in World.Person
	World1 in CurrentWorld
W1 next W2	World1.next = World2
W2 next W3	World2.next = World3
W3 next W4	World3.next = World4
W4 next W5	World4.next = World5
W5 next W6	World5.next = World6
John present_in W1, W2, W3, W4, W5, W6	John in World1.exists John in World2.exists John in World3.exists John in World4.exists John in World5.exists John in World6.exists
Mary present_in W1, W2, W3,	Mary in World1.exists

W4, W5, W6	Mary in World2.exists Mary in World3.exists Mary in World4.exists Mary in World5.exists Mary in World6.exists
Fred present_in W1, W2, W3, W4, W5, W6	Fred in World1.exists Fred in World2.exists Fred in World3.exists Fred in World4.exists Fred in World5.exists Fred in World6.exists
Thomas present_in W1, W2, W3, W4, W5, W6	Thomas in World1.exists Thomas in World2.exists Thomas in World3.exists Thomas in World4.exists Thomas in World5.exists Thomas in World6.exists
John is_referred_to_in { nonRigidClasses = Developer } holds_in W1, W2, W3, W4, W5, W6	John in World1.Developer John in World2.Developer John in World3.Developer John in World4.Developer John in World5.Developer John in World6.Developer
Mary is_referred_to_in { nonRigidClasses = Developer } holds_in W1, W2, W3, W4, W5, W6	Mary in World1.Developer Mary in World2.Developer Mary in World3.Developer Mary in World4.Developer Mary in World5.Developer Mary in World6.Developer
Fred is_referred_to_in { nonRigidClasses = Developer } holds_in W1, W2, W3, W4, W5, W6	Fred in World1.Developer Fred in World2.Developer Fred in World3.Developer Fred in World4.Developer Fred in World5.Developer Fred in World6.Developer

Thomas is_referred_to_in { nonRigidClasses = Developer } holds_in W1, W2, W3, W4, W5, W6	Thomas in World1.Developer Thomas in World2.Developer Thomas in World3.Developer Thomas in World4.Developer Thomas in World5.Developer Thomas in World6.Developer
ProcessDiagram instance_of Diagram ProcessDiagram present_in W1, W2, W3, W4, W5, W6	ProcessDiagram in World.Diagram ProcessDiagram in World1.exists ProcessDiagram in World2.exists ProcessDiagram in World3.exists ProcessDiagram in World4.exists ProcessDiagram in World5.exists ProcessDiagram in World6.exists
Thomas is_referred_to_in { nonRigidClasses = ConfigurationManager } holds_in W1, W2, W3, W4, W5, W6	Thomas in World1.ConfigurationManager Thomas in World2.ConfigurationManager Thomas in World3.ConfigurationManager Thomas in World4.ConfigurationManager Thomas in World5.ConfigurationManager Thomas in World6.ConfigurationManager
John is_referred_to_in { nonRigidClasses = ConfigurationManager } not_holds_in W1, W2, W3, W4, W5, W6	not John in World1.ConfigurationManager not John in World2.ConfigurationManager not John in World3.ConfigurationManager not John in World4.ConfigurationManager not John in World5.ConfigurationManager not John in World6.ConfigurationManager
Fred is_referred_to_in { nonRigidClasses = ConfigurationManager } not_holds_in W1, W2, W3, W4, W5, W6	not Fred in World1.ConfigurationManager not Fred in World2.ConfigurationManager not Fred in World3.ConfigurationManager not Fred in World4.ConfigurationManager not Fred in World5.ConfigurationManager not Fred in World6.ConfigurationManager
Mary is_referred_to_in {	Mary in World1.ConfigurationManager

nonRigidClasses = Configura-tionManager } **holds_in** W1, W2, W3, W4, W5, W6	Mary in World2.ConfigurationManager Mary in World3.ConfigurationManager Mary in World4.ConfigurationManager Mary in World5.ConfigurationManager Mary in World6.ConfigurationManager
ChangeRequest001 instance_of ChangeRequest	ChangeRequest001 in World.ChangeRequest
ChangeRequest001 present_in W2, W3, W4, W5, W6	ChangeRequest001 in World2.exists ChangeRequest001 in World3.exists ChangeRequest001 in World4.exists ChangeRequest001 in World5.exists ChangeRequest001 in World6.exists
ChangeRequest001 not_present_in W1	ChangeRequest001 not in World1.exists
Link001 is Link Link001 source Fred Link001 target ChangeRequest001 Link001 present_in W2, W3, W4, W5, W6 Link001 not_present_in W1	direct_rel_in_w[(Fred),(ChangeRequest001),World2] direct_rel_in_w[(Fred),(ChangeRequest001),World3] direct_rel_in_w[(Fred),(ChangeRequest001),World4] direct_rel_in_w[(Fred),(ChangeRequest001),World5] direct_rel_in_w[(Fred),(ChangeRequest001),World6] not direct_rel_in_w[(Fred),(ChangeRequest001),World1]
Change001 instance_of Change Change001 present_in W2, W3, W4, W5, W6	Change001 in World.Change Change001 in World2.exists Change001 in World3.exists Change001 in World4.exists Change001 in World5.exists Change001 in World6.exists
Change001 not_present_in W1	Change001 not in World1.exists
Link002 is Link Link002 source Change001 Link002 target ChangeRequest001 Link002 present_in W2, W3, W4, W5, W6 Link002 not_present_in W1	direct_rel_in_w[(ChangeRequest001),(Change001),World2] direct_rel_in_w[(ChangeRequest001),(Change001),World3] direct_rel_in_w[(ChangeRequest001),(Change001),World4] direct_rel_in_w[(ChangeRequest001),(Change001),World5] direct_rel_in_w[(ChangeRequest001),(Change001),World6] not direct_rel_in_w[(ChangeRequest001),(Change001),World1]
Evaluation001 instance_of	Evaluation001 in World.RequestEvaluation

RequestEvaluation	
Evaluation001 present_in W3, W4, W5, W6	Evaluation001 in World3.exists Evaluation001 in World4.exists Evaluation001 in World5.exists Evaluation001 in World6.exists
Evaluation001 not_present_in W1, W2	Evaluation001 not in World1.exists Evaluation001 not in World2.exists
Link003 is Link	direct_rel_in_w[(ChangeRequest001),(Evaluation001),World3]
Link003 source Evaluation001	direct_rel_in_w[(ChangeRequest001),(Evaluation001),World4]
Link003 target ChangeRequest001	direct_rel_in_w[(ChangeRequest001),(Evaluation001),World5]
Link003 present_in W3, W4, W5, W6	direct_rel_in_w[(ChangeRequest001),(Evaluation001),World6]
Link003 not_present_in W1, W2	not direct_rel_in_w[(ChangeRequest001),(Evaluation001),World1] not direct_rel_in_w[(ChangeRequest001),(Evaluation001),World2]
CheckOut001 instance_of CheckOut	CheckOut001 in World.CheckOut
CheckOut001 present_in W3, W4, W5, W6	CheckOut001 in World3.exists CheckOut001 in World4.exists CheckOut001 in World5.exists CheckOut001 in World6.exists
CheckOut001 not_present_in W1, W2	CheckOut001 not in World1.exists CheckOut001 not in World2.exists
Link004 is Link	direct_rel_in_w[(John),(CheckOut001),World3]
Link004 source John	direct_rel_in_w[(John),(CheckOut001),World4]
Link004 target CheckOut001	direct_rel_in_w[(John),(CheckOut001),World5]
Link004 present_in W3, W4, W5, W6	direct_rel_in_w[(John),(CheckOut001),World6]
Link004 not_present_in W1, W2	not direct_rel_in_w[(John),(CheckOut001),World1] not direct_rel_in_w[(John),(CheckOut001),World2]
Link005 is Link	direct_rel_in_w[(John),(CheckOut001),World3]
Link005 source CheckOut001	direct_rel_in_w[(John),(CheckOut001),World4]
Link005 target Change001	direct_rel_in_w[(John),(CheckOut001),World5]
Link005 present_in W3, W4, W5, W6	direct_rel_in_w[(John),(CheckOut001),World6]
Link005 not_present_in W1, W2	not direct_rel_in_w[(John),(CheckOut001),World1] not direct_rel_in_w[(John),(CheckOut001),World2]

Modification001 instance_of Modification	Modification001 in World.Modification
Modification001 present_in W4, W5, W6	Modification001 in World4.exists Modification001 in World5.exists Modification001 in World6.exists
Modification001 not_present_in W1, W2, W3	Modification001 not in World1.exists Modification001 not in World2.exists Modification001 not in World3.exists
Link006 is Link Link006 source Modification001 Link006 target John Link006 present_in W4, W5, W6 Link006 not_present_in W1, W2, W3	direct_rel_in_w[(John),(Modification001),World4] direct_rel_in_w[(John),(Modification001),World5] direct_rel_in_w[(John),(Modification001),World6] not direct_rel_in_w[(John),(Modification001),World1] not direct_rel_in_w[(John),(Modification001),World2] not direct_rel_in_w[(John),(Modification001),World3]
CheckIn001 instance_of CheckIn CheckIn001 present_in W5, W6 CheckIn001 not_present_in W1, W2, W3, W4	CheckIn001 in World.CheckIn CheckIn001 in World5.exists CheckIn001 in World6.exists CheckIn001 not in World1.exists CheckIn001 not in World2.exists CheckIn001 not in World3.exists CheckIn001 not in World4.exists
Link007 is Link Link007 source CheckIn001 Link007 target Modification001 Link007 present_in W5, W6 Link007 not_present_in W1, W2, W3, W4	direct_rel_in_w[(Modification001),(CheckIn001),World5] direct_rel_in_w[(Modification001),(CheckIn001),World6] not direct_rel_in_w[(Modification001),(CheckIn001),World1] not direct_rel_in_w[(Modification001),(CheckIn001),World2] not direct_rel_in_w[(Modification001),(CheckIn001),World3] not direct_rel_in_w[(Modification001),(CheckIn001),World4]
Verification001 instance_of Verification	Verification001 in World.Verification
Verification001 present_in W6	Verification001 in World6.exists
Verification001 not_present_in W1, W2, W3, W4, W5	Verification001 not in World1.exists Verification001 not in World2.exists Verification001 not in World3.exists Verification001 not in World4.exists Verification001 not in World5.exists

Link008 is Link Link008 source Verification001 Link008 target Mary Link008 present_in W6 Link008 not_present_in W1, W2, W3, W4, W5	direct_rel_in_w[(Mary),(Verification001),World6] not direct_rel_in_w[(Mary),(Verification001),World1] not direct_rel_in_w[(Mary),(Verification001),World2] not direct_rel_in_w[(Mary),(Verification001),World3] not direct_rel_in_w[(Mary),(Verification001),World4] not direct_rel_in_w[(Mary),(Verification001),World5] direct_rel_in_w[(Change001),(Verification001),World6]
Link009 is Link Link009 source Verification001 Link009 target Change001 Link009 present_in W6 Link009 not_present_in W1, W2, W3, W4, W5	not direct_rel_in_w[(Change001),(Verification001),World1] not direct_rel_in_w[(Change001),(Verification001),World2] not direct_rel_in_w[(Change001),(Verification001),World3] not direct_rel_in_w[(Change001),(Verification001),World4] not direct_rel_in_w[(Change001),(Verification001),World5]

5.7. Generating Formal Narratives with Formal Story Specifications

To generate a Formal Narrative with the Alloy Analyzer one must specify a run command in the Alloy specification. This means the model finding capacities of the Alloy Analyzer are employed to finding World States that satisfy both the the conceptual model (defining the types of things that exist in the simulation and how they may relate) and the constraints implied by the Formal Story Specification (which elements exist, how they are related to each other, which classes they instantiate and in which World they exist). In such situation, the Alloy Analyzer may find many different arrangements of elements and World States and we call each one of these arrangements a Formal Narrative for the Formal Story Specification. The run command defines the so-called “scope” of each signature in the specification and takes the story predicate as input. The possibilities are constrained by the scope: given a larger scope, more Worlds may exist and more elements of each signature may exist. The scope must be large enough to include every necessary element for the Formal Story Specification to be satisfied, otherwise the Alloy Analyzer will not find any Formal Narrative for the given scope.

This means the scope of each Identity Provider Class must be adjusted carefully to generate a Formal Narrative exactly as expected. If the scopes are loosely constrained, the Alloy Analyzer will very often create many elements which are not specified in the

Formal Story, making the Formal Narrative hard to comprehend. To illustrate how these simulations may be complex under this unconstrained circumstances, see Fig. 19 showing the first world of a Formal Narrative generated using the Formal Story Specification we created in this Chapter.

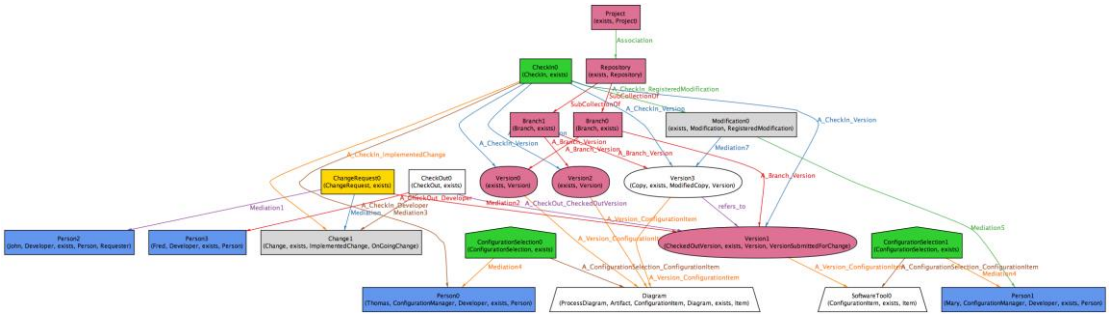

Fig. 19.Formal Narrative generated with no scope control

These may be useful only for those who are experts in using the simulation approach. In our approach, the generation of Formal Narratives must be controlled to make the Formal Narratives resemble the Natural Language Narrative as much as possible. Therefore, in the next chapter we elaborate on an explicit account to how we should elaborate these run commands, controlling the scope of the run command and creating Formal Story Specifications iteratively, to deal with the potential complexity of the simulations.

5.8. Concluding remarks

In this chapter we have introduced Formal Story Specifications as part of our validation approach. Formal Story Specifications are developed using the Formal Story Specification Language, which we have also introduced in this Chapter. They can be developed using our prototype application Story Modeler. Finally, Formal Story Specifications are transformed to Alloy predicates, which are used constrain the Alloy simulations and generate Formal Narratives.

The Formal Story Specification created in this chapter has little value to the validation approach, as the Formal Narratives it generates are too complex; except perhaps to experts in the simulation approach. It was used only as a didactic artifact for the reader, with the purpose of exemplifying the use of the Formal Story Specification Language. The elaboration of the Formal Story Specifications is instrumental and an approach to generating them iteratively taking account scope is detailed in the next chapter.

6. Iterative Validation using Formal Narratives

"All media as extensions of ourselves serve to provide new transforming vision and awareness."
-Marshall McLuhan

6.1. Introduction

In this chapter we build on the previous chapters to define an iterative validation procedure using Formal Story Specifications to generate Formal Narratives.

There are many reasons that justify an iterative procedure. First, from a technical point of view, our underlying technology assumes a small scope hypothesis i.e. that design problems can be detected using a small number of instances. More specifically, the Alloy Analyzer (the underlying technology of our model simulator) cannot handle a large number of instances and the time to analyze a model grows very fast with the size of the narrative.

Second, if a Formal Story Specification turns out to be inconsistent or generates Formal Narratives in an unintended way it can be hard to track the source of the problem in large Formal Story Specifications such as the one developed in the previous chapter. Our prototype currently has no explicit means of pinpointing what makes a story inconsistent. Therefore, it is desirable to start testing the model with stories as small as possible, to facilitate identifying possible sources of inconsistency.

Third, from a cognitive point of view, using several small examples that build up in complexity reduces the possibility of overwhelming the audience (including the modeler!) with too much detail. If opportunities for changing the model can be found in small examples, it will be easier to show and justify them.

The rest of the chapter is structured as follows: in section 6.2 we present the procedure to be followed to conduct the iterative validation; in section 6.3 we discuss how the scope of an Alloy analysis affects our approach; in section 6.4 we apply this procedure to validate and correct the model; in section 6.5 we generate a complete Formal Narrative for the corrected model, illustrating the Natural Language Narrative created in previous chapters; finally, in section 6.6 we offer concluding remarks.

6.2. Iterative procedure

The iterative process, in general, can be described by the following steps

(i) Elaborate a Natural Language Narrative that exercises the model elements;

(ii) Generate an Alloy Model for the model;

(iii) Gradually elaborate a Formal Story Specification for the Natural Language Narrative. Start with a Formal Story Specification that is as small as possible and add more elements with each iteration of the method;

   a. transform the Formal Story Specification to Alloy;
   b. merge the story predicate with the Alloy Model generated in step (ii);
   c. write a run command for the story predicate;
   d. generate Formal Narratives to assess the model. Optionally, create a Natural Language Narrative that illustrates this formal narrative;
   e. elaborate and document alternative hypothesis for model improvement;
   f. choose between one of those alternatives (which may involve SME[12]) and implement the change in the model;
   g. with the modified model go back to step (ii), adjust the Formal Story Specification (if necessary) and repeat the process until a satisfying quality is achieved and no more model modifications are necessary;

(iv) Go back to step (iii) and add more elements to the Formal Story Specification to increase the scope of analysis until the whole Natural Language Narrative is formalized;

If the diagrams generated by the Alloy Analyzer are hard to understand, and the formal narratives must be presented to Subject Matter Experts or documented, create a visual notation for the instance diagrams following [9] and re-draw them for better legibility.

When specifying a scope to the run command in step ‘c’, one should adjust the scope to be large enough to allow all the nodes specified in the Formal Story Specification, but as small as possible, to avoid noisy simulations. A noisy simulation is one that includes elements that do not help model assessment. These elements are considered, therefore, noise i.e. a disruption that interferes with the interpretation of information. By specifying the individual scope of each Identity Provider Class (Kind, Collective, Quan-

[12] Presenting the consequence of choosing between alternative modeling can be challenging and storytelling using diagrams for the formal narratives may help in such task.

tity, Relator or Mode), we can control the complexity of the simulation. For example, if we specify the scope to be exactly the number of elements we described in the Formal Story Specification, then the run command will have a much more predictable result than if we have a larger scope. With a larger scope, the analyzer may insert unspecified elements. While these can be helpful to identify unexpected scenarios, they often act only as noise, disrupting the assessment activity. We discuss the subject of scope for the run command in the next section.

When Formal Story Specifications are unsatisfiable, it can be for three different reasons. The first is that, since Formal Story Specifications are also models, they are also subject to model quality. In other words, they could be unsatisfiable due to a mistake the modeler made while elaborating them. In the second case, the modeler made no mistakes while elaborating the Formal Story Specification and their unsatisfiability can be useful since that may be an indication that the model cannot satisfy an intended state of affairs. The third and last case is that the model is satisfiable, but only to a larger scope.

6.3. Scope concerns for generating Formal Narratives

To generate the simulations (Formal Narratives), one must issue a run command in the Alloy model for the predicate generated using the Formal Story Specification. The run command is part of the Alloy model and it specifies the scope of the analysis that is executed by the Alloy Analyzer i.e. the scope specifies how many atoms will be generated at most for each signature. A run command is usually structured as the example below.

```
run story for 4 but exactly 1 World, 3 Object
```

The run command above specifies the maximum scope of every signature to 4, except the World signature, which will have exactly 1 atom in every simulation and the Object signature, which will have at most 3 atoms in any given simulation. Leaving some signature unspecified defaults their scope to the maximum scope. For example, in the command above, Property has scope size of 4. The extension set of any class that is stereotyped as a Moment is a subset of the Property set, therefore the scope of the Property signature defines how many Moment Individuals exist. *Mutatis mutandis* the same can be said about classes stereotyped as a Substance and the Object signature.

Defining the scope over the Object and Property signature can be too abstract and there may be need to define the scope more precisely, distributing the atoms between

classes of the model. To that effect, we could manually constraint the cardinality of the World signature's fields that represent each class extension. Such constraint may be included in the story predicate such as

```
#World.Person = 2
#World.Diagram = 1
```

Considering that Diagram and Person in the example above are Objects and are disjoint, in this case, the scope of the Object signature is divided between them. Combined, they take the whole scope of the Object signature. No other Object class disjoint from Person and Diagram could be instantiated in this example, unless the scope of the Object signature is increased. "Greater than" (>) and "smaller than" (<) operators can also be used instead of the "equals" (=) operator used above.

Large models that have many different classes will therefore require a large scope to be satisfied. For example, to instantiate every class in a model with X Moment Classes (such as Relator, Mode or Quality), one would need a scope of at least X. As the scope increases, simulations exponentially take longer to run. To deal with these performance limitations we have proposed a modification to the OntoUML2Alloy model transformation. The variation is specified in Appendix B and is called "Scope-Reducing" transformation since it is the main concern behind the variation. The Alloy model must be manually adapted to use the "Scope-Reducing" approach since no software support was developed.

The same run command above in the "Scope-Reducing" model transformation variation looks like this:

```
run story for 4 but exactly 1 World, 2 Person, 1 Diagram
```

In this variation, the scope of each Indentity Provider Class can be controlled in the run statement and the overall scope can be greatly reduced. In this case, to instantiate every class in a model with X Moment Classes (such as Relator, Mode or Quality), one would need a scope of 1, unless cardinality constraints impose otherwise (for example, an association with minimum cardinality of 2).

While scope details are not part of the Formal Story Specification they are a part of the model validation strategy. However, one should be cautious when restricting scope; for the scope definition can make the model unsatisfiable. For example, if we specify our running example to have exactly 1 Repository and exactly 0 Branches, the model will be unsatisfiable, as each repository has a mandatory relationship to at least one branch.

6.4. Applying the method to the running example

In this section we apply the method described in the previous section to the model we have been using as our running example (Fig. 16). Step (i) of our approach was performed when we elaborated the Natural Language Narrative in Chapter 4 . We generated the Alloy model in step (ii) but have omitted it.

To facilitate reading of the Formal Narrative diagrams, we have elaborated a visual notation for the diagrams depicting Formal Narratives. The instance diagram notation we have designed for this purpose helps reading the diagram [9]. The legend is presented in Fig. 20. The diagrams that represent Formal Narratives and the legend were generated manually, although their content was generated by Alloy in simulation.

We have selected different shapes and colors for some relevant classes and roles are represented as annotations between less than and greater than signs (“<” and “>”) and in italics, near the association that implies the role, except for Developer, which is always represented as a blue person icon (because here we do not include people who are not developers in the analysis).

Some visual similarities can be highlighted to facilitate reading. The red elements are all very tightly connected: Projects contain repositories, which in turn contain branches, which in turn contain versions. The most relevant red element is the version and the other elements are merely containers. Copy has the same shape as Version, but doesn’t have a fill color. That signals that they are similar, yet different. A Check Out also doesn’t have a fill color, to make a visual association with a copy, since every copy is always associated to a check out. A Check In is a plus sign, since it adds Versions to a branch. The Check Out is a minus sign to create a visual duality with the Check In symbol i.e. Check In is a plus, Check Out is a minus. A configuration selection is green, similar to the Check In to create a visual association a check-in. A configuration selection effectively adds Items to a branch, through the check-in. The purple symbols are also tightly related. A Change may be verified so a Verification is always associated to a Change. Modifications are associated to Check Ins, much like the Changes that motivated them. The orange symbols, Change Request and Request Evaluation are both activities that are done by someone who is not necessarily a developer. One person may request a change and someone else may evaluate such request, while a third (developer) implements it.

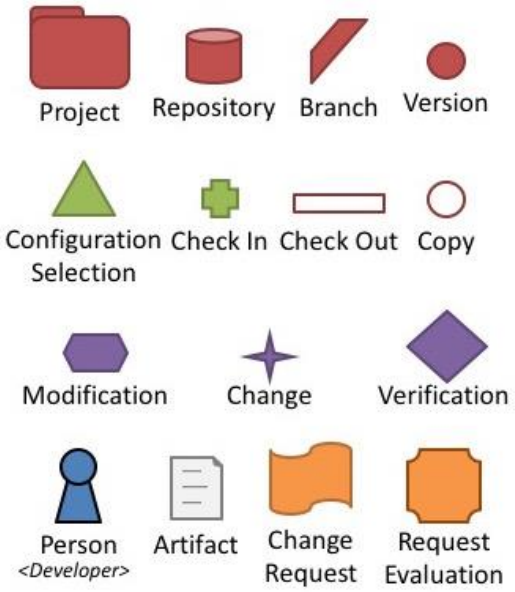

Fig. 20. Diagram legend

Starting with step (iii) of our approach, in our first iteration we specify only the existence of Thomas (a Node instance of Person, Developer and Configuration Manager) and the Diagram (a Node instance of Diagram) in an initial world (Fig. 21). Although we have not specified the existence of a Configuration in the Formal Story Specification, we expect it to appear in the Formal Narrative, as it is a logical necessity for Thomas to be a configuration manager.

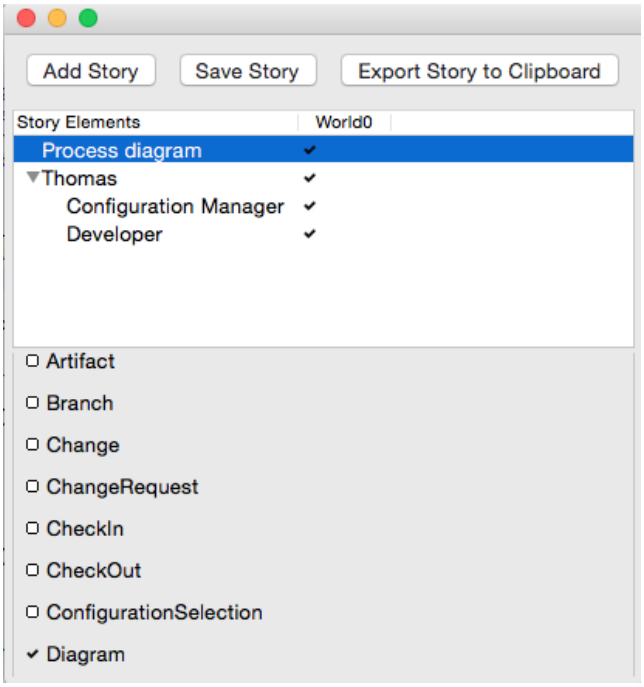

Fig. 21. First Formal Story Specification

For step (a), transforming the Formal Story Specification to Alloy, we will use the "Scope-reducing" variation of the transformation to Alloy described in Appendix B. We merge the predicate to the Alloy model (step b) and write a run command with a scope of 1. Running this very simple story (surprisingly) yields no results for a scope of 1. Increasing the scope, we find Formal Narratives that satisfy the model and they all seem to imply necessarily at least 2 branches per repository and 2 versions per branch. Inspecting these relations closely in the model we may find that even though the cardinalities are 1..* in the parthood relationships, the weak supplementation[13] axiom still applies to them, requiring a minimum cardinality of two branches and two versions per branch. This is considered inadequate from the perspective of the subject domain, which may have repositories with only one branch, as well as a single version per branch. Here we have detected an important issue with the model that could have negative consequences if used as a basis for an implementation of a configuration management system:

> **Issue 1**: it is impossible to bootstrap a simple repository with a single branch or a branch with a single version.

To deal with this problem, we enumerate alternatives: (i) configure the parameters of generating the Alloy model (in step ii of the approach) to opt-out from enforcing weak supplementation. (ii) model these relationships using a different kind of relationship such as a formal relationship or (iii) model some other part of a repository and some other part of branch, to satisfy the weak supplementation axiom.

Given the two options, we will simply opt not to enforce the weak supplementation axiom (option i). We agree with the philosophical importance of the axiom (defended in length in [49]) and this axiom helped notice what could be a problematic conceptualization. However, we will assume the model is incomplete and we believe it is safe to opt-out of it, since the ontological reasons for enforcing them do not apply to this case. We found no other parts to model that would satisfy the axiom (option iii) and the stereotype used to model the relationship seems to be the most adequate (option ii).

Having re-generated the Alloy model without the weak supplementation axiom, we restart the process (going back to step (ii)), simulating it again. However, setting the maximum scope to 1 is still not enough to generate a Formal Narrative: we still need a larger scope to satisfy the Formal Story Specification. In this scenario we may specify

[13] Briefly, weak supplementation states that a whole should have at least two parts. This does not necessarily imply that any parthood relationships should have a minimum cardinality of two: a whole could have two different parthood relationships with a minimum cardinality of 1 and still satisfy this axiom.

the scope of individual classes to narrow down where the problem is. Therefore, we may constrain the scope to show no Modification, Change, Verification, Change requests or Check Outs. However, counter intuitively, that blocks the generation of Formal Narratives. It seems that the model does not allow a simple scenario such as the one we designed, i.e. some instance of these classes (or a subset of them) are necessary for the model to be satisfiable. To find out which classes are necessary (i.e. which classes can't be set to have zero scope), we constrain the scope of each one of them individually, testing if we can still generate Formal Narratives without one of them. This reveals that the classes Modification, Change, Change Request and Check Out are all necessary for generating Formal Narratives based on our first Formal Story Specification. A close inspection of the model reveals there is a cycle of necessary relationships that requires at least one instance of each of these classes for a single Version to exist. Again, here we have detected an important issue with the model that could have negative consequences if used as a basis for an implementation of a configuration management system:

> **Issue 2**: it is impossible to create a single (unmodified) first version of an artifact.

We may solve the issue breaking this cycle by (i) changing the cardinalities of the mediation between Check In and Implemented Change from 1 to 0..1 (to allow check-ins without a Change Request) and changing the cardinality of the mediation between Check In and Registered Modification from 1..* to 0..* (to allow the first-check in scenario, where a version hasn't been modified) or (ii) creating classes to represent the first check-in and the first version; where these classes wouldn't have the relationships that imply the cycle of necessity. We model the first option in Fig. 22 and the second one is in Fig. 23.

An undesired consequence of the first option is that the model would allow check-ins (that are not the first check-in) not to be a modification of a previous version. Therefore, we would have to add constraints to the model (e.g. using OCL) to ensure that every check-in which is not the first check-in has a minimum cardinality of 1 for the mediation relations between CheckIn and ImplementedChange, and between CheckIn and RegisteredModification. The second option, on the other hand makes the model more complex, introducing vocabulary that is a priori not natural to of the domain, such as "ModificationCheckIn". In order to continue the process, we select the first option, as it is less intrusive to the model under consideration.

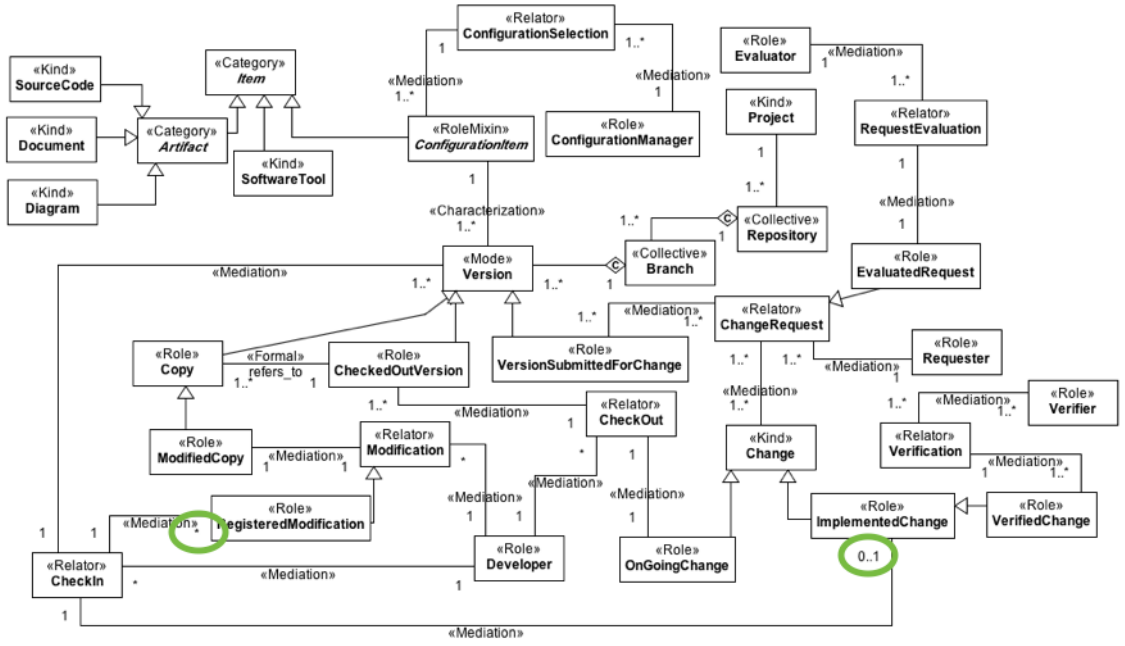

Fig. 22. Modified model with relaxed cardinalities. Changed cardinalities are circled

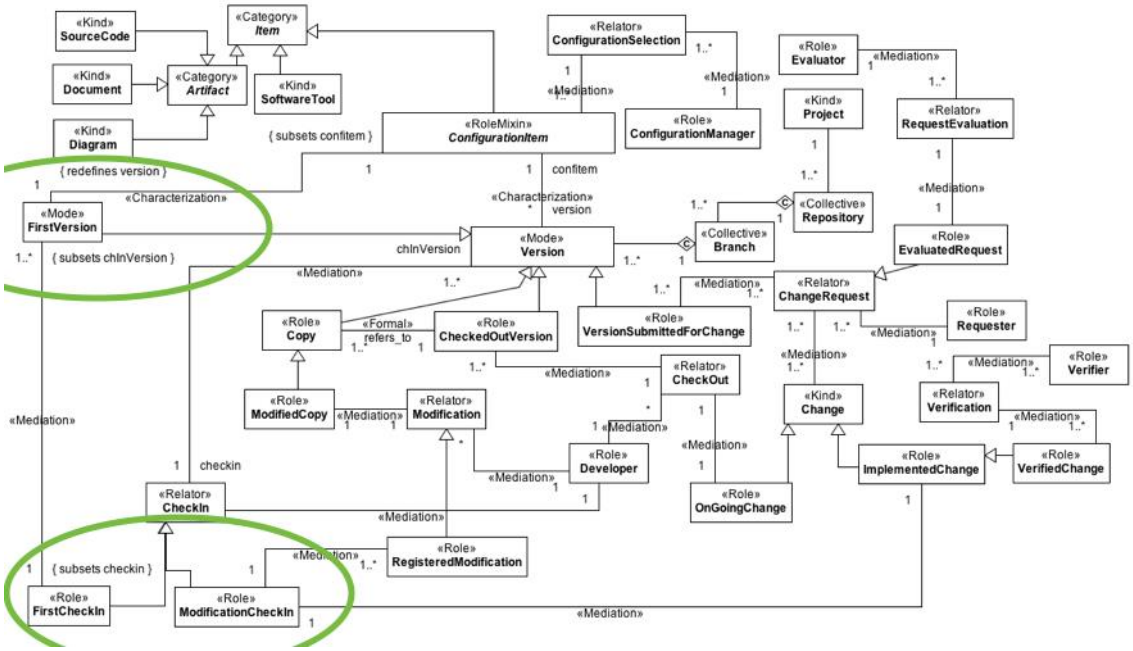

Fig. 23. Modified model including new classes (“FirstCheckIn”, “FirstVersion” and “ModificationCheckIn”)

Returning to the Formal Narrative generation, we run the simulator with a scope of 1, specifying the scope of Change and Modification to zero. This time the simulator successfully generates Formal Narratives. One of them is depicted in Fig. 24 showing a Project with a single Branch and a single Version of the Diagram, checked in and selected by Thomas, as expected. To narrate this Formal Story Specification, we could use some (micro) Natural Language Narratives such as “Thomas is a developer at Ontosoft

and commits a process diagram for the first time", "Thomas is a developer and configuration manager" or "Thomas selects a Diagram for Configuration".

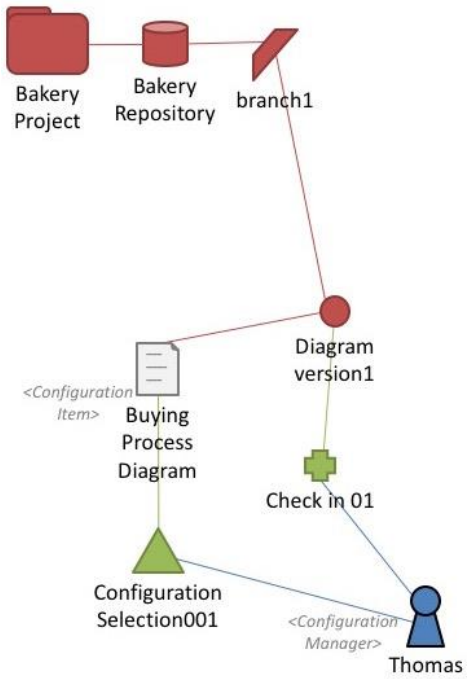

Fig. 24. First successful attempt at iterative simulation
"Thomas is a developer at Ontosoft and commits a process diagram for the first time"

Moving on with our iterative process, we will add John to the Formal Story and let him check-out the diagram in the second World (Fig. 25). A Natural Language narrative to illustrate this Formal Story Specification could be "Thomas and John are developers at Ontosoft. Thomas commits a process diagram for the first time and John checks it out to improve it."

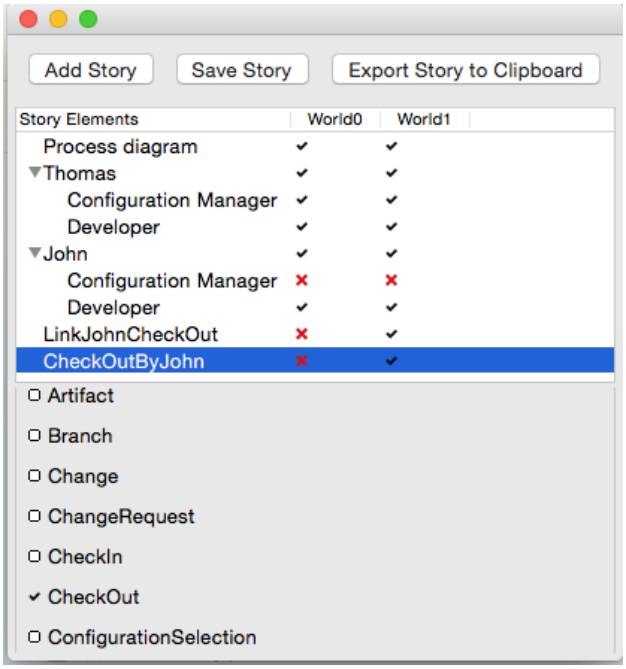

Fig. 25. Formal Story adding John and a checkout

To generate the Formal Narratives, we constrain the scope of the analysis to 2 (overall) but exactly 1 Diagram, exactly 1 Check Out, exactly 1 Check In, exactly 1 Branch and exactly 1 Repository. The first Formal Narrative we generate with this Formal Story Specification can be seen in Fig. 26 (World0) and Fig. 27 (World1). It does generate a Formal Narrative where John checks out the diagram that was previously checked in. We scrutinized this Formal Narrative for validation purposes. Here we consider some potential problems that we have identified.

The first potential problem is that in Fig. 26 (World0), we see that John checks in the diagram that someone else - Thomas - selects for configuration. As this could represent a case of under constraining, we must consult subject matter experts to determine whether this scenario is in fact admissible.

> **Question 1**: is it possible that one Developer selects an Item for Configuration and someone else checks in the selected item for the first time? In other words, should the first check in of a configuration item be done by the same person who made a configuration selection for the item?
>
> **Question 2**: is “First Check In” a concept relevant enough to include in the conceptual model?

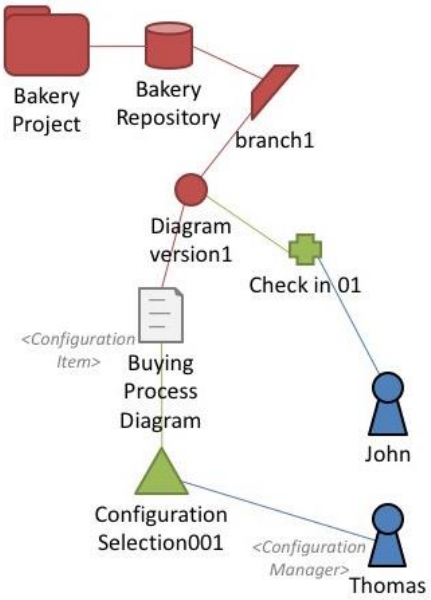

Fig. 26. World0 of the first simulation of John's checkout formal story
"John checks in the first version of the diagram selected by Thomas"

The second potential problem is that in Fig. 27 (World1), we see that John requests a change and evaluates the request himself, which seems pointless (why would you evaluate your own request?). The evaluator of a change request should probably be someone different than the person who requested the change. Additionally, we can see John both evaluated the request and checked out the diagram, while in the Narrative it was Mary who evaluated the request. Again, the Subject Matter Expert should be consulted in order to determine whether the scenario is admissible.

Question 3: Can the Requester of a Change evaluate his own Request or should it be done by someone else?

Question 4: Can the evaluator of the request be the same person who checks the version out to change it?

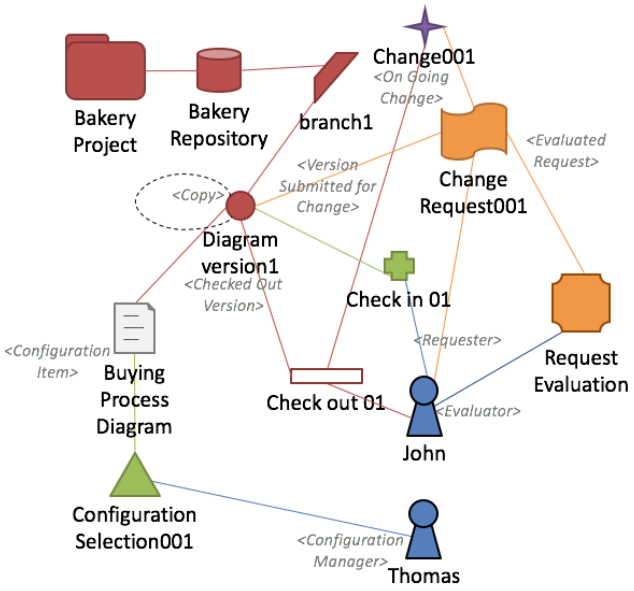

Fig. 27. World1 of the first simulation of John's checkout formal story
"John makes a Change Request for the Diagram version 1, evaluates the request and checks out the diagram"

The third potential problem is that in Fig. 27 (World1), the Diagram's version is a copy of itself (refers to itself).

Issue 3: A Version may be a copy of itself

According to the authors, a copy is created in a check-out and destroyed in a check in. Also, a version is created at checked-in, which is not the case for the copy. Therefore, it entails that a checked-out version and its copy must be different entities (as the checked out version existed prior to the copy). Therefore, we suggest correcting the model by making the Copy a separate entity, a mode of a checked out version (Fig. 28) and **not** a role of Version.

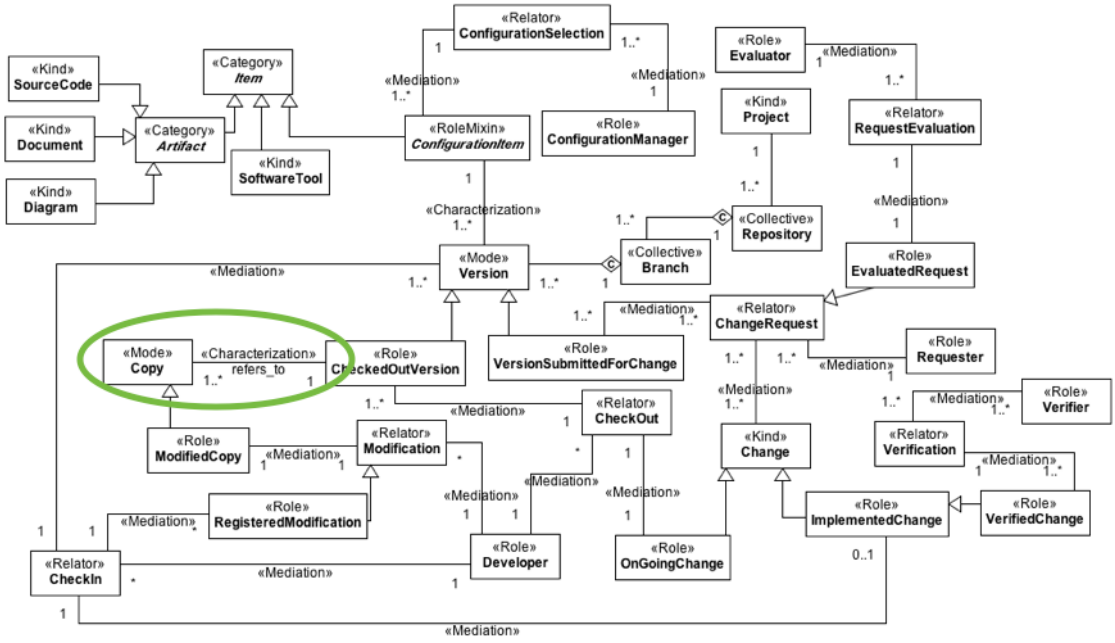

Fig. 28. Modified model with Copy as a mode of Checked Out Version

There still is a problem with this model. We will illustrate this problem using a thought experiment. First, keep in mind that a Check-In destroys a Copy. Second, notice in the diagram that there is an existentially-dependent relationship between a Check-In (which is not the "first check-in") and a Modification (Registered Modification), and also between a Modification and a Copy (Modified Copy). In our thought experiment we will imagine the moment John checks in the diagram, after implementing Fred's requested changes. Just before the check-in, there is a copy John is working on. It is a modified copy, since John made a Modification. In the moment John checks in the diagram, this modification plays the role of Registered Modification. However, if the copy John was working on is destroyed in the process of checking in the diagram, the modification associated to it must be destroyed as well by necessity (since it has a mediation relation with Modified Copy) and so would the Check In since it depends existentially on any mediated entities (in this case, it depends on the modification). Therefore, the check-in cannot exist in this setting, if the copy must be destroyed after checking in the modification.

> **Issue 4**: A Check-In cannot destroy a Copy without destroying itself as a consequence of formal necessity.

The solution lies in what is called a historical dependence relationship. The Check-In is historically dependent on the modification (and therefore on the Copy), as it was based on it. Since historical dependency relations are not covered by the simulator, we

can instead consider that Copies related to Registered Modifications are Consumed Copies (Fig. 29), and mark the class Copy as permanent [45] i.e. “its individuals, once they exist, always exist” [45]. The association ends of the mediation relation between Consumed Copy and Registered Modification are subsets of the mediation relation between Modified Copy and Modification.

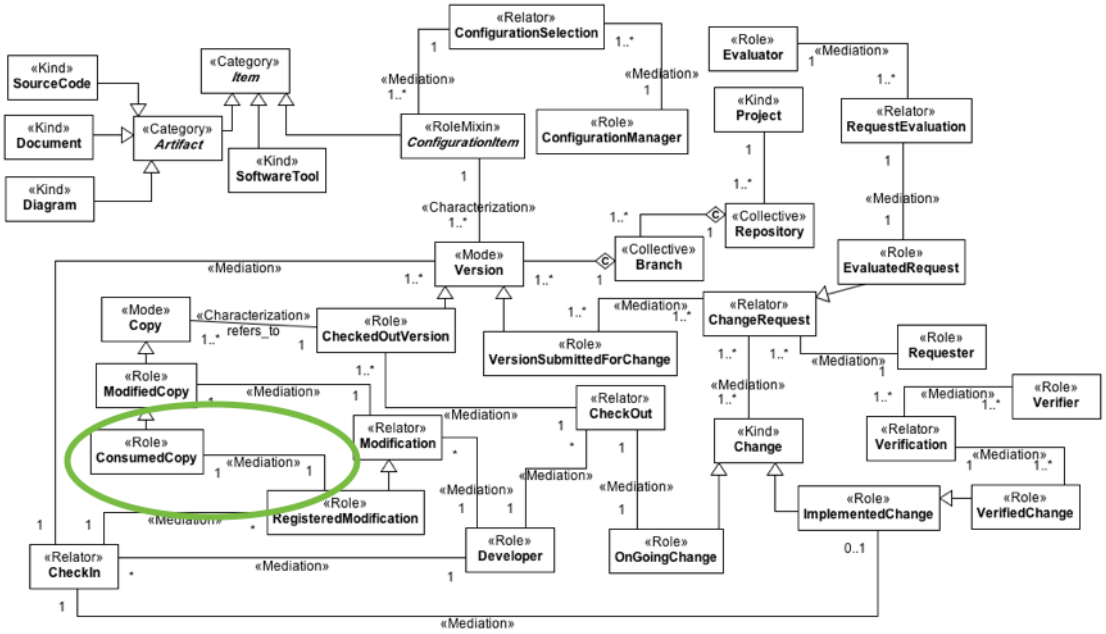

Fig. 29. Modified model including ConsumedCopy

Alternatively, the concept of copy could be removed altogether (Fig. 30). In this model, a Checked Out Version that has been modified is a Modified Version.

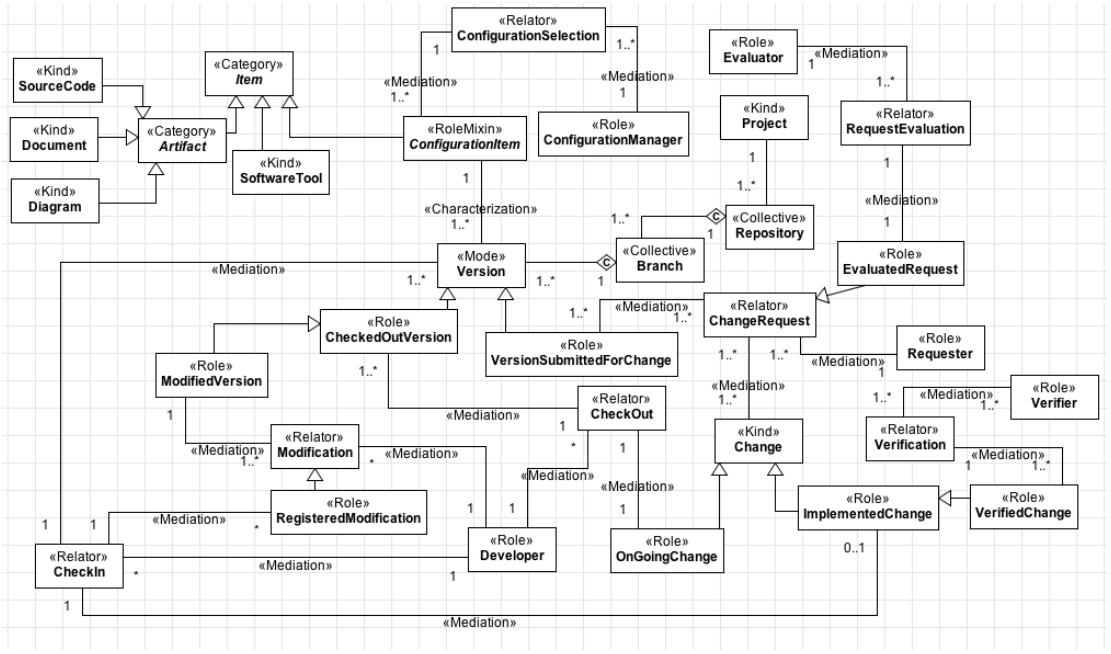

Fig. 30. Modified model removing Copy, including ModifiedVersion

These modifications suffice to generate Formal Narratives representing the whole Natural Language Narrative. These Formal Narratives are presented in the next section. Below, we summarize issues we have resolved and questions that were raised for the modelers/stakeholders of this model:

- **Issue 1**: In the original model the parthood relationship between Project and Branch and between Branch and Version makes it is impossible to bootstrap a simple repository with a single branch and a single version.
    - Adopted solution: opt out from enforcing weak supplementation, assuming the model is incomplete with respect to the parts of project and branch;
    - Alternative 1: explore alternative stereotypes for the whole-part relation;
    - Alternative 2: add different parts to Project and Branch to satisfy the weak supplementation principle.
- **Issue 2**: It is impossible to create a single (unmodified) first version of an artifact.
    - Adopted solution: Change the cardinalities of the mediation between Check In and Implemented Change from 1 to 0..1 (to allow check-ins without a Change Request) and change the cardinality of the mediation between Check In and Registered Modification from 1..* to 0..* (to allow the first-check in scenario, where a version hasn't modified a previous version);
    - Alternative: create classes to represent the first check-in and the first version; where these classes wouldn't have the relationships that form the cycle of necessity.
- **Issue 3**: A Version may be a copy of itself
    - Adopted solution: a Copy becomes a mode of Version and not a role of a version in some context (Fig. 28).
- **Issue 4**: A Check-In cannot destroy a Copy without destroying itself as a consequence of formal necessity
    - Adopted solution: refactor the model including a class to represented the Consumed Copy, making the Copy class *permanent* (Fig. 29)
    - Alternative: refactor the model and remove the Copy class (Fig. 30)

- **Question 1**: is it possible that one Developer selects an Item for Configuration and someone else checks in the selected item for the first time? In other words, should the first check in of a configuration item be done by the same person who made a configuration selection for the item?
- **Question 2**: is “First Check In” a concept relevant enough to include in the conceptual model?
- **Question 3**: Can the Requester of a Change evaluate his own Request or should it be done by someone else?
- **Question 4:** Can the evaluator of the request be the same person who checks the version out to change it?

In this section we have applied the iterative procedure to create Formal Story Specifications and Formal Narratives. We have shown how applying these steps can reveal characteristics of the model and help detect mistakes or opportunities for improvement. These can be presented to subject matter experts in the form of diagrams (Formal Narratives) which can be narrated using Natural Language Narratives, to explain the possible consequences of each modeling choice.

6.5. Revisiting Formal Narrative generation for the running example

In this section, we generate a Formal Narrative that corresponds to the complete Natural Language Narrative presented previously in Chapter 4. For this section we use the modified model presented previously in Fig. 29, product of the iterative model validation activities.

Fig. 31 is the first world of our story. In it we can see John, Thomas, Fred and Mary (in blue). Thomas is a Configuration Manager (since he is associated to Configuration-Selection001) and he selects the Buying Process Diagram (a Configuration Item, since it has a version), and checks it in to branch1 of the Bakery Repository in the Bakery Project. Notice we have not included in the Formal Story Specification the other configuration items such as the source code and the documentation. That is because they have no interaction with the rest of the story, so we can save computational resources by omitting them from the simulation. We make sure no other Items are generated by restricting

the scope of this class to 1. Other classes also have their scopes restricted, so we will only get elements that are specified in the Formal Story Specification.

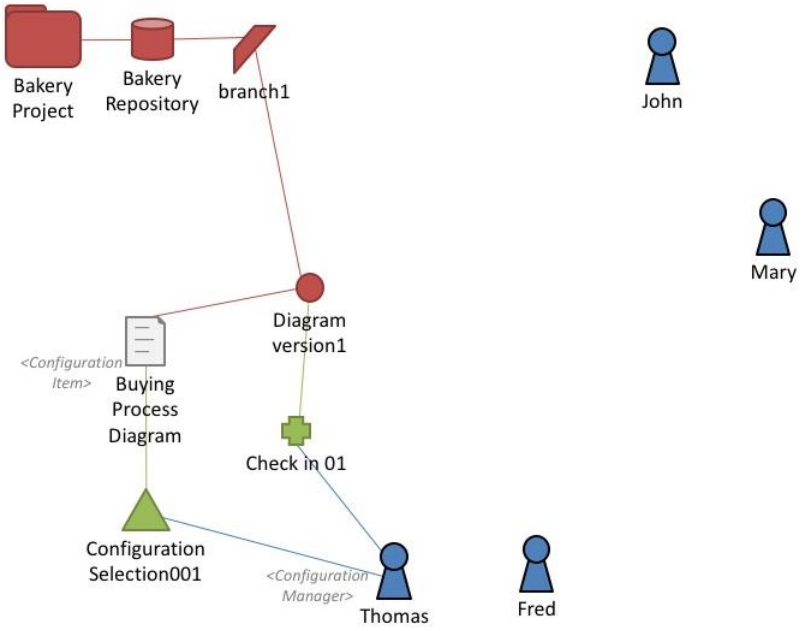

Fig. 31. World0 – Thomas selects for configuration and checks in the Buying Process Diagram

In the second world (**Fig. 32**), Fred finds a bug in the Buying Process Diagram (Version1) and files a Change Request (ChangeRequest001). Every element that was present in the previous World is concealed with some transparency to highlight the new elements. Notice there is nothing in the model (and therefore nothing in the Formal Story Specification or the Formal Narrative) about "finding bugs", which is merely a narrative device of the Natural Language Narrative. (Naturally, this is an opportunity to question whether the notion of "bugs" should be addressed formally in the domain.) The element Change001 reifies "what needs to be changed". Such Change is the subject of ChangeRequest001.

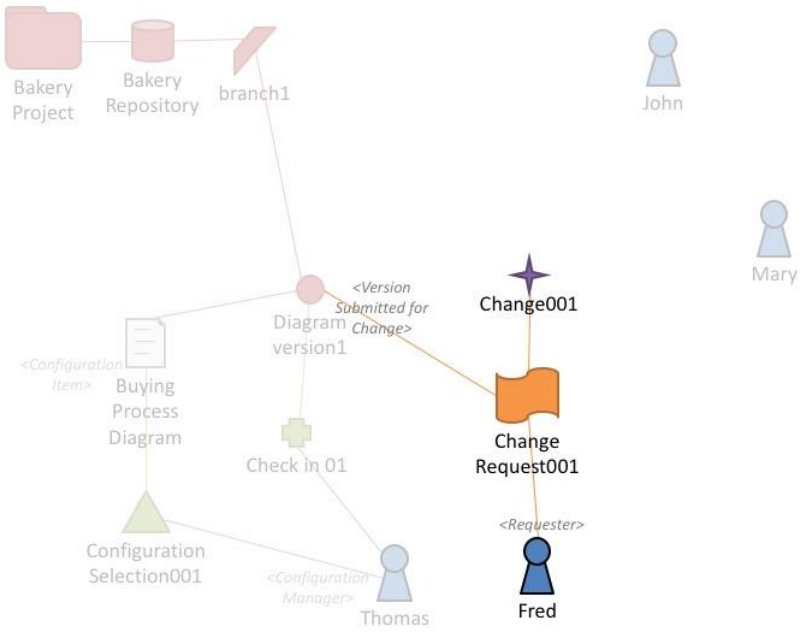

Fig. 32. World1 –Fred files a change request for DiagramVersion1

In the third world (**Fig. 33**), Mary evaluates Fred's request.

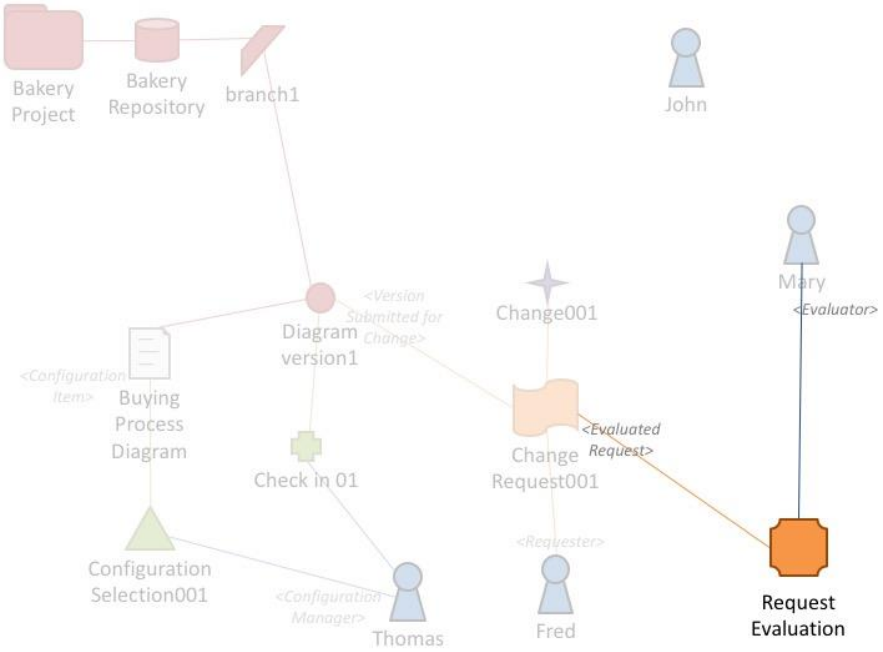

Fig. 33. World2 –Mary evaluates the request

The fourth world (**Fig. 34**) features John checking out the diagram. Since this check-out refers to Change001, the change is now an "On Going Change". Notice also that the check-out generates a Copy of DiagramVersion1, which is now a checked-out version.

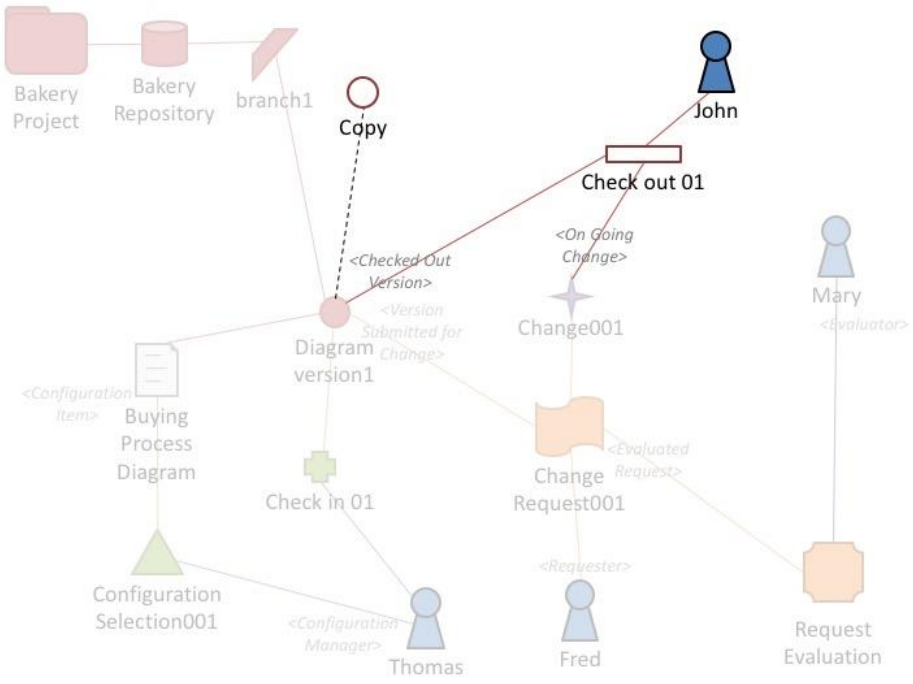

Fig. 34. World3 –John Checks out the diagram to implement the change

In **Fig. 35**, John modifies the copy to fulfill Fred's change request. That amounts to the copy now instantiating the "Modified Copy" class. Then, in Fig. 36**,** John checks in the modified copy. Mary then verifies these changes in Fig. 37.

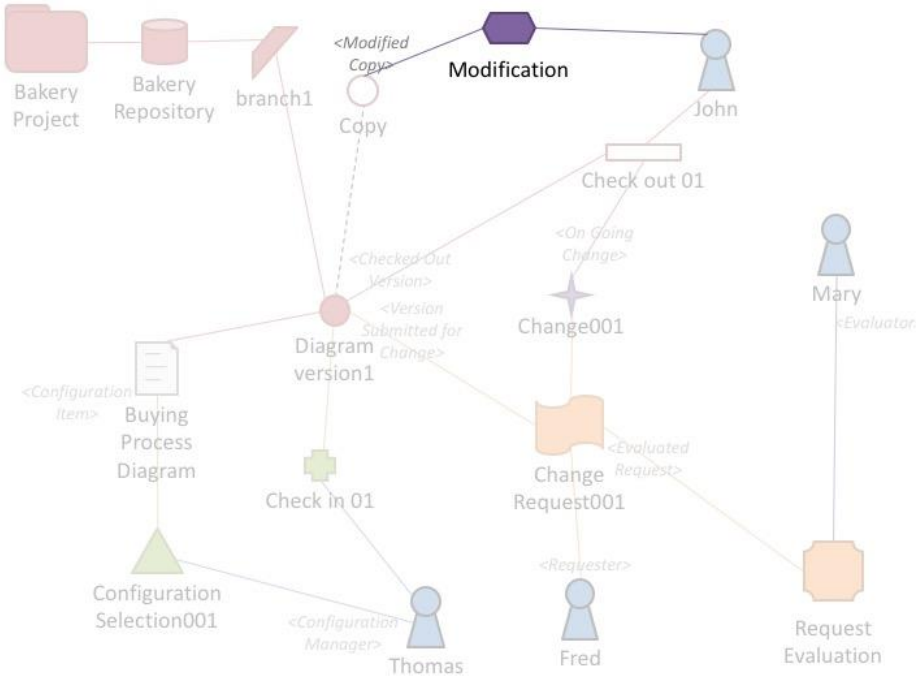

Fig. 35. World4 –John modifies the copy

Going through the diagrams from **Fig. 31** to **Fig. 37** one may see how the Natural Language Narrative from Section 4.4 is realized in the model's terms i.e. how the model

may be instantiated and realize the narrative. Taking the Natural Language Narrative, formalizing it (as a Formal Story Specification) and generating Formal Narratives allows us to test how these situations we conceptualize in reality may take place formally.

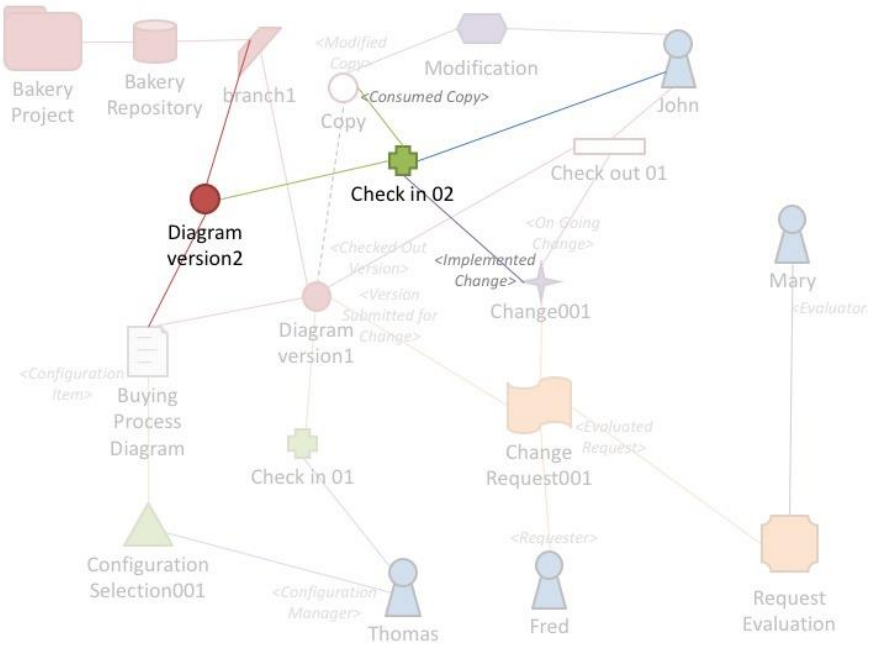

Fig. 36. World5 –John checks in the modified copy

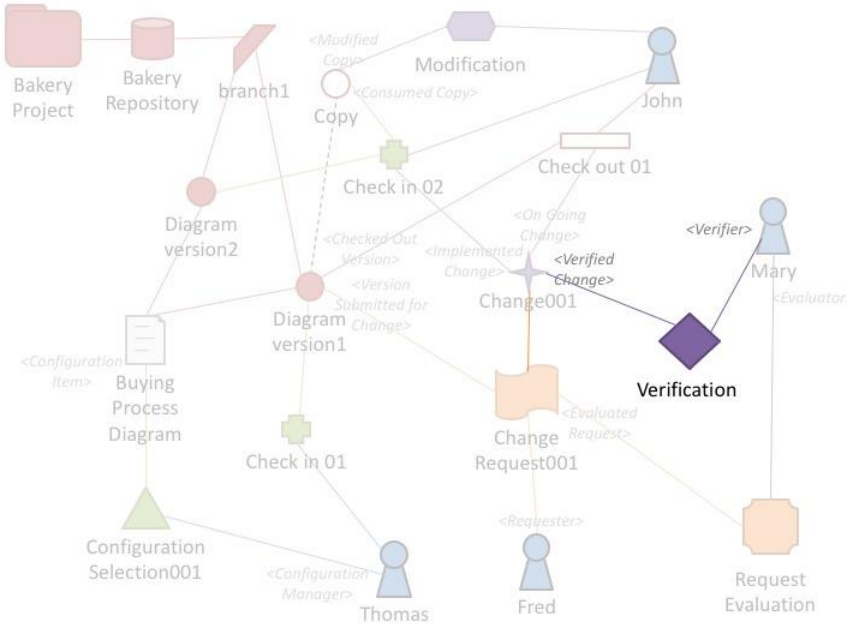

Fig. 37. World6 –Mary verifies change

6.6. Concluding remarks

In this Chapter we have shown how the elements of the approach interact using our running example. The Natural Language Narrative motivated the iterative elaboration of

Formal Story Specifications, which in turn were used to generate Formal Narratives. The Formal Narrative generation revealed problems with the model and motivated changes and questions to ask Subject Matter Experts.

Although the method can help expose errors and mistakes, it is a type of testing and as such there are no guarantees that every possible error in the model was identified.

7. Related Work

The approach presented in this work touches many different subjects and in this chapter we will focus on two different types of related work: those that also address OntoUML model assessment and those works in computer science that also involve storytelling in some way or another.

There are many different ways OntoUML models can be assessed and approaches to improve the quality of OntoUML Conceptual Models are reviewed in section 7.1. While we have explored the potential application of storytelling to model validation, there are many other types of application of storytelling in computer science and we explore some of these approaches in section 7.2.

7.1. OntoUML model assessment approaches

Previous efforts on OntoUML model assessment led to the development of tool support and some methodological guidelines. The first tool to be developed for OntoUML was called "OntoUML editor" [5]. The editor aided the construction of OntoUML conceptual models by providing a visual interactive environment to construct OntoUML conceptual models and by allowing automatic syntax verification. The OntoUML model editor was built on the Eclipse Modeling Framework (EMF), is open-source and available online via Google Code [7]. However, its development has been abandoned.

One of the main contributions of the tool was the elaboration of a meta-model for OntoUML which enabled many other tools to be built. The first few tools, OntoUML2OWL [84] and OntoUML2Alloy [10,8] used the model files built with Benevides' tool as input to generate OWL models and Alloy models, respectively.

The meta-model for OntoUML (and the syntax verification for it) was later revisited and improved by Roberto Carraretto [18], which was in turn used in OLED (Ontology Lightweight Editor) [47]. OLED offered a better modeling framework and superseded the "OntoUML Editor" by Benevides. Additionally, OLED incorporated many existing software tools for OntoUML in a single framework, including OntoUML2Alloy, OntoUML2OWL and OntoUML2SVBR [15]. OLED became the hub for OntoUML research and has since then incorporated many other tools.

To overcome some of OLED's limitations, the Menthor Editor was created as a professional alternative for OLED. It is the most advanced tool for building OntoUML models to this date and was built based on OLED's open source code.

All of the aforementioned tools provide syntax verification and while that guarantees some quality to the model, namely the adherence to the language's syntactic rules, it does not serve to increase the modeler's confidence in the correct representation of the domain [10]. In other words, it helps determining if the model was built correctly, but it does not help determining if the right model was built. This justified several efforts in model validation such as OntoUML2Alloy [10], methodologies for diagram design [9] and Identification of Semantic Anti-Patterns [69]. OntoUML2Alloy and the methodologies for diagram design have been reviewed in section 2.4.

Semantic Anti-Patterns are patterns of configuration of model elements that are error-prone. These patterns have been empirically elicited using OntoUML2Alloy. These patterns can be detected automatically using software, however, they are not necessarily errors. One must investigate if the intentions are correctly expressed in the model or not. If they are not, the software can suggest corrections.

The approach is very useful to detect errors in the model and the automatic detection is superior to our approach. However, they can only detect the patterns that have been catalogued, which is limiting. Also, the results of the analysis can be very hard to understand. In fact, some anti patterns use stories (thought experiments) to make clear the type of correction they deem necessary, which makes the two approaches complementary.

While both OntoUML2Alloy and Anti-Pattern detection can help find errors in the models, in order to constrain it to conform to the intended conceptualization it is often necessary to use a rule language. In [46] TOCL is defined, which is an extension of OCL including temporal constraints, which are important to constraining the dynamic aspects of OntoUML conceptual models. Anti-Pattern detection has the additional benefit of automatically generating some TOCL rules.

7.2. Storytelling in Computer Science

There are many applications of storytelling and narratives in computer science [80]. While we use storytelling with a focus on *a posteriori* assessment of conceptual models, the approaches we discuss in this section approach storytelling with many different pur-

poses, such as acquiring knowledge to drive modeling, [71, 65, 20], driving software development [83], assessing database systems [19], symbolic annotating textual narratives [28] and investigating the nature of narrativity[72].

One approach closely related to ours is described in [71], which is a "a group storytelling approach as an alternative to the traditional individual interviews to elicitate processes. Information gathering is proposed to be done through capturing the stories told by process performers, who describe their work, difficulties and suggestions. A process to abstract and transform stories into business process representations is also part of the method. A tool to support storytelling and this transformation is described as well." While the approach is closely related to ours in the sense that it uses storytelling to transfer information from subject matter experts to the modelers, it is concerned with the knowledge elicitation to perform quality modeling *a priori* while we are concerned mainly with *a posteriori* model validation. Concerns with the subject matter expert's lack of understanding of formal conceptual modeling languages can also be seen in this work. We could benefit from incorporating insights from this approach to ours, despite the differences between structural conceptual models and business process models. In particular, the *TellStory* tool [65], used to capture stories from multiple parties creating this collaborative storytelling seems promising, providing structure for dialogue between the parties involved, which could benefit in the authoring of Natural Language Narratives in our approach. A similar, but in this case automatic, approach of extracting the knowledge needed to build a model is presented in [20]. This approach is similar to ours as it is also concerned with Ontologies, instead of business process models. Nevertheless, the ontologies approached by [20] are OWL and RDF(s) ontologies, not well-founded domain ontologies as in our work. Additionally, as a consequence of automation the models generated in this approach have a limited expressiveness, which is a common consequence of automatically generated models.

Cucumber [83], shares some of our goals by aiming to bridge communication between subject matter experts and developers. Differently from our approach, their technique consists of elaborating short stories that exemplify systems features with the purpose of driving development. The technique shows a promising direction for future work in expanding our approach to use stories to guide model development (and not only *a posteriori* assessment).

Ciarlini and Furtado [19] use storytelling and a simulation environment for the definition of information systems. They "assume the specification of information systems in

three levels: the static level, the dynamic level and the behavioral level. The static level specifies the kinds of facts represented and the dynamic level the operations that bring about state transitions in the information systems. The behavioral level is intended to describe why events occur (i.e. the goals leading to the execution of operations) and how operations are usually combined in typical plans." They use the concept of Plot as a structure for database changes, using *Situation Calculus*. In a way similar to our approach, the formal specifications of database operations are interpreted as natural language narratives that describe it. These narratives follow structured formats such as: "Since <situation> and as <pre-conditions>, then <event>, so that <goal> and, in addition, <effects>" or "As <pre-conditions>, <operation>, so <effects>". Additionally, partial specifications can be completed using the "Interative Plot Generator" to simulate scenarios. However, their approach is heavily based on the definition of goals, libraries of typical plans and relies on an already existing database, while ours is focused based on conceptual models. This approach could be complementary to ours in the sense that we could also define some *story patterns* that can be reused in the elaboration of Formal Story Specifications. These patterns could imply some sequences of events, that structure the stories in a similar way that design patterns are used to build conceptual models.

Scheherazade is a symbolic annotation tool [28] tool for encoding, managing, analyzing and exporting Story Intention Graph encodings. Story Intention Graphs are a set of (multilayered) discourse relations which "brings out coherence at both the local and global levels: what events happen, when, why, and to whom." The layers are Textual, Timeline and Interpretative. The textual layer contains "the utterances of the original discourse that is being modeled.". The Timeline layer "formally encode story-world happenings that have been expressed in the textual layer, such as events and statives." Finally, the Interpretative layer "represent goals, plans, beliefs, affectual impacts, and the underlying intentions of characters (agents) as interpreted by the story's receiver." Much like our approach, Scheherazade allows formal modeling of narrative discourse. However, their purpose is story analysis, providing a framework that allows the user to annotate fragments of the natural language narrative to their interpretations as narrative functions, identifying discourse relations. In a way, the work is closely related to text processing. We have no intention of being so precise in our Natural Language Narrative interpretations. However, this approach could be useful if the model includes elements of UFO-C [51], which deal with the intentionality of agents using concepts such as goals, plans, beliefs and so on. This way we could include in our approach, for example,

Natural Language Narratives that express the intentions of agents and use these narratives to validate models based on UFO-C.

A similar approach to text analysis is conducted by Schärfe in his Doctor Dissertation "Computer Aided Narrative Analysis" [72] . In this work, Schärfe conducts a throughout analysis of narrativity, including the elaboration of an ontology of the terms involved using the OIL+DAML language (an OWL predecessor). The "dissertation is motivated by a wish to understand the nature of narrativity and to investigate how computer technology may assist in enhancing this understanding." [72] The work is in philosophy and is much more concerned with Narrative theory, how different theories can be combined and how ontologies can be used to refine them and aid discussion regarding their differences and similarities. While there the study of narrativity is an objective in itself, here we aim to use it to other purposes, namely, validation of conceptual models.

8. Final Considerations

In this work we have presented an approach to incorporate storytelling in an existing model validation approach. The benefits are threefold: there is more control over the model assessment process, the communication between modelers and experts is improved and the cognitive effort of the validation activities is reduced for both parties. An earlier version of the work was published [11] and presented in a workshop.

In our approach, Natural Language Narratives are authored to work as thought experiments. Roughly speaking, these thought experiments can be *constructive* i.e. creating a context to discuss the model or *destructive* i.e. offering a context and rhetoric structure to refute the model. By showing a story that fails to be valid, we can motivate change. Otherwise, a story that satisfies the modified model and proves to be valid increases the confidence on the model.

Analyzing how the elements of the model interact in a narrative allows an intuitive understanding of the model and of the consequences of abstract definitions. For example, in Appendix E we show a model that was improved simply by creating Natural Language Narratives about it and using them in validation sessions with experts to improve the understanding of the concepts in the model and negotiate modifications.

Additionally, in our approach these Natural Language Narratives are formalized using a specification language we defined. The artifacts produced in this activity are called Formal Story Specifications and they are used as basis to control the simulation defined in previous work, improving the existing model validation approach. In detail, the Formal Story Specifications are transformed to Alloy predicates that are used to constrain the Alloy simulation. Constrained by the predicate, the Alloy simulation conforms to what was specified in the Formal Story Specification. Therefore, these simulations generate diagrams (called Formal Narratives) that formally represent the Natural Language Narratives.

The process of formalizing Natural Language Narratives or interpreting Formal Narratives in terms of a Natural Language Narrative adds detail to the interpretation of the theoretical logical constructs. As "(…) thought experiments are communicated (…) frequently with diagrams." [13], our approach to generate Formal Narratives is justifiable.

The communication between modelers and subject matter experts is improved because using this approach creates a medium for them to communicate. Experts can understand the Natural Language Narrative regardless of their ineptness in the modeling

language. Using the Formal Narratives, modelers may show how the domain abstractions communicated in the Natural Language Narratives are formalized using the elements of the conceptual model. This reveals how the conceptualization was captured in the conceptual model.

Another way that this approach may improve conceptual model validation activities is by providing means for the people involved to handle large amounts of information in their minds. To analyze the model by itself one must unfold in their own mind the possibilities and interactions between classes. The mental workload of performing this analysis is offloaded to the Alloy Analyzer, shifting the focus of the people involved to the validation task. According to Cognitive Load Theory, reducing the demands on working memory improves speed and accuracy of understanding and facilitates cognitive processing of information [74]. Narratives act as a structure for the elements of the simulation, creating context which can help extend the reach of memory.

We have used a single model as an example to demonstrate the assessment technique in the previous chapters. However, in Appendix C, Appendix D and Appendix E we exemplify the application of the method to other models.

There are, naturally, some limitations to the approach. Creating Natural Language Narratives can be a creative effort which depends on the experience of the author. While we have discussed that existing thought experiments and story patterns may be used as inspiration, we have offered no prescriptive way to use them to generate these narratives.

Formal Story Specifications and their transformation to Alloy do not cover properties, which can be limiting in some domains. In Appendix C we have assessed a model which includes some important aspects of the conceptualization as properties of classes, which we could not validate due to this limitation.

Additionally, the diagrams generated by the Alloy Analyzer are hard to understand [9] and creating cognitively efficient diagrams adds significant manual work to the activity.

Finally, the power of the approach may be dependent on certain characteristics of the subject domain. “[T]here exists such a thing as unnarratable knowledge. In other words: certain constructs of the thought defy narrativization.” [72]. For example, a model about the theory of evolution by natural selection could be difficult to narrate “because there is no agent involved” [72]. “One faces, then, the difficulty of constructing an explanatory narrative that shows agency but that has to make do with an apparent

lack of entities and even an apparent lack of events, without which, of course, there can be no narrative" [1 apud 72]. There are no clear requisites that we have identified to determine if the subject is fit for our approach or not, but there is some evidence that having people or elements that can be anthropomorphized could be essential, as the intentionality of agents is important for narrativity [72]. Examples include the The Conceptual Schema of Human Genome [30] and the very successful Electrocardiogram (ECG) model [33, 34, 35, 36, 37, 38, 39, 40, 41, 84, 87].

Both models have dense conceptualizations which are hard to understand for those who are not experts in the subject domain. While it is possible to create instance diagrams to show the dynamics of the model, it is hard to see how an interesting story around these concepts could be created.

These models are a good example of what is discussed in Chapter 2 in Fig. 13; the knowledge of the domain in this case is inaccessible to the expert in the modeling language who is conducting validation. It seems it would be a good case for developing stories with the aid of domain experts.

8.1. Future Work

There are a number of additions that could improve this work and we list a few of them below to pave the direction for those who wish to pursue this line of work. These include conducting empirical evaluation of the approach (section 8.1.1), some opportunities for expanding the scope of the work (sections 8.1.2 to 8.1.5) and some software that could be implemented (section 8.1.6).

8.1.1. Empiric evaluation of the approach

While we have applied the approach on a number of models and performed qualitative evaluations, a first step in evaluating the approach would be to obtain feedback with the original modelers about the validation that were performed on the models.

A second step would be to systematically evaluate the approach and specify quality criteria that could be quantitatively measured. There are some challenges in defining an experiment to evaluate the approach quantitatively, such as which method to use to evaluate it individually or comparatively. A particular challenge is the lack of comparable approaches and controlled environments for experimentation.

We have considered using NASA's Task Load Index, which "is a subjective workload assessment tool. NASA-TLX allows users to perform subjective workload assessments on operator(s) working with various human-machine systems. NASA-TLX is a multi-dimensional rating procedure that derives an overall workload score based on a weighted average of ratings on six subscales. These subscales include Mental Demands, Physical Demands, Temporal Demands, Own Performance, Effort and Frustration. It can be used to assess workload in various human-machine environments such as aircraft cockpits, command, control, and communication (C3) workstations; supervisory and process control environments; simulations and laboratory tests." [64]

Since this test allows a subjective assessment of the cognitive load to conduct the validation activities we can effectively measure it and make predictions based on already known effects of working with high cognitive load. Further investigation is required to establish what kinds of evidence can be collected from such evaluation effort.

8.1.2. Coverage of UFO-B and UFO-C

With regard to the foundational ontology, we have covered UFO-A in this work and none of UFO-B (events) and UFO-C (intentionality). Although stories can not be defined merely as "sequences of events", it is agreed that stories necessarily involve events and transformation, which is the subject covered by UFO-B. While it could be said that events are implicitly represented in our current approach, there is no way of making explicit reference to them. Something similar can be said about UFO-C, as agency is a significantly important aspect of narrativity.

Expanding the approach to cover UFO-B and UFO-C could expand the applicability of the approach to a wider range of models. One experiment has been conducted using a model that includes elements of UFO-B and UFO-C and is reported in Appendix E. However, the Formal Story Specification Language needs to be improved to include the elements of these foundational ontologies and the Alloy model transformation approach would have to be expanded as well.

8.1.3. Thought Experiments in Conceptual Modeling

Here we have used thought experiments as means to validate a single model but elsewhere they have been used to justify design decisions in the Unified Foundational Ontology [49] and exemplify the need for some method [10,46,52]. The use of these

thought experiments is often focused on the meta level, which strikes as a significant difference from other fields of science, while showing some similarity to thought experiments in Philosophy. Additional work exploring the existing thought experiments and their potential applications in Conceptual Modeling is required.

8.1.4. Story Patterns

It seems promising that we could create design patterns for Formal Story Specification patterns based on story patterns that are discussed in Chapter 3. We have shown how works in Narratology have found that many stories can be seen as composed of elementary patterns and, if some of these patterns can act as templates for Formal Story Specifications, we could potentially draw the same benefits that software engineering finds in design patterns. Further investigation is required to identify which patterns are useful to model validation.

8.1.5. Applying the approach to systematic model testing

Drawing inspiration from software testing, we believe there is potential for exploring the use Formal Story Specifications and Alloy simulation in systematic model testing. During model development, modifications can have unexpected consequences and creating a framework for automatic testing could improve reliability in model development, especially for long modeling projects. To this end, Formal Story Specifications could be elaborated as unit tests for classes (to ensure their satisfiability), as integration tests between model fragments and between classes (showing the satisfiability of relationships) and as acceptance tests to show the implementation of use cases.

8.1.6. Additional software support

In this section we discuss some opportunities for software development which could improve the approach by automating processes which are otherwise manual.

First, while we have defined in Appendix B improvements to the model transformation to Alloy, they are currently manually performed and developing software support to automate the transformation would be a great addition to this work. The transformation is justified since there are limitations to the current OntoUML2Alloy model transformation with respect to the scalability of the analysis, given that the approach

based on the Alloy Analyzer becomes intractable when the size of the model grows. We have significantly mitigated this limitation by offering a new "Scope Reducing" model transformation to Alloy (see Appendix B) that greatly reduces the complexity of the analysis.

Second, controlling the scope is an important part of our model validation approach but currently, the scope is specified manually by modifying the Alloy model. While that does not impose a severe restriction to the approach, it may better to include it as part of the Story Specification definition in the Story Modeler.

Third, there is no software support for specifying the *same_as* and *different_from* relations in the Story Modeler. Specifying these relations increases the expressivity of the approach, allowing the definition of Formal Story Specifications that are more flexible. For example, the same Individual could be shown to satisfy the constraints of two different nodes at the same time. Currently, every element is transformed as being *different_from* each other by default.

Fourth, there are some difficulties using the approach when the model changes. This stems from references to OntoUML classes (using the *instance_of, not_instance_of* or *nonRigidClass* relations of our Formal Story Specification Language). These are references to a specific model in a specific file. If a model is modified for some reason (i.e. resulting from the validation activity), the Formal Story Specifications that had been previously used in such model could be referencing classes that no longer exist. Additionally, if the model is saved as a different file (e.g. to try two different modeling choices) the same problem would occur for every class (since the classes are identified relative to the file they are in). This could be corrected by including some sort of identifier for the classes so they could be tracked independently of the file they are in or by implementing some mechanism of searching for elements with the same name (e.g. if the class "Person" can't be found and there is another class named "Person" in the model, they are probably the same class).

Fifth, currently there is no way to "save" a specific Formal Narrative. To illustrate how this could be useful, imagine a scenario where a modeler is validating a genealogy model and he defines a Formal Story Specification where John is the father of Mary. When he generates the Formal Narrative, he is confronted with a case where not only John is the father of Mary, but Mary is also the mother of John. The modeler then decides that this should not be allowed in the model (Mary cannot be the mother of her father!) and whishes to constrain the model and revisit the Formal Story to make sure it

is not possible any more. However, his Formal Story Specification defines many possible Formal Narratives, and finding this specific example again is not guaranteed. In order to find this specific example, the modeler could generate another Formal Story Specification which is similar to the first one, but specifies additionally that Mary is the mother of John. To facilitate this kind of activity and offer means to "save" one specific Formal Narrative that was generated, a reverse transformation from Formal Narratives to Formal Story Specifications could be developed. One way to achieve this is to use the programming API offered by The Alloy Analyzer. The API can be invoked to produce XML files that represent a specific Formal Narrative. To interpret the generated XML files, one would need to rely on some sort of mapping between the signatures and the OntoUML Classes. This mapping could be generated alongside the Alloy transformations. Documenting specific Formal Narratives could prove to be useful in model testing, which has been discussed in the previous section.

Sixth, in chapter 6, we have discussed how our approach has no way of pinpointing what makes a Formal Story Specification unsatisfiable, i.e. what logical inconsistency prevents the simulator from generating Formal Narratives. However, the Alloy Analyzer can be configured to find the "unsatisfiable core" [57], which is a feature that can highlight a set for Alloy formulas for which no satisfying instance exists. This could help the user find the inconsistency but it is heavily reliable on knowledge of the transformation strategy and of the Alloy Language. Often, the feature is unusable due to the complexity of the result. We could explore further means of levering this technology in ways that can help find the inconsistencies in a Formal Story Specification.

Finally, the diagrams generated by the Alloy Analyzer are hard to read [9] and while the Alloy instance visualizer does provide customization of elements using different shapes and colors, further work is required to incorporate visualization techniques described in [9] to generate better diagrams. Here, we suggest the manual design of instance diagrams to improve diagram legibility. However, this process can be laborious and efforts into building software that can support this activity would greatly improve the approach, making it more accessible.

9. References

1. Abbot, H. Porter. Unnarratable Knowledge: The Difficulty of Understanding Evolution by Natural Selection. In Narrative Theory and the Cognitive Sciences, ed. David Herman, 143-162: CSLI Publications (2003).
2. Andoni, A., Daniliuc, D., Khurshid, S., & Marinov, D.. Evaluating the "small scope hypothesis". Technical Report MIT-LCS-TR-921, MIT CSAIL. (2003).
3. Aristotle. Poetics. translated by SH Butcher. In: Aristotle's Theory of Poetry and Fine Art. Macmillan. (1895).
4. Basili, V. R.; Boehm, B. Software Defect Reduction Top 10 List. Computer, v. 34, n. 1, p. 135-137. (2001).
5. Benevides, A. B., Guizzardi, G. A model-based tool for conceptual modeling and domain ontology engineering in OntoUML. In Enterprise Information Systems (pp. 528-538). Springer Berlin Heidelberg. (2009).
6. Benevides, A. B. A model-based graphical editor for supporting the creation, verification and validation of OntoUML conceptual models Doctoral dissertation, Ph. D. thesis, Federal University of Espırito Santo (UFES), Vitória, ES, Brazil (2010).
7. Benevides, A. ontouml - An editor for ontologically well-founded conceptual modeling - Google Project Hosting. Retrieved June 16, 2011, from http://code.google.com/p/ontouml/. (2011).
8. Benevides, A.B., Guizzardi, G., Braga, B.F.B., Almeida, J.P.A.: Validating modal aspects of OntoUML conceptual models using automatically generated visual world structures. Journal of Universal Computer Science, 16, 2904–2933. (2011).
9. Braga, B.F.B.: Cognitive effective instance diagram design. Graduation Thesis, Federal University of Espírito Santo. (2011).
10. Braga, B.F.B., Almeida, J.P.A., Guizzardi, G., Benevides, A.B.: Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal method. Innovations in Systems and Software Engineering, v. 6. (2010).
11. Braga, B. F., & Almeida, J. P. A.: Modeling Stories for Conceptual Model Assessment. In Advances in Conceptual Modeling (pp. 293-303). Springer International Publishing. (2015).
12. Brown, J. R. Thought experiments since the scientific revolution. International Studies in the Philosophy of Science, 1(1), 1-15. (1986).
13. Brown, James Robert; Fehige, Yiftach: Thought Experiments. The Stanford Encyclopedia of Philosophy (Fall 2014 Edition), Edward N. Zalta (ed.), Accessed on 26/02/2016 <http://plato.stanford.edu/archives/fall2014/entries/thought-experiment/>. (2014)
14. Calhau, R.F.: Uma Abordagem Baseada em Ontologias para a Integração Semântica de Sistemas, Master Thesis, Federal University of Espírito Santo. (2011).
15. Campbell, J. The hero with a thousand faces. (1949).
16. Campbell, J. The Masks of God: Primitive Mythology. London, Secker & Warbug. (1960).
17. Campbell, J., & Moyers, B.: The power of myth. Anchor. (2011).
18. Carraretto, R.: A modeling infrastructure for OntoUML. Graduation Thesis, Federal University of Espírito Santo. (2010).
19. Ciarlini, A.E.M., Furtado, A.L.: Understanding and Simulating Narratives in the Context of Information Systems. 21st International Conference on Conceptual Modeling (ER), Proceedings. v. 2503, 291–306. Springer. (2002).
20. Confort, V. T., Revoredo, K., Baião, F. A., & Santoro, F. M: Learning Ontology from Text: A Storytelling Exploratory Case Study. In Knowledge Management in Organizations (pp. 477-491). Springer International Publishing. (2015).
21. Dawkins, R.: The magic of reality: How we know what's really true. (2011).
22. de Cassia C Castro, R., Gracia, A. S., & Gomes, M. J. N.: Mapping of vulnerabilities in the public cloud with the use of foundational ontology: A perspective for service IaaS. In Digital Information Management (ICDIM), 2012 Seventh International Conference on (pp. 245-252). IEEE. (2012).
23. Dennett, D. C.: Intuition pumps and other tools for thinking. WW Norton & Company. (2013).
24. Dennet, D C.: On tools to transform our thinking. Intelligence Squared. Speech presented at the Royal Geographical Society, London, UK <https://youtu.be/EJsD-3jtXz0> (2013).
25. Department of Defense. Directive 5000.59. ,2007, August 8.
26. Donald, M.: Origins of the modern mind: Three stages in the evolution of culture and cognition Cambridge, MA: Harvard University Press. (1991).
27. Dijkstra, E. W. The humble programmer. Communications of the ACM 15, 10, 859-866. DOI=http://dx.doi.org/10.1145/355604.361591. (1972).
28. Elson, D.: Scheherazade, Accessed on 20/05/2015 <http://www.cs.columbia.edu/~delson/software.shtml>
29. Felluga, D. General Introduction to Narratology. In: Introductory Guide to Critical Theory. Last updated on Jan. 31, 2011. Purdue U. Accessed on 20/05/2015. <http://www.purdue.edu/guidetotheory/narratology/modules/introduction.html>.
30. Ferrandis, A. M. M., López, O. P., & Guizzardi, G. Applying the principles of an ontology-based approach to a conceptual schema of human genome. In Conceptual Modeling (pp. 471-478). Springer Berlin Heidelberg. (2013).

31.Ferreira, M. I., Moreira, J. L., Campos, M. L. M., Braga, B. F. B., Sales, T. P., Cordeiro, D. K. F., & Borges, M.: OntoEmergePlan: variability of emergency plans supported by a domain ontology. In The 12th International Conference on Information Systems for Crisis Response and Management (ISCRAM). (2015).
32.Freytag, G. Die technik des dramas. Hirzel. (1872).
33.Gonçalves, B., Guizzardi, G., & Pereira Filho, J. G. An electrocardiogram (ECG) domain ontology. In Workshop on Ontologies and Metamodels for Software and Data Engineering, 2nd, João Pessoa, Brazil (pp. 68-81). (2007).
34.Gonçalves, B., Pereira Filho, J. G., & Andreão, R. V. ECGWARE: an ECG Markup Language for Ambulatory Telemonitoring and Decision Making Support. In HEALTHINF (2) (pp. 37-43). (2008).
35.Gonçalves, B., Andreão, R. V., & Guizzardi, G. ECG data provisioning for telehomecare monitoring. In Proceedings of the 2008 ACM symposium on Applied computing (pp. 1374-1379). ACM. (2008).
36.Gonçalves, B., Zamborlini, V., & Guizzardi, G. Using a lightweight ontology of heart electrophysiology in an interactive web application. In Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web (pp. 77-80). ACM. (2008).
37.Gonçalves, B. An ontological theory of the electrocardiogram with applications. MSc, Informatics, Universidade Federal Do Espírito Santo, Vitória. (2009).
38.Gonçalves, B., Zamborlini, V., & Guizzardi, G. Uma análise ontológica do eletrocardiograma. Revista Eletrônica de Comunicação, Informação & Inovação em Saúde, 3(1). (2009).
39.Gonçalves, B., Zamborlini, V., & Guizzardi, G. An ontology-based application in heart electrophysiology: Representation, reasoning and visualization on the web. In Proceedings of the 2009 ACM symposium on Applied Computing (pp. 816-820). ACM. (2009).
40.Gonçalves, B., Zamborlini, V., & Guizzardi, G. An ontological analysis of the electrocardiogram. Electronic Journal of Communication, Information & Innovation in Health, 3(1). (2009).
41.Gonçalves, B., Guizzardi, G., & Pereira Filho, J. G. Using an ECG reference ontology for semantic interoperability of ECG data. Journal of Biomedical Informatics, 44(1), 126-136. (2011).
42.Guarino, N. Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy. Vol. 46. IOS press. (1998).
43.Guarino, N., & Welty, C. Ontological analysis of taxonomic relationships. In Conceptual Modeling—ER 2000 (pp. 210-224). Springer Berlin Heidelberg. (2000).
44.Guarino, N., & Guizzardi, G: "We Need to Discuss the Relationship": Revisiting Relationships as Modeling Constructs. In Advanced Information Systems Engineering (pp. 279-294). Springer International Publishing. (2015).
45.Guerson, J., Almeida, J. P. A, Guizzardi, G. Support for Domain Constraints in the Validation of Ontologically Well-Founded Conceptual Models. Lecture Notes in Business Information Processing. 15ed.: Springer Berlin Heidelberg, v. 175, p. 302-316. (2014).
46.Guerson J., Almeida, J.P.A.: Representing Dynamic Invariants in Ontologically Well-Founded Conceptual Models, 20th EMMSAD, Sweden (2015).
47.Guerson, J., Sales, T. P., Guizzardi, G., & Almeida, J. P. A. OntoUML Lightweight Editor: A Model-Based Environment to Build, Evaluate and Implement Reference Ontologies. In Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International (pp. 144-147). IEEE. (2015).
48.Guizzardi, G., Wagner, G., Guarino, N., & van Sinderen, M. An ontologically well-founded profile for UML conceptual models. In Advanced Information Systems Engineering (pp. 112-126). Springer Berlin Heidelberg. (2004).
49.Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Telematica Instituut, The Netherlands. (2005).
50.Guizzardi, G. Agent roles, qua individuals and the counting problem. In Software Engineering for Multi-Agent Systems IV (pp. 143-160). Springer Berlin Heidelberg. (2006).
51.Guizzardi, G., de Almeida Falbo, R., & Guizzardi, R. S. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In CIbSE (pp. 127-140). (2008).
52.Guizzardi, G. Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In Conceptual Modeling (pp. 13-27). Springer International Publishing. (2014)
53.Hoare, C. A. R: Communicating sequential processes. Prentice Hall International (1985)
54.Jackson, D. Software Abstractions-Logic, Language, and Analysis. The MIT Press. (2012).
55.Jackson, D. Alloy Frequently Asked Questions. Accessed on 06/01//2016. <http://alloy.mit.edu/alloy/faq.html> (2012).
56.Jackson, D.: About Alloy. Accessed on 06/01/2016. <http://alloy.mit.edu/alloy/> (2012).
57.Jackson, D.: unsatisfiable core. Accessed on 06/01/2016. <http://alloy.mit.edu/alloy/documentation/quickguide/unsat.html> (2012).
58.Kastelic, P. H. F.: Era uma vez uma marca – Storytelling e ficção na construção identitária da Diletto. Graduation Thesis, Escola Superior de Propaganda e Marketing (2013).
59.Lord, A. B. The Singer of Tales. Cambridge, Mass. (1960).

60.Moody, D. L. The “physics” of notations: a scientific approach to designing visual notations in software engineering. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. ICSE ’10. p.485–486. New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1810295.1810442 (2010).
61.Murray, J. H. Hamlet on the holodeck: The future of narrative in cyberspace. Simon and Schuster. (1997).
62.Mylopoulos, J.: Conceptual Modeling, Databases, and CASE: An Integrated View of In-formation Systems Development; Conceptual Modeling and Telos; Wiley. (1992).
63.narrate. Oxford Dictionaries. Oxford University Press. http://www.oxforddictionaries.com/us/definition/american_english/narrate (accessed January 21, 2016).
64.NASA: NASA TLX: Task Load Index <http://humansystems.arc.nasa.gov/groups/tlx/> (accessed January 21, 2016).
65.Perret, R., Borges, M. R., & Santoro, F. M: Applying group storytelling in knowledge management. In Groupware: Design, Implementation, and Use (pp. 34-41). Springer Berlin Heidelberg. (2004).
66.Petroski, H. Design paradigms: Case histories of error and judgment in engineering. Cambridge University Press. (1994).
67.Propp, V. I. Morphology of the Folktale (Vol. 9). American Folklore Society. (1958).
68.Sales, T. P. Identificação de Padrões de Erro em Modelagem Conceitual por Meio de Validação de Ontologias OntoUML Utilizando Alloy. Graduation Thesis, Federal University of Espírito Santo. (2012).
69.Sales, T. P., Barcelos, P. P. F., & Guizzardi, G. Identification of semantic anti-patterns in ontology-driven conceptual modeling via visual simulation. In 4th International Workshop on Ontology-Driven Information Systems (ODISE 2012), Graz, Austria. (2012).
70.Sales, T.P.: Ontology Validation for Managers, MSc Thesis, Federal University of Espírito Santo, UFES (2014).
71.Santoro, F. M., Borges, M. R., & Pino, J. A.: Acquiring knowledge on business processes from stakeholders’ stories. Advanced engineering informatics, 24(2), 138-148. (2010).
72.Schärfe, H. CANA: A study in Computer Aided Narrative Analysis (Doctoral dissertation, Videnbasen for Aalborg Universitet VBN, Aalborg Universitet Aalborg University, Det Humanistiske Fakultet The Faculty of Humanities, Naturlige og formelle sprog Naturlige og formelle sprog). (2004).
73.Searle, J.: Minds, Brains and Programs, Behavioral and Brain Sciences, 3: 417–57 (1980)
74.Sweller, J. Van Merrienboer, J.; Paas, F. Cognitive Architecture and Instructional Design. Educational Psychology Review, v. 10, p. 251-296. (1998).
75.Tuffield, M. M., Shadbolt, N. R., & Millard, D. E. Narrative as a form of knowledge transfer: Narrative theory and semantics. In In Proceedings of the First AKT DTA Colloquium. (2005).
76.Tufte, E. The magical number seven, plus or minus two: Not relevant for design. edwardtufte.com. Retrieved February 11, 2011, from http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0000U6&topic_id=1, 2000, September 23.
77. T.V. Tropes <http://tvtropes.org/> Accessed on 11/02/2016. (2016)
78.Ware, C. Information Visualization, Second Edition: Perception for Design. 2nd ed. Morgan Kaufmann. (2004).
79.Wilton, P., Tarling, J., Mc Ginnins, J.: Storyline Ontology; Accessed on 04/01/2016 <http://www.bbc.co.uk/ontologies/storyline>
80.Winer, D.: Review of Ontology Based Storytelling Devices. In Language, Culture, Computation. Computing of the Humanities, Law, and Narratives, 394–405. Springer. (2014).
81.Wikimedia Commons, Hero’s Journey. Last updated on April 05, 2012. Accessed on 04/01/2016. <https://commons.wikimedia.org/wiki/File:Heroesjourney.svg>
82.Wikipedia contributors. Alloy (specification language). Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 20 Aug. 2014. Web. 1 Feb. 2016.
83.Wynne, M., and Hellesoy, A.: The cucumber book: behaviour-driven development for testers and developers. Pragmatic Bookshelf. (2012).
84.Zamborlini, V., Gonçalves, B., & Guizzardi, G. Codification and application of a well-founded heart-ECG ontology. In Workshop on Ontologies and Metamodels for Software and Data Engineering, 3rd, Campinas, Brazil. (2008).
85.Zamborlini, V. C., and Guizzardi, G.: On the representation of temporally changing information in OWL. In: 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 283–292. IEEE. (2010).
86.Zamborlini, V. C. Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML Para OWL: Abordagens Para Representação de Informação Temporal. Federal University of Espírito Santo. Available only in Portuguese. (2011).
87.Zamborlini, V., Gonçalves, B., & Guizzardi, G. Codification and application of a well-founded heart-ECG ontology. In Workshop on Ontologies and Metamodels for Software and Data Engineering, 3rd, Campinas, Brazil. (2008).
88.Zimmerman, J.: Unit Testing. Principles of Imperative Computation. (2012).

Appendix A - Alloy

In this section we will provide a brief introduction to Alloy. Only elements necessary to discuss the transformation strategy from Formal Story Specifications to Alloy in Chapter 5 are introduced.

"Alloy is a language for describing structures and [the Alloy Analyzer is] a tool for exploring them." [56]. Technically speaking, Alloy is a declarative specification language (based on Z) "for expressing complex structural constraints and behavior in a software system" [82]. The Alloy Analyzer "is a solver that takes the constraints of a model and finds structures that satisfy them" [56]. It is also an IDE for creating the Alloy specifications. In the context of the OntoUML2Alloy approach, we refer to the act of running of the Alloy Analyzer (i.e. finding structures that satisfy an Alloy specification) as a model simulation.

A model in Alloy consists of logical constraints which are captured in signature, fact, predicate, assertion and function declarations. *Signatures* define the vocabulary of a model by introducing *top-level sets*. A top-level set is a special type of set which introduces in the simulation elements called Atoms. Any atom is necessarily member of one and only one top-level set. When a model is instantiated by the Alloy Analyzer, atoms are generated from signatures respecting the logical constraints in the model. In other words, a signature at the model level introduces a set of atoms at the instance level. Additionally, relationships are defined as *Fields* of a signature; which defines sets of atom tuples.

In this section we will use as a running example[14] a model that describes an Address Book consisting of names and addresses. Below we define three signatures: Name, Address and AddressBook. In the AddressBook signature, a field entry defines a mapping between names and addresses. In fact, entry is a three-way mapping associating AddressBooks, Names and Addresses.

```
sig Name{}
sig Address{}
sig AddressBook{
   entry: Name -> lone Address
}
```

[14] This example has been adapted from [54]

The keyword lone in the declaration of entry indicates multiplicity. In this case, this indicates that each name is mapped to at most one address. The available multiplicity keywords are none, one, lone, set and some, which specify 0, 1, 0..1, 0..* and 1..* multiplicities, respectively.

To generate instances of this model, we must issue a run command using some predicate. Predicates are (possibly parameterized) constraints and can be used to represent operations[15]. For example, the predicate show below constrains that there should be at least one atom for the Name, Address and AddressBook signature. The number after the for keyword specifies the scope of the simulation. In practical terms, the scope[16] defines the maximum number of atoms a simulation will generate for each signature declaration. Using the but keyword we can specify smaller scopes for each signature individually. In the example below, the predicate specifies there should be at least one atom of each signature and on the scope constraints of the run command we specify that there can be at most three atoms of any signature, but at most one AddressBook atom.

```
pred show [] {
   some Name
   some Address
   some AddressBook
}
run show for 3 but 1 AddressBook
```

Running the predicate show could generate a large number of different structures that satisfy it. One such structure could be, for example, an address book b with an entry for john associated to address addr, which would be represented as b→john→addr. In this case, b is part of the set AddressBook, john is part of the set Name, addr is part of the set Address and the tuple b→john→addr is part of the set entry.

Alloy allows a *navigation expression style*, where sets are formed by "navigating" from quantified variables along relations. In our previous example, b.entry forms john→addr. Signatures may also be used to navigate on relations. Since there is only one address book in our example, AddressBook.entry also returns john→addr.

---

[15] Predicates can also be used in other predicates, functions, assertions and facts (which will be introduced later)

[16] Alloy relies on what is called *the small scope* hypothesis. "The "small scope hypothesis" argues that a high proportion of bugs can be found by testing the program for all test inputs within some small scope." [2] This hypothesis is reflected in the Alloy Analyzer's internal structure. "All problems are solved within a user-specified scope that bounds the size of the domains, and thus makes the problem finite (and reducable [sic] to a boolean formula)." [55] The analysis is complete within the specified scope, but as the scope increases, the performance quickly drops, therefore one should try to work with a scope as small as possible.

A larger example may make the subtleties of the navigation style clearer. Let there be two additional Name atoms frank and mary, their addresses addr2 and addr3 and a second address book b2. Let the previously defined address book b also contain Frank's address and b2 only contain Mary's address. In this case, the set entry would contain b→john→addr, b→frank→addr2 and b2→mary→addr3. The navigation AddressBook.entry would form the set john→addr, frank→addr2, mary→addr3, while b.entry would form the set john→addr, frank→addr2 and b2.entry would form the set containing only mary→addr3.

Facts introduce constraints which are assumed to always hold. For example, the fact below specifies that the set entry is not empty. It entails that Name, Address and AddressBook are also not empty, since there must be at least one atom of each signature to compose a tuple of the entry set. Names of the facts are formally irrelevant and serve only to improve model legibility.

```
fact no_empty_books{
   some entry
}
```

Functions are (possibly parameterized) expressions that return results: the body of a function is an expression, rather than a constraint and it is used for reoccurring expressions. For example, the function below returns a tuple with the given parameters. Between the brackets we define variables separated by commas. Each variable definition is composed of the variable name and type, separated by a colon. Following the brackets, after the colons, the function's return type is specified. Inside the curly brackets the body of the function is specified. In this very simple example, the function returns a tuple using the parameters.

```
fun combine[
   b:AddressBook,
   n:Name,
   a:Address] : AddressBook->Name->Address
{
   b->n->a
}
```

Functions and predicates can be reused in other functions and predicates. For example, below we define the create_entry predicate that uses the combine function. The in keyword specifies that the left hand side is a subset of the right hand side.

```
pred create_entry[
```

```
    b:AddressBook,
    n:Name,
    a:Address]
{
    combine[b,n,a] in entry
}
```

The set operators are

- + union
- & intersection
- - difference in subset
- = equality

and here is what they mean:

- a tuple is in p + q when it is in p or in q (or both);
- a tuple is in p & q when it is in p and in q;
- · a tuple is in p - q when it is in p but not in q;
- · p in q is true when every tuple of p is also a tuple of q;
- · p = q is true when p and q have the same tuples.

We can use set operators to constrain the Analyzer to generate specific structures of atoms (among other things).

To generate model instances similar to previously presented john, frank and mary example, we can create predicates and assign each element to a variable. Note that below we use the disj predicate, which is a built-in predicate that constrains variables to be disjoint. In this case every variable is a single atom and the disj predicate guarantees that they are not the same atom.

```
pred show2 [
    john: one Name,
    mary: one Name,
    frank: one Name,
    addr: one Address,
    addr2: one Address,
    addr3: one Address,
    b: one AddressBook,
    b2: one AddressBook
] {
    disj[john,mary,frank]
```

```
    disj[addr,addr2,addr3]
    disj[b,b2]
    create_entry[b,john,addr]
    create_entry[b,frank,addr2]
    create_entry[b2,mary,addr3]
}
run show2 for 3 but 2 AddressBook
```

The run command above could generate the example we are interested in, but it could also generate a large number of other possible configurations. For example, a scenario where both address books have the addresses of all the names in them is also possible according to this predicate.

We could constrain the predicate further, restricting the entry of each address book using set operators.

```
(john->addr + frank->addr2) = b.entry
mary->addr3 = b2.entry
```

This concludes the overview of the Alloy language and the Alloy Analyzer. We have introduced sufficient elements to discuss the model transformation strategy in Chapter 5.

Appendix B– Scope-reducing OntoUML2Alloy model transformation variation

In this appendix we detail a variation of the "Branching World" model transformation to allow model simulations using smaller scopes, improving performance. We call this variation the "Scope-reducing" transformation. We have not implemented the transformation in software, so we propose ways to manually adjust to the "Branching World" transformation.

As explained in Appendix A- Alloy, the scope of a run command defines the maximum number of atoms of each top-level signature. As the scope increases, the time to run the simulation increases drastically. In the "Branching World" transformation any atom representing an individual is in the Object signature or in the Property signature. By breaking down these signatures into multiple signatures, we can reduce the necessary scope size for a simulation. For example, to instantiate every class in a model with a 'X' number of Moment Classes (such as Relator, Mode or Quality) in the "Branching World" approach, one would need a scope of at least 'X'. In the "Scope-reducing" approach, to instantiate every class in a model with 'X' Moment Classes, one would need a scope of 1 (unless cardinality constraints impose otherwise).

The difference between the two transformations is mostly about creating these extra signatures and dealing with the consequences of removing the Object and Property signatures. Instead of using a single signature for every Substantial and another for every Moment, we use a different signature for each *Identity Provider Class*. An Identity Provider Class is a class that provides identity to its instances; each individual must be instance of one and only one Identity Provider Class. An Identity Provider Class is, in the case of Substantials, a Kind, Collective or Quantity class. In the case of Moments, it is a Relator, Mode or Quality class. In this interpretation, a Moment Individual is classified by one and only one Relator, Mode or Quality class. Therefore, to specify superclasses or subclasses of Moments, we assume that Categories, Phases, Roles, RoleMixins, Mixins and SubKind may be used in Moment hierarchies in the same way they are used in Substantial hierarchies.

Each identity provider signature is defined with a double underscore prefix to its name. That way, the field in the World signature with the class name remains as the ex-

tension set of the Class, maintaining compatibility with the previous approach (see example below).

Some constraints of the previous approach (e.g. rigidity and generalization constraints) refer to Object and Property. To maintain compatibility, we create "Object" and "Property" functions. Each returns the union set of every signature, respectively i.e. the Object function returns the union set of every signature that represents a Substantial and the Property function returns the union set of every signature that represents a Moment. Therefore, any constraint that would reference the Object or Property signature still works.

Since the Alloy syntax does not allow the use of functions in field definitions, they should be defined in terms of the Identity Provider signatures, instead of using Object or Property, as it is done in the previous approach. For the Identity Provider Classes, it is trivial: the field is defined using the homonymous signature (prefixed by double underscore). In the case of subclasses, they are defined in terms of their identity provider class. Even if there is a hierarchy of subclasses, defining the subclasses in terms of the identity provider signature is enough (it is not necessary to define it in terms of the immediate superclass), generalization constraints are generated such that the hierarchy is preserved. In the case of Mixins, RoleMixins and Categories, the fields should be defined as a union of the possible Identity Provider Classes.

To further improve performance, we also recommend the use of the util/ordering library (linear ordering) to order worlds, as opposed to the world_structure library provided by the "Branching World" transformation. The world_structure library includes the continuous_existence predicate which should be copied to the model, as exemplified above. Additionally, there is a generated statement that should be removed from the Alloy story predicate generated by the Story Modeler: World0 in CurrentWorld. In the util/ordering approach, there is no distinction between past, current or future worlds; this distinction is specified in the world_structure library.

The following model in Fig. 38 is transformed to Alloy using both approaches in the table below. To the left, using the "Branching World" approach and to the right using the "Scope-Reducing" approach. Notice the difference between the two approaches regarding the signature definitions and the World signature field declarations. Rules have been omitted from the model. The model on the left has some blank spaces between the lines to align each of the elements in the two models.

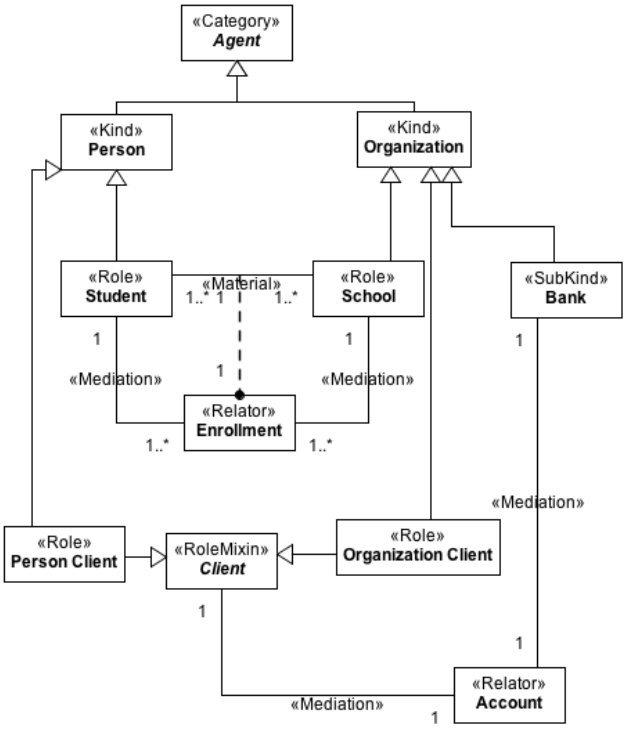

Fig. 38. Student enrollment / Bank client model

module Model	module Model
open world_structure[World]	open útil/ordering[World]
open ontological_properties[World]	open ontological_properties[World]
open util/relation	open util/relation
open util/ternary	open util/ternary
open util/boolean	open util/boolean
sig Object {}	sig __Person {}
	sig __Organization {}
	fun Object : univ{
	__Person+__Organization
	}
sig Property {}	sig __Enrollment {}
	sig __Account {}
	fun Property : univ {
	__Enrollment+__Account
	}
sig DataType {}	sig DataType {}

```
abstract sig World {
    exists: some Object+Property,

    Account: set exists:>Property,
    Agent: set exists:>Object,

    Bank: set exists:>Object,
    Client: set exists:>Object,

    Enrollment: set exists:>Property,
    Organization: set exists:>Object,
    OrganizationClient: set exists:>Object,

    Person: set exists:>Object,
    PersonClient: set exists:>Object,
    School: set exists:>Object,
    Student: set exists:>Object,
    Material1: set Student -> Enrollment -> School,
    Mediation11: set Account one -> one Client,
    Mediation1: set Enrollment some -> one Student,
    Mediation21: set Account one -> one Bank,
    Mediation2: set Enrollment some -> one School
}
```

```
abstract sig World {
    exists: some __Enrollment+ __Account+ __Person+
__Organization,
    Account: set exists:>__Account,
    Agent: set exists:>__Person+__Organization,
    Bank: set exists:>__Organization,
    Client: set exists:> __Person+ __Organization,
    Enrollment: set exists:>__Enrollment,
    Organization: set exists:>__Organization,
    OrganizationClient: set exists:>__Organization,
    Person: set exists:>__Person,
    PersonClient: set exists:>__Person,
    School: set exists:>__Organization,
    Student: set exists:>__Person,
    Material1: set Student -> Enrollment -> School,
    Mediation11: set Account one -> one Client,
    Mediation1: set Enrollment some -> one Student,
    Mediation21: set Account one -> one Bank,
    Mediation2: set Enrollment some -> one School
}
/*Objects can't die and come to life later*/
pred continuous_existence [exists: World->univ] {
all w : World, x: (@next.w).exists | (x not in w.exists) => (x
not in (( w. ^next).exists))
}
```

Appendix C– Applying method to Bank Model

a) – The model

In this section we apply the method in a Bank Model extracted from http://www.menthor.net/bank.html. The model was developed as a graduate course assignment and there is no documentation available for it, so some of the meanings of concepts (such as Inactive Account) will be deduced.

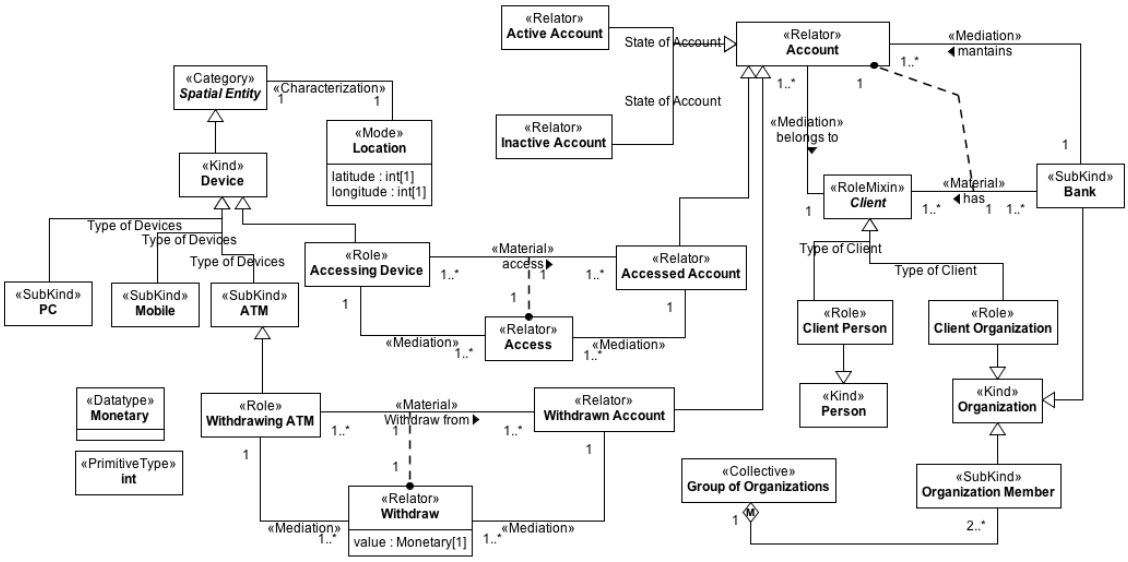

Fig. 39. Bank Model

The model specifies Banks, ATMs and two types of transactions ATMs can perform on accounts: Accessing and Withdrawing money. Accesses may also be performed on other type of devices such as PCs and Mobile Phones. Accounts can belong to a Person or to an Organization. The model also specifies Groups of Organizations and their Members, but we consider the fragment related to the structural aspect of organizations outside the scope for the exercise in this annex.

In a brief inspection of the model, a section that stands out is the “Account” hierarchy, which uses the Relator stereotype to model seemingly anti-rigid concepts, namely Active/Inactive Account, Accessed Account and Withdrawn Account. Modal properties of relators were not explored in the first versions of OntoUML and the current OntoUML2Alloy transformation considers any class stereotyped as a relator to be a rigid class. To apply our “scope-reducing” model transformation to Alloy (Appendix B) we will need to elect one of the classes in the hierarchy to be the Identity Provider Class.

Account seems to be the fair choice as they are frequently identified by a number and is the top-most type of the hierarchy, providing identity to its subclasses.

b)– Overviewing Natural Language Narrative

The following Natural Language Narrative exercises most of the concepts present in the model. Here, we instantiate the model by drawing the meaning of the concepts from our own experience with dealing with the domain of discourse and later check (using our method) if the model is fit to formalizing the Natural Language Narrative we produced or not.

> *John felt hungry after his Calculus classes and wanted to buy food from the cafeteria, but realized he had no money on him. Luckily, there was a Santander Bank branch* ***located*** *inside the university, and he was a* ***client*** *with an* ***active account*** *there.*
>
> *He pulled his mobile phone from his pocket and* ***accessed*** *his account to make sure his balance was positive. He noticed he had less money than he expected. Checking his account statement, he was shocked to find the bank had charged much more for the account maintenance than what they had been charging for the previous months. He headed to the bank, furious. Getting there, he inserted his card in the* ***ATM*** *and* ***withdrew*** *all of his remaining money: R$120,00. He then entered the Agency and, after some discussion, closed his* ***account****.*
>
> *After leaving the agency, he checked on his mobile phone to make sure his* ***account*** *was* ***inactive,*** *and it was. That's when it came to him that he had spent so much time in the agency arguing that he was already late for his next class; and he didn't even get to eat anything.*

The Natural Language Narrative above exercises the following aspects of the model

- Person being a Client (Client Person, to be more specific) of a Bank,
- Having an account there (an Active Account),
- The account is accessed from a Device (a Mobile Phone),
- A Withdraw is performed in an ATM

- The ATM is Located in the University
- The account was closed

Two elements of the model were not exercised in the Natural Language Narrative: PC (Personal Computer) and Client Organization. To exercise them, we could modify the narrative to make John check his account balance in a PC instead of his Mobile Phone, and John could instead be accessing his Company's account instead of his own account.

c) – Model Assessment

In this section, we elaborate Formal Story Specifications for fragments[17] of the Natural Language Narrative presented above and generate Formal Narratives to assess the model. For the first iteration of the method, we create a Formal Story Specification that formalizes only a small fraction of the Natural Language Narrative, namely, the fragment in which John withdraws money from the ATM.

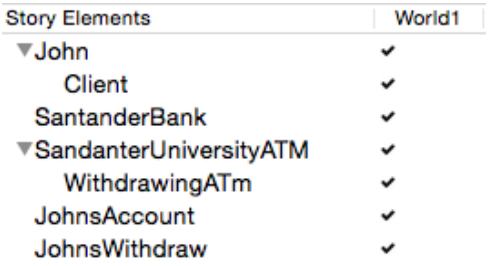

Fig. 40.First Formal Story Specification for the Bank Model

In Fig. 40, five Nodes are specified: John, SantanderBank, SantanderUniversityATM, JohnsAccount and JohnsWithdraw. Every node exists in World1. Additionally, there are two classification statements, one about John and one about the SantanderUniversityATM. John's classification statement specifies John is a Client in World1, and SantanderUniversityATM's classification statement specifies it is an WithdrawingATM in World1. With regard to the Node's classification, John is a Person, SantanderBank is a Bank, JohnsAccount is an Account and JohnsWithdrawn is a Withdraw.

The first Formal Narrative produced (Fig. 41) shows the model is satisfiable with a scope of 1. The diagram notation is as follows: each figure represents an individual. Inside each figure, in the first line, there is the name of the individual, generated automati-

[17] As described in Chapter 6, Formal Story Specifications and Formal Narratives should be created iteratively. One should start with a small Formal Story Specification and assess the model with the generated Formal Narratives and increment the Formal Story Specification in each iteration until the Natural Language Narrative is fully covered.

cally by the Alloy Analyzer. The name of the individual is the name of the signature, followed by a number, if and only if there is more than one individual of the same signature in that simulation. On the second line, between parentheses, there is a list of every set the individual belongs to. Sets that start with a dollar $ sign are the variables defined by our story predicates. These are prefixed with the name of the story predicate, followed by an underscore and finally followed by the variable name. Here we have defined that Person is represented by a blue square, Organization by a green square, Account by a red hexagon with a dashed border, Withdraw by a red inverted trapezoid, Location by a red circle, and Device by a yellow square.

One detail revealed in this generated Formal Narrative is that John's account is an InnactiveAccount. Should that be allowed? It certainly does not conform with the scenario depicted in the Natural Language Narrative which is an ATM withdraw, but if the Client wishes to withdraw the money of an Inactive Account in an Agency, it could be permitted. Nevertheless, this point could be discussed with subject matter experts to give insight on what conditions this is allowed or forbidden.

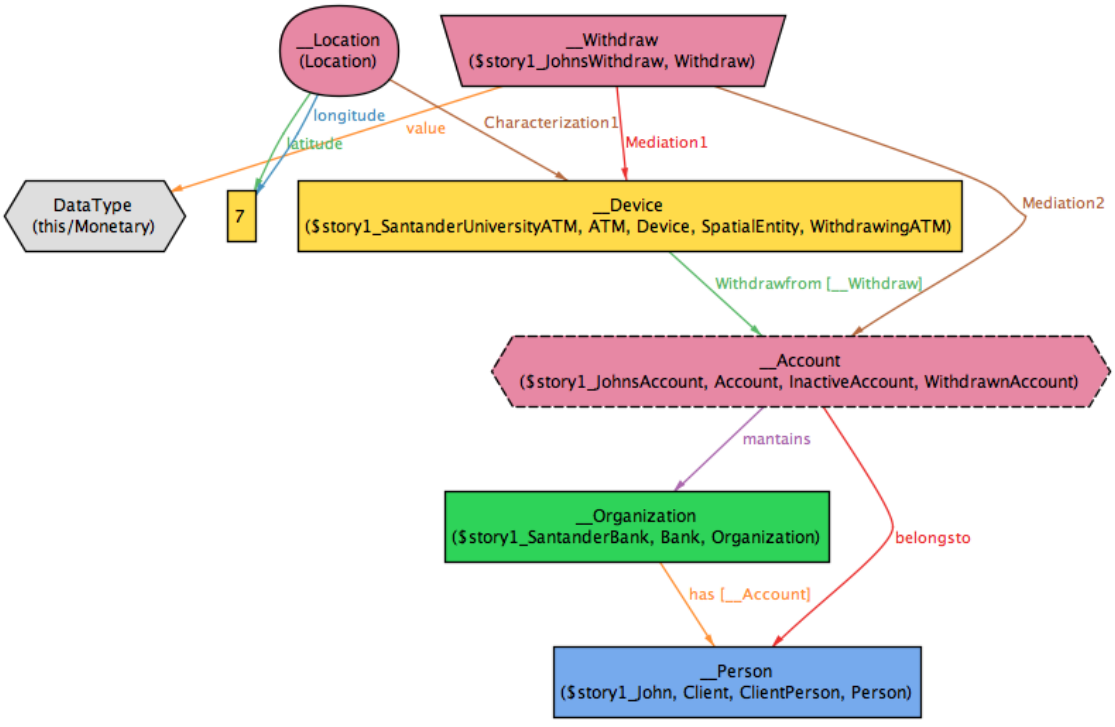

Fig. 41.First Formal Narrative for the Bank Model

Proceeding with the expansion of the Formal Story Specification, we include the part of the narrative where John checks the account on his mobile phone prior to going to the ATM to withdraw the money. To represent this, we add to the Formal Story Specification (Fig. 42): a World; two new Nodes, namely JohnsMobilePhone, a Mobile, and

JohnsMobileAccess, an Access; a classification statement about JohnsMobilePhone (it is an Accessing Device in the first world) and specify that the existing classification statement about the university ATM holds in the second world, but doesn't hold in the first world. Additionally, we specify that JohnsAccount is both a Withdrawn Account and an Accessed Account.

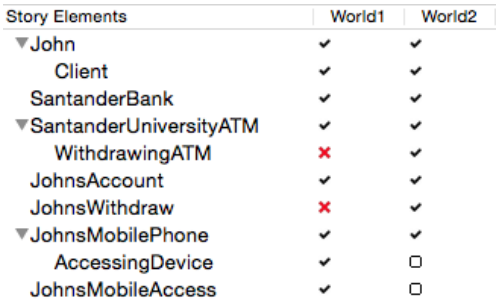

Fig. 42.Second Formal Story Specification for the Bank Model: John checks his balance on his mobile phone

Since there are two worlds and two devices specified, we assume a scope of 2 would be sufficient. However, the simulator cannot find an instance with a scope of 2. The smallest scope that satisfies the story is 3, which means the model is at least satisfiable. Checking the unsatisfiability core for the scope of 2 allows us to find hints about what made the model unsatisfiable for this scope. Fig. 43 shows the second story marked up in red. Other sections of the model have also been marked, such as the rigidity predicate and the disjointness of Mobile and ATM (although these are not shown in Fig. 43).

```
pred story2 [
                        World1:one World,
                        World2:one World,
                        John:one univ-World,
                        SantanderBank:one univ-World,
                        SantanderUniversityATM:one univ-World,
                        JohnsAccount:one univ-World,
                        JohnsWithdraw:one univ-World,
                        JohnsMobilePhone:one univ-World,
                        JohnsMobileAccess:one univ-World,]{
    John in World.Person
    John in World1.exists
    John in World2.exists
    John in World1.Client
    John in World2.Client
    SantanderBank in World.Bank
    SantanderBank in World1.exists
    SantanderBank in World2.exists
    SantanderUniversityATM in World.ATM
    SantanderUniversityATM in World1.exists
    SantanderUniversityATM in World2.exists
    SantanderUniversityATM in World2.WithdrawingATM
not SantanderUniversityATM in World1.WithdrawingATM
    JohnsAccount in World.Account
    JohnsAccount in World.WithdrawnAccount
    JohnsAccount in World1.exists
    JohnsAccount in World2.exists
    JohnsWithdraw in World.Withdraw
    JohnsWithdraw in World2.exists
    JohnsWithdraw not in World1.exists
    JohnsMobilePhone in World.Mobile
    JohnsMobilePhone in World1.exists
    JohnsMobilePhone in World2.exists
    JohnsMobilePhone in World1.AccessingDevice
    JohnsMobileAccess in World.Access
    JohnsMobileAccess in World1.exists
    World1.next = World2
}
```

Fig. 43.Second story highlighted with unsatisfiability core markup (in red).

Analyzing the unsatisfiability core indicates that the unsatisfiability is related to ATM, Phone (therefore, Device) and Account. The problem lies somewhere in the relation between JohnsMobilePhone, JohnsAccount and SantanderUniversityATM. More specifically, it is related to the fact that SantanderUniversityATM is not a withdrawing ATM in World 1 and JohnsAccount is a Withdrawn Account (in all worlds, since Withdrawn Account is a rigid classifier). This could lead to the following conclusion: since Withdrawn Account is rigid, and the minimum cardinality to Withdrawn is 1, there must be a Withdrawn linked to the account in the first world; SantanderUniversityATM is not a Withdrawing ATM in the first world, therefore there must be another ATM which is a Withdrawing ATM in the first world. Since JohnsMobile is a Device, and Mobile and ATM are disjoint, Device would need a scope of 3 to satisfy the simulation (1 for SantanderUniversityATM, 1 for JohnsMobile and 1 for the unspecified ATM in the first world). Raising the scope to 3 and inspecting the Formal Narratives (Fig. 44 and Fig. 45) could help reaching the same conclusion.

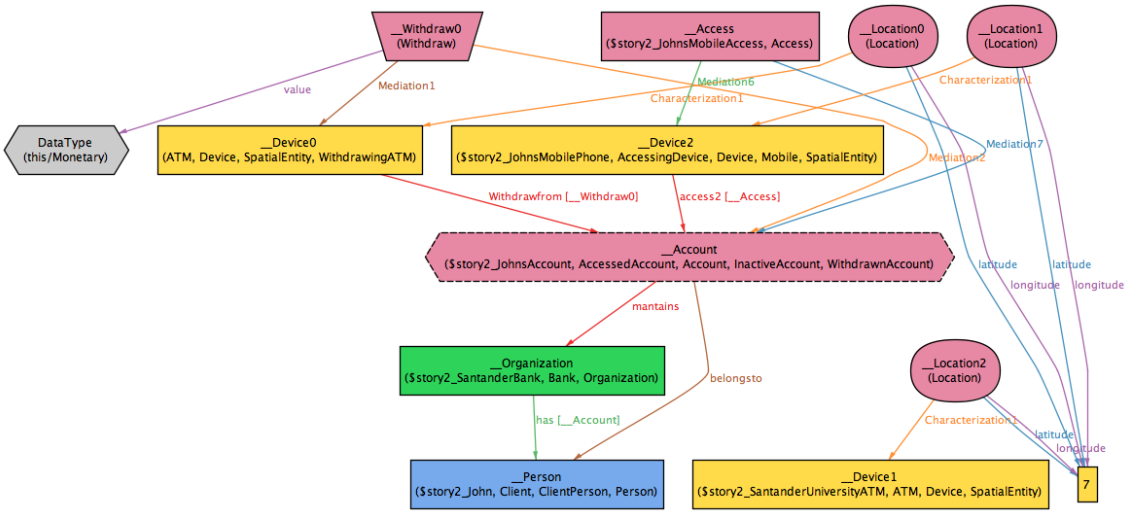

Fig. 44.First World of the second story. Unspecified ATM and Withdrawn are required to satisfy the story

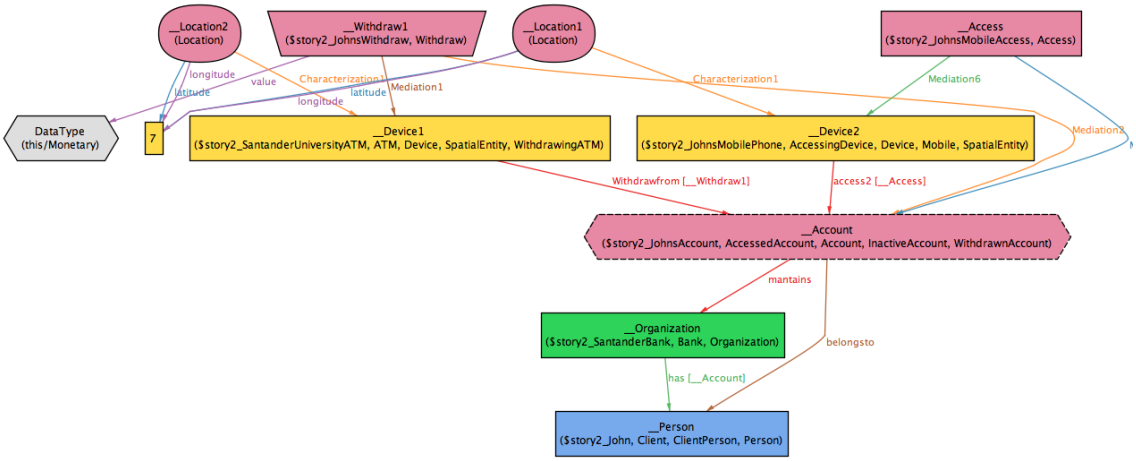

Fig. 45.Second World of the second story.

The narrative above exemplifies the problem with modeling Withdrawn Account as a rigid classifier: we cannot model a situation where an Account has no withdraws in one World and has a Withdraw in the next. For example, the following Natural Language Narrative could not be satisfied by the current model:

John opened a new Account in Santander. For the first month, he only checked his balance on his Mobile phone. On the second month he withdrew some money at an ATM.

The rigidity of the Withdrawn Account classifier, with the minimum cardinality of 1 with the Withdraw class enforces that any Withdraw Account must have Withdraws in all Worlds it exists. Therefore, the “first withdraw” scenario is not possible.

The Accessed Account classifier has the same structure and shares the same problem (Fig. 39). For the sake of brevity, we will omit the discussion and Formal Narratives to demonstrate the same problem for the Accessed Account classifier, as it is similar to the Withdrawn Account classifier.

There are two ways to solve the presented Withdrawn Account and Accessed Account problem. The first approach (Fig. 46) is to change the stereotype of the classes from Relator to Role. The second (Fig. 47) is to change the cardinality of the relationships from 1..* to 0..*. Both approaches are not standard OntoUML, as optional mediation relationships are not allowed and Roles can only specialize Substance classes. However, both options are widely used to deal with problems such as this.

We believe the best alternative for the model would be the one in Fig. 46, for the same reasons Guizzardi [49] defends the use of Roles for Substance classifiers. Additionally, the use of Roles for Moment classes has recently been defended in [44]. Since the two options we presented are commonly adapted to deal with this type of problem we will show why the approach in Fig. 46 is superior by showing the problems that may arise from adopting the model in Fig. 47.

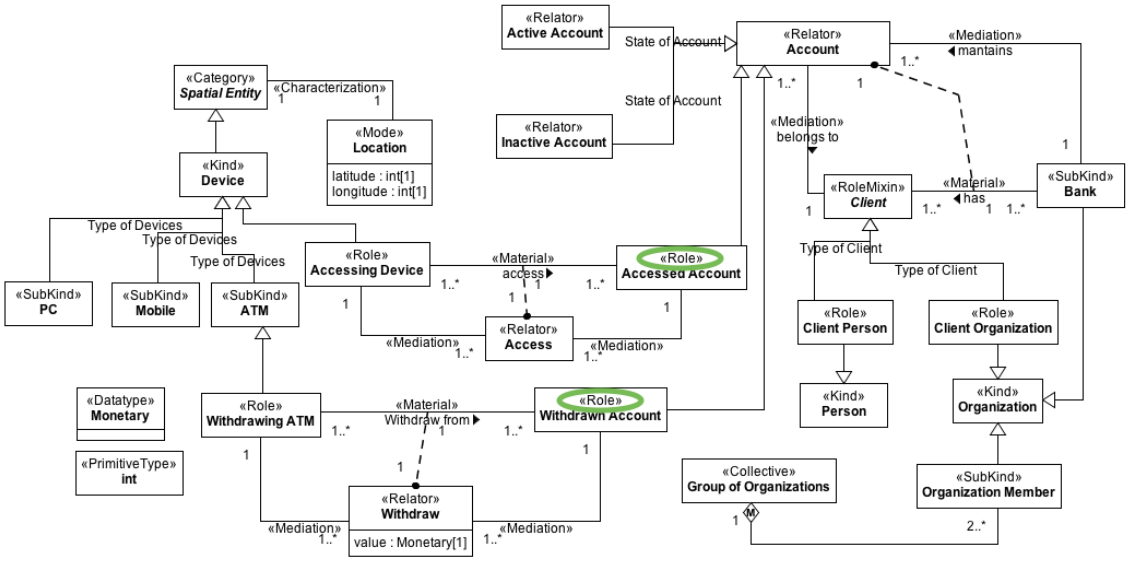

Fig. 46. Changing the stereotype of the classes fixes the problems found.

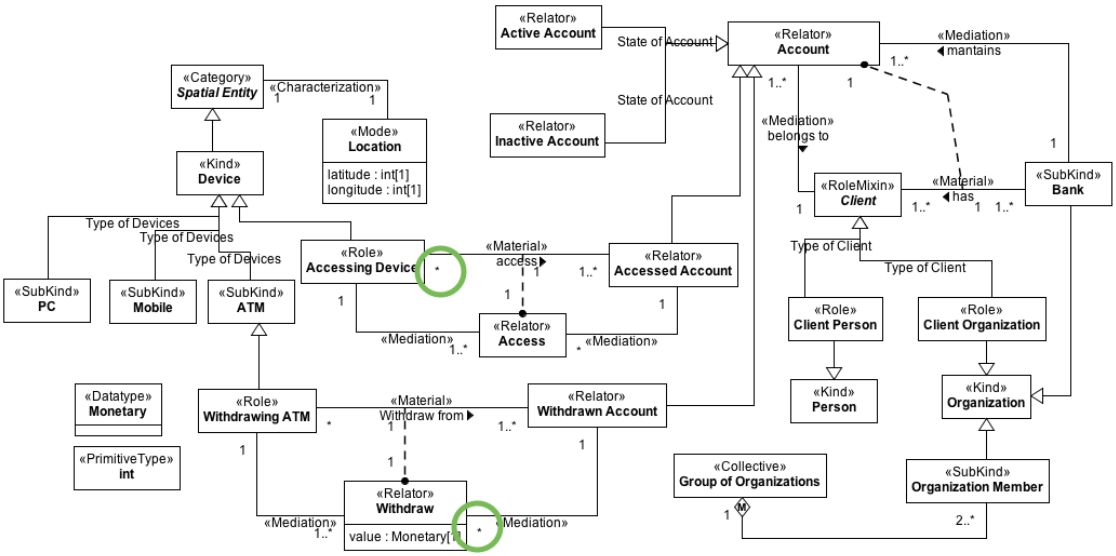

Fig. 47. Changing the cardinality of the relations fixes the problems found.

The model in Fig. 47 allows the unfortunate scenario depicted in Fig. 48, where there could be an Account that can never be accessed or withdrawn from. This version of the model could be improved to rule out this scenario by removing the Accessed Account and Withdrawn Account entirely, as shown in Fig. 49.

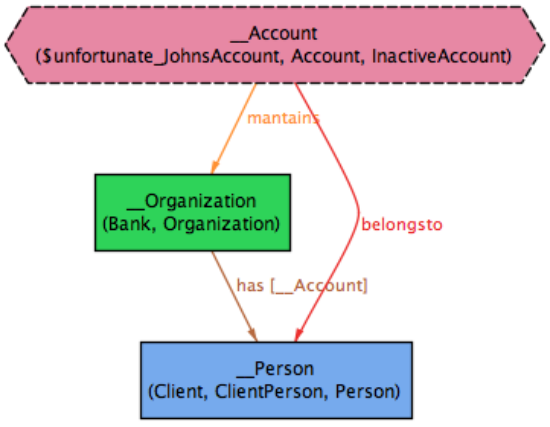

Fig. 48. Formal Narrative showing an account that cannot be accessed or withdrawn from

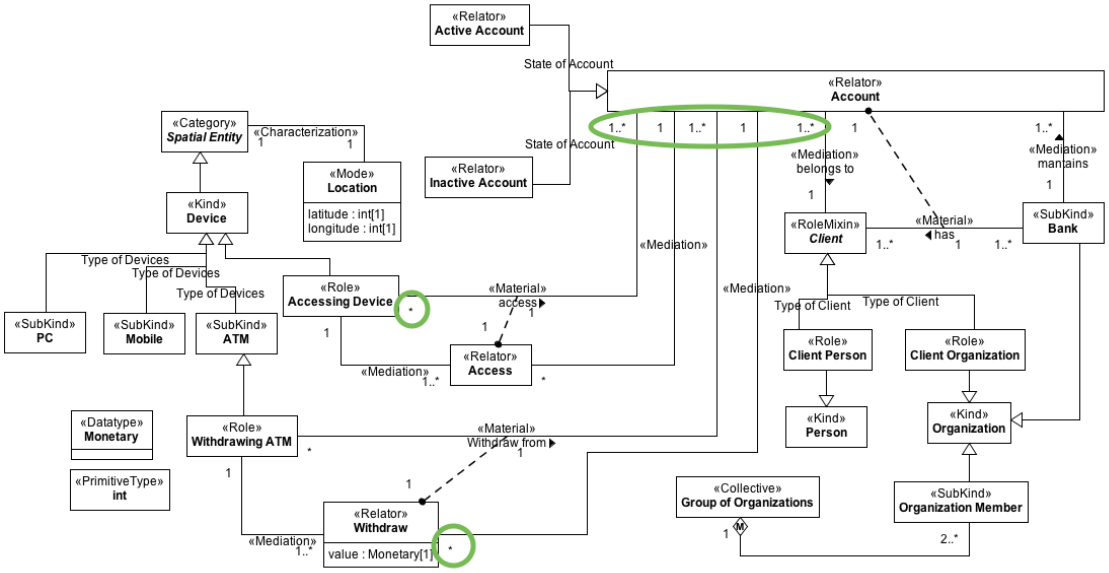

Fig. 49. Removing Accessed Account and Withdrawn account.

However, this amounts to a model that is less expressive because it omits relevant domain concepts. We will proceed with the validation using the model in Fig. 46 and adapt the Formal Story Specification to include classification statements for JohnsAccount (Fig. 50), specifying it to be an Accessed Account in the first World and a Withdrawn Account in the second. Running the story for the adapted model wields the following Formal Narrative (Fig. 51 and Fig. 52).

Story Elements	World1	World2
▼John	✔	✔
Client	✔	✔
SantanderBank	✔	✔
▼SantanderUniversityATM	✔	✔
WithdrawingATM	✘	✔
▼JohnsAccount	✔	✔
AccessedAccount	✔	▢
WithdrawnAccount	✘	✔
JohnsWithdraw	✘	✔
▼JohnsMobilePhone	✔	✔
AccessingDevice	✔	▢
JohnsMobileAccess	✔	▢

Fig. 50. Third formal story specification: adapting the second formal story for the model modifications

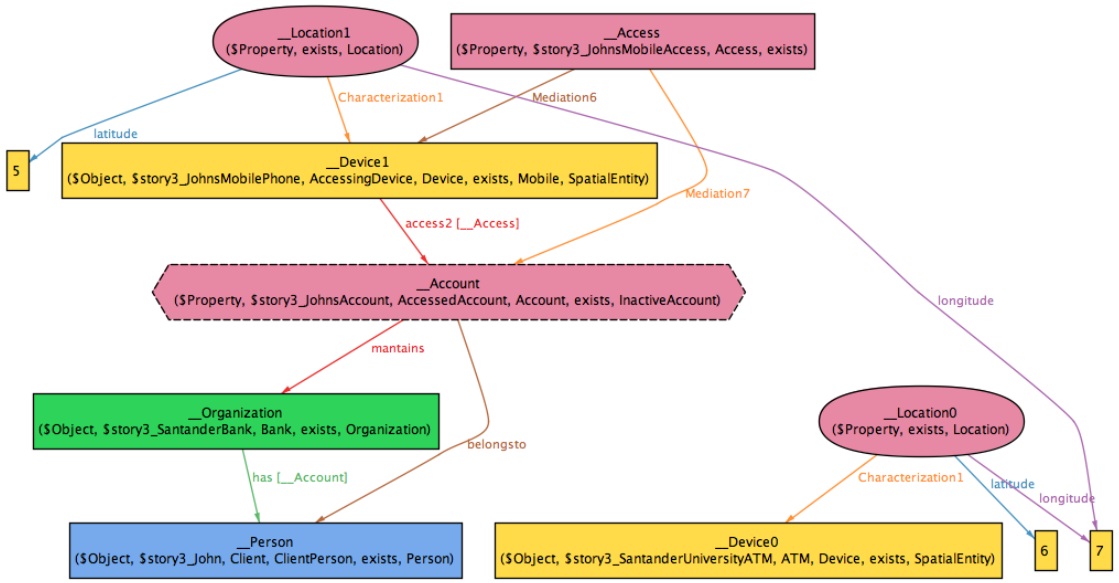

Fig. 51. First world of the third story. The elements specified in the Formal Story Specification are enough to satisfy the Formal Narrative

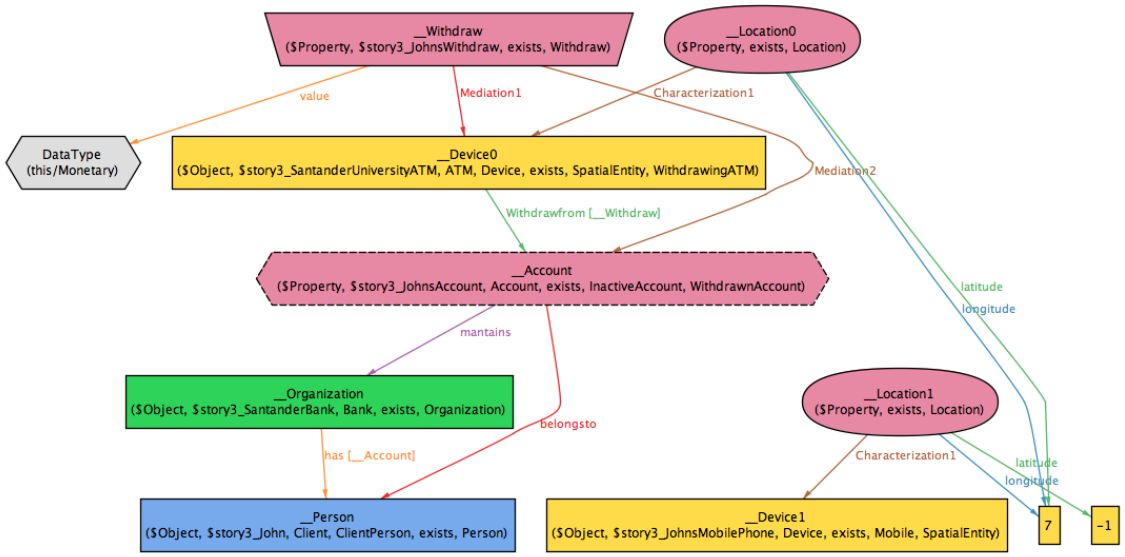

Fig. 52. Second World of the third story.

Going back to the original Natural Language Narrative, to formalize the fragment where John closes his account (makes it Inactive) we must change the stereotypes for the Active Account and Inactive Account classifiers in the model similarly to how we have changed the Withdrawn Account and Accessed Account. Since in this case the change is not motivated by an association, we use the Phase stereotype instead of Role (Fig. 53). Otherwise we could not have the Account change classes from Active to Inactive such as we see in Fig. 57 (where JohnsAccount is active) and Fig. 58 (where it is inactive). Fig. 54 shows the fourth story specification. In relation to the third story spec-

ification (Fig. 50) this one has an additional world in the beginning to represent the starting point and an extra world in the end, to represent the moment John makes the account inactive. The Formal Narratives generated based on these Formal Story Specifications are in Fig. 55, Fig. 56, Fig. 57 and Fig. 58.

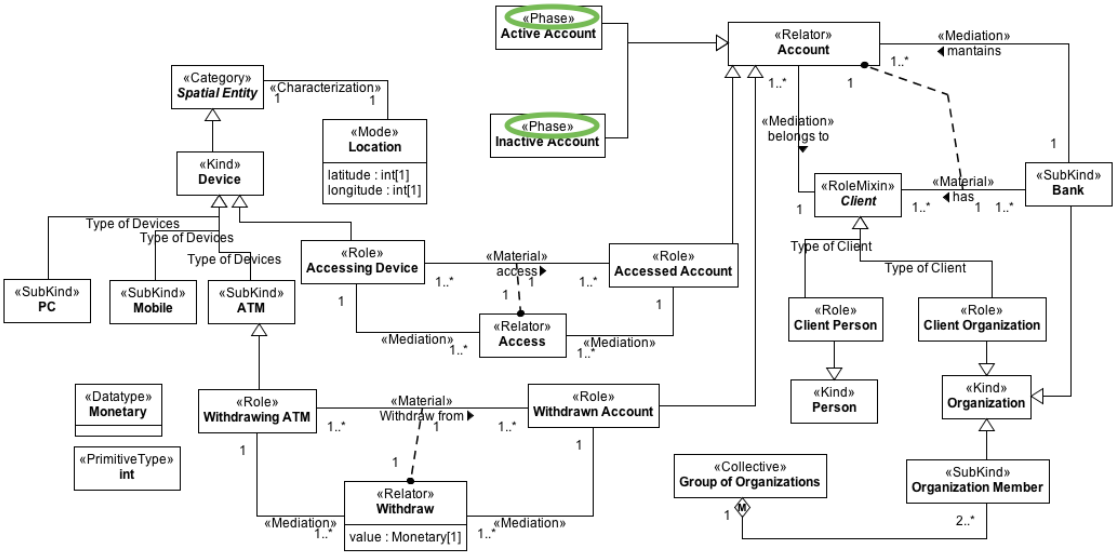

Fig. 53. Changing the stereotype for Inactive Account and Active Account from Relator to Phase.

Story Elements	World0	World1	World2	World3
▼John	✔	✔	✔	✔
Client	✔	✔	✔	✔
SantanderBank	✔	✔	✔	✔
▼SantanderUniversityATM	✔	✔	✔	✔
WithdrawingATM	✘	✘	✔	▢
▼JohnsAccount	✔	✔	✔	✔
ActiveAccount	✔	✔	✔	✘
AccessedAccount	✘	✔	▢	▢
WithdrawnAccount	✘	✘	✔	▢
JohnsWithdraw	✘	✘	✔	▢
▼JohnsMobilePhone	✔	✔	✔	✔
AccessingDevice	✘	✔	✘	✔
JohnsMobileAccess	✘	✔	✘	▢

Fig. 54. Fourth story specification. Added a world in the beginning to represent the starting point and a world in the end, to represent the moment John makes the account inactive.

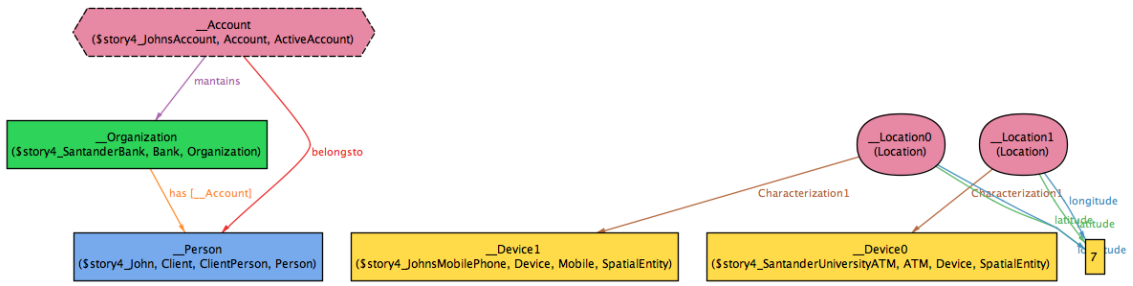

Fig. 55. First world of the fourth story: John has not yet accessed his account on his mobile phone.

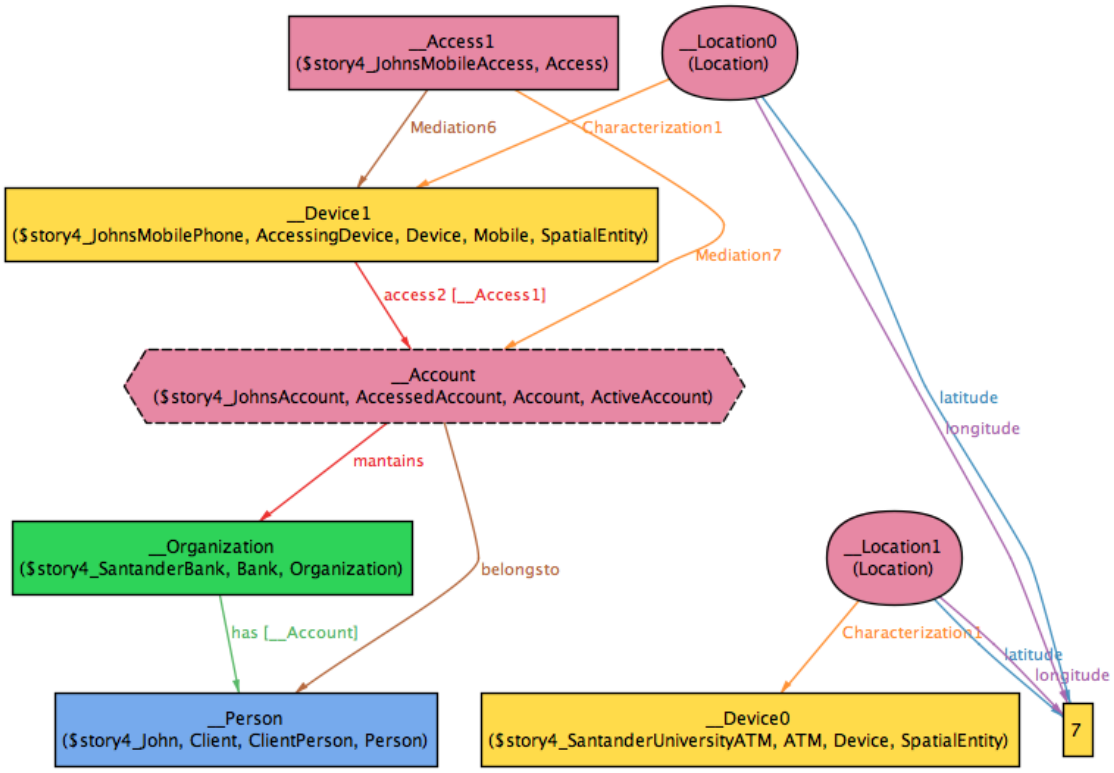

Fig. 56. Second world of the fourth story: John accesses his account on his mobile phone

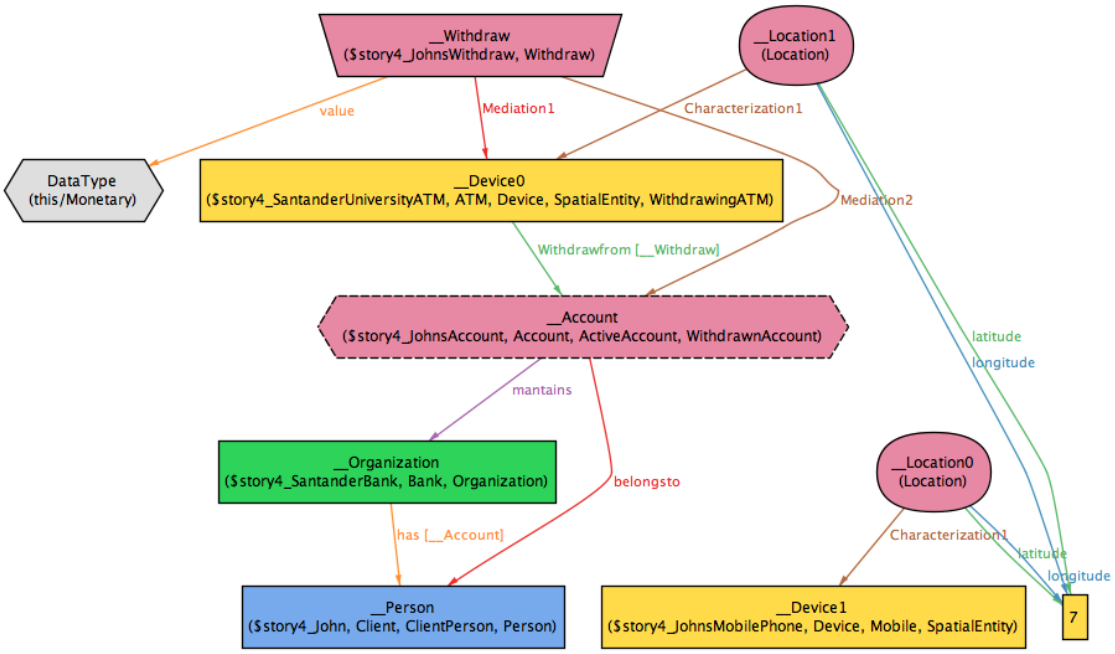

Fig. 57. Third world of the fourth story: John withdraws money from his account using an ATM

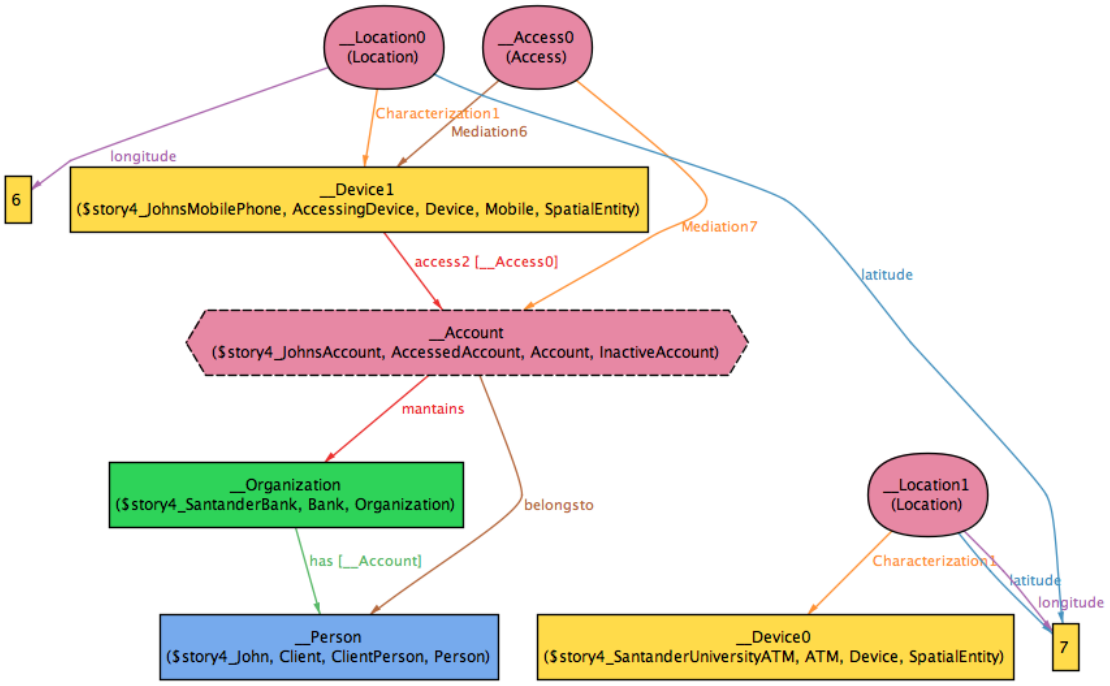

Fig. 58. Fourth world of the fourth story: John checks on his mobile phone that his account is indeed inactive.

d) – Conclusion

We have applied our method to the Bank model and detected fragments that are inaccurate in the sense that they were unfit to formally representing the Natural Language Narrative we elaborated for it. They were unfit because they did not allow dynamic classification of Accounts as Withdrawn, Accessed, Inactive or Active. Using Formal Story Specifications and showing the Formal Narratives generated based on them, we have justified the need to change the model and offered possibilities for changing it.

Applying the method to the model also revealed opportunities for improving coverage, such as including branches in the model and allowing withdraws there. Another improvement could involve restricting withdraws to inactive accounts to only be possible in branches and constrain ATMs to only allow withdrawing money from active accounts. We suggest the coverage of the model could be broadened by including the causes that can make an account inactive; in our example the account holder closed his account but maybe there are other reasons such as court order, fraud, too many incorrect password attempts etc.

Appendix D– Applying method to The Inventory Management System model

a) – The model

The Inventory Management System model is a generic model for stores. It models the Store, its employees responsible for selling items to customers, the labor contracts of such employees, the receipts for the items sold and the stock. Items in the receipt can be cancelled and the receipt can be cancelled as well. Stock can be active or inactive, and customers as well. A Customer can be either a Person or a Company.

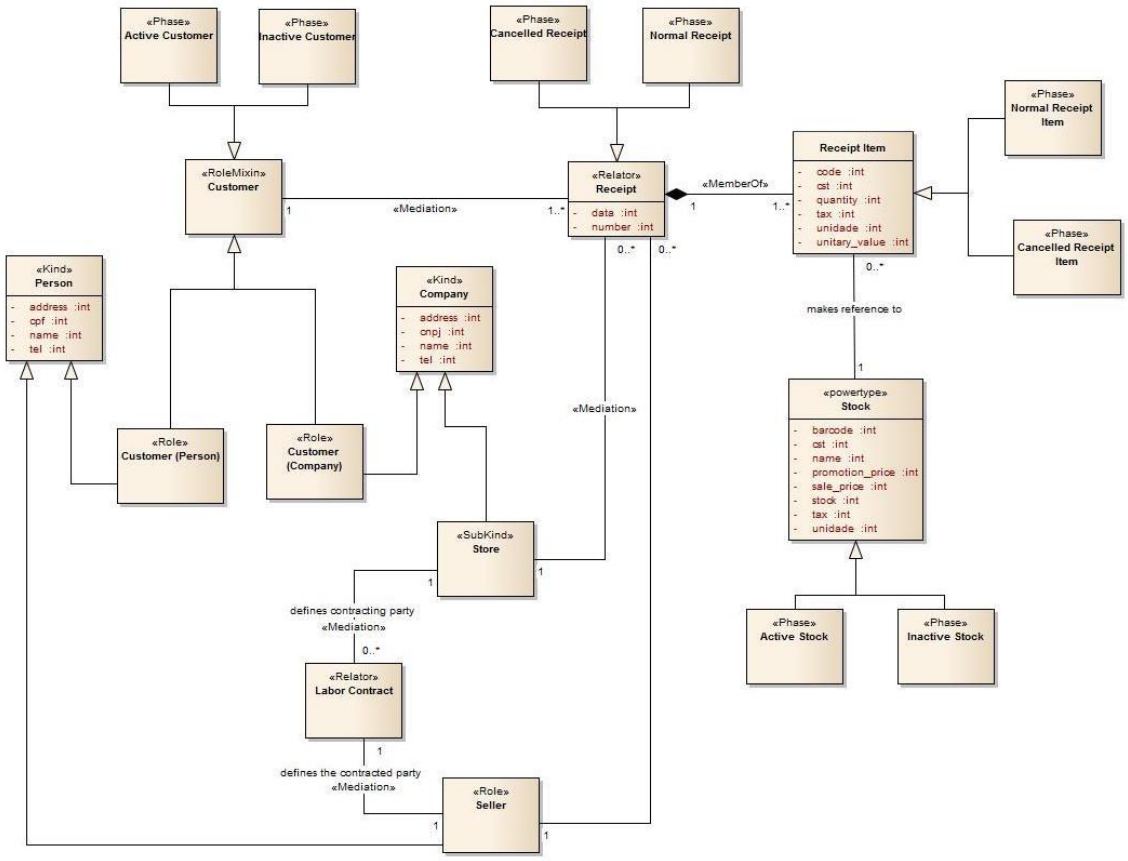

Fig. 59. A model for Inventory Management System.

The model is a fragment of a larger model that was developed for a business that made it available for this dissertation on the grounds of keeping both the company and the modeler anonymous.

b)– Overviewing Natural Language Narrative

*Mark was just hired to work at an office supply store as a cashier. He is happy to have signed a **labor contract** for the next five years, meaning he will be able to afford his children's health insurance.*

*Unfortunately, his first **customer** is a mean looking bald man. As the man approaches Mark's stall, Mark can see the desperate look on the other employees faces and starts sweating. The man brings a shopping basket full of things, puts it on the counter and grunts.*

Good morning, sir - said Mark as he started scanning the bald man's items

Ghrmnp - grunted the man, ignoring Mark

*"What an awful man", thought Mark. "I hope I can make everything alright". This was a big sale for his first try! There were **multiple items of the same kind** and fortunately he remembered how to multiply them, surely this guy would complain if he scanned them individually.*

*"BIP BIP" the scanner sounded as Mark added each **item** on the purchase*

*-Stop - said the bald man - these pencils are announced with a different **value** on the shelves.*

*Mark felt cold in his spine. "It's not my fault" he thought. Surely someone messed up in the **stock references.***

*-They are on **sale**, sir - said the manager standing behind Mark - I will ask someone to fix the price on the shelves, thank you for pointing that out to us.*

*Mark was baffled, he didn't notice the manager standing behind him and thought the price on the shelves were lower, not higher! Maybe the **customer** wasn't such a bad guy after all.*

-Well I guess that's good for me, then - said the bald man.

Mark scanned the erasers and the man halted again

-Those erasers were cheaper last week.

*-Well, those were on **sale** and now they are not anymore, sir - said the manager*

*-Then I won't take them. **Cancel** them, cashier.*

*Mark knew what to do and **cancelled the receipt item.** When he finished scanning all the items he was feeling good, he made no mistakes.*

-Will that be all, sir? -said Mark with a smile on his face

*-Yes, make the receipt to my **company**'s **CNPJ.***

*Mark looked to the computer screen and saw no option to enter a **CNPJ**, only to enter **CPF**. He turned to the manager and asked*

*-I didn't know; does that mean I have to **cancel** the **receipt?***

*-No, Mark, it's ok, see: all you have to do is check this box here to change from **Customer(Person)** to **Customer(Company)**. Now you can enter the cnpj.*

The sale was over, Mark was relieved.

-Thank you for shopping with us, sir! - Mark said as the man left

-Grhmp -grunted the man in response.

- Well done, Mark. Keep up the good work - the manager said as he patted Mark on the back and left. Mark could see on the other employees faces that they were happy on how well Mark performed under the stress and he was pleased with himself.

c) – Model assessment

The model must be adapted for us to conduct our assessment approach since it includes (i) a class without any stereotypes (Receipt Item), (ii) a class stereotyped as Powertype (Stock) and (iii) a relationship that is not stereotyped (between seller and receipt). We set the stereotype of Receipt Item to Kind, of Stock to Kind as well and the relationship between seller and receipt to mediation. While we are aware that Powertype is a better stereotype for Stock, neither the Menthor Editor nor the Story Modeler supports with Powertypes yet. Additionally, since our approach is not prepared to deal with attributes, we have omitted all of them.

This model includes other syntactical errors, which are negligible i.e. they do not interfere much with the model simulation. These errors include: (i) phases of Mixins are not allowed (active customer and inactive customer), (ii) a MemberOf relation for a relator (between Receipt and Receipt Item) is not allowed, the whole of a memberOf relation must be a collective (iii) two mediation relations have 0 as minimum cardinality and mediation relations must have a minimum 1 cardinality.

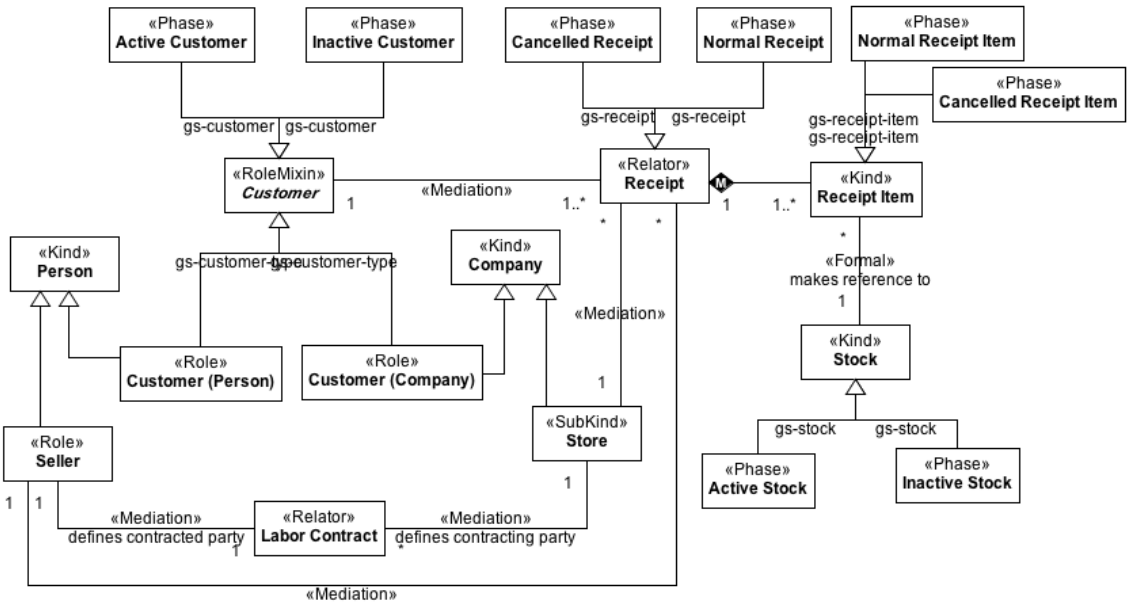

Fig. 60. Adapted Inventory Management System model

In the first formal story we will start by specifying Mark, the store and Mark's Contract. Fig. 61 shows the first Formal Story Specification and Fig. 62 shows the corresponding Formal Narrative generated by such Formal Story Specification.

The diagram notation is as follows: each figure represents an individual. Inside each figure, in the first line, there is the name of the individual, generated automatically by

the Alloy Analyzer. The name of the individual is the name of the signature, followed by a number, if and only if there is more than one individual of the same signature in that simulation. On the second line, between parentheses, there is a list of every set the individual belongs to. Sets that start with a dollar $ sign are the variables defined by our story predicates. These are prefixed with the name of the story predicate, followed by an underscore and finally followed by the variable name.

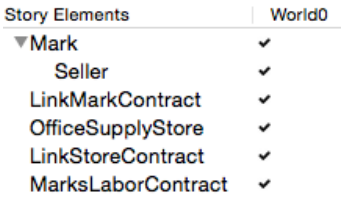

Fig. 61. First Formal Story Specification for the Inventory Management System model

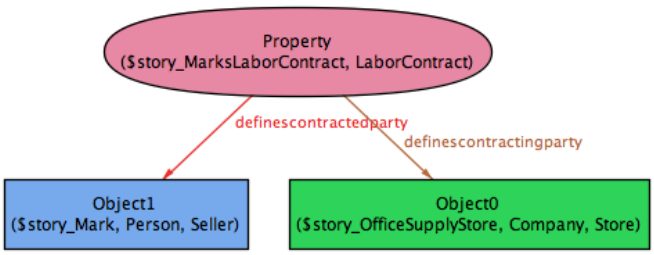

Fig. 62. First World of the first Formal Narrative for the Inventory Management System model. Mark has a contract with a store.

For the second formal story specification (Fig. 63) we will add the Bald Man as a customer and a receipt item; as well as a second world to represent the state of affairs before the Bald Man becomes a customer. The resulting Formal Narrative (Fig. 64 and Fig. 65) shows an unexpected scenario: the Bald Man is also an employee of the store in the second world. Since the Formal Narrative does not correspond to our intentions, we will constrain the Formal Story Specification further. However, before we do, we may inspect the results of this Formal Narrative and look for any insights it may give us. First, it seems plausible that an employee may also be a customer of the store he works. Second, we can see that the receipt is cancelled; while that seems to be something that should be allowed by the model, it is not the state of affairs we intended to represent i.e. it does not conform to our Natural Language Narrative.

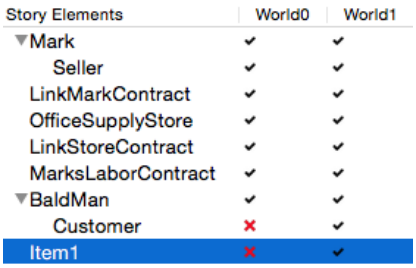

Fig. 63. Second Formal Story Specification for the Inventory Management System model

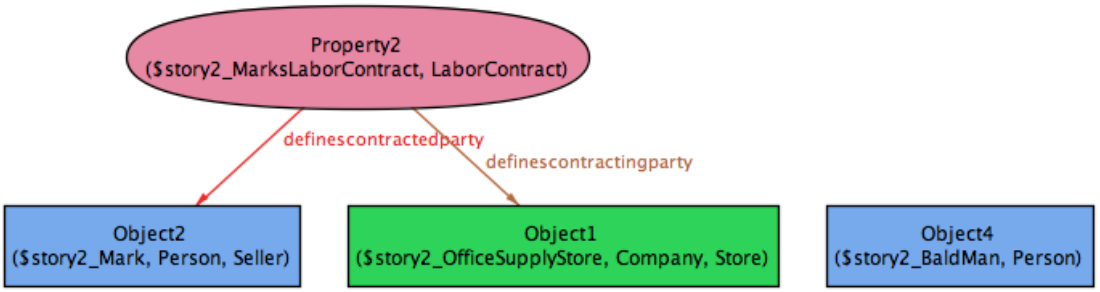

Fig. 64. First World of the second Formal Narrative for the Inventory Management System model. Mark has a contract with a store and the Bald Man is there.

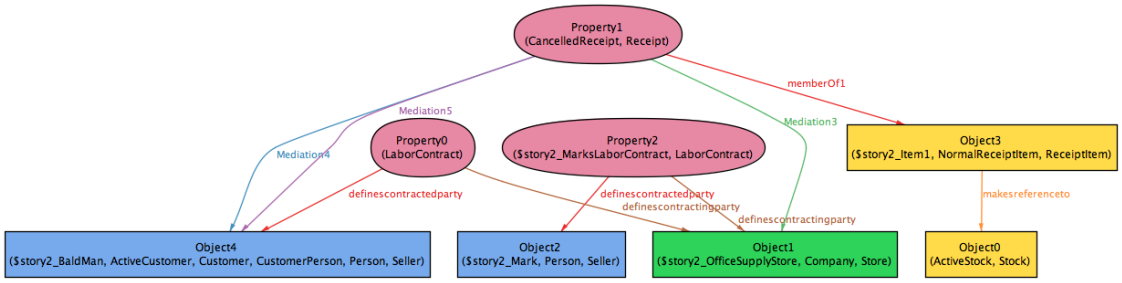

Fig. 65. Second World of the second Formal Narrative for the Inventory Management System model. Mark is selling an Item to the Bald Man but the Bald Man is also an employee of the same Store.

For the third Formal Story Specification we will specify that the Bald Man is not a Seller and specify the receipt and the link between the receipt and the receipt item. We will also specify a classification statement about the receipt that it is a Normal Receipt (and therefore not a Cancelled receipt as it happened in the last Formal Narrative). We will go ahead and define the Item as Normal as well, to avoid the similar problem of having the item instantiating the unintended phase.

Story Elements	World0	World1
▼Mark	✔	✔
Seller	✔	✔
LinkMarkContract	✔	✔
OfficeSupplyStore	✔	✔
LinkStoreContract	✔	✔
MarksLaborContract	✔	✔
▼BaldMan	✔	✔
Seller	✘	✘
Customer	✘	✔
▼Item1	✘	✔
Normal Rcpt Item	✘	✔
LinkRcpt-rcptItem	✘	✔
▼MarksRcpt	✘	✔
NormalRcpt	✘	✔

Fig. 66. Third Formal Story Specification for the Inventory System Management model.

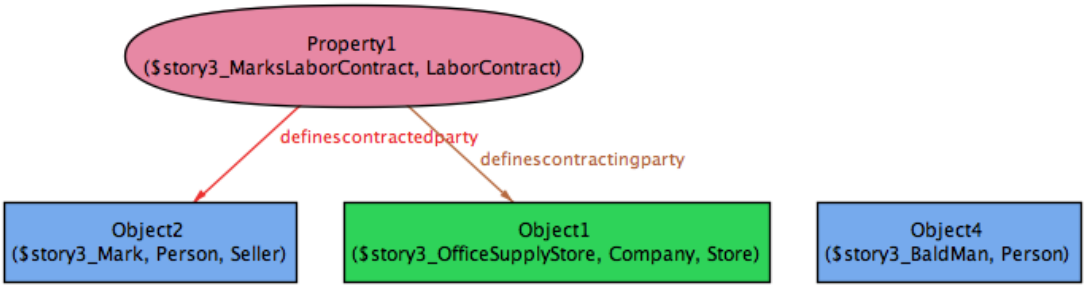

Fig. 67. First World of the third Formal Narrative for the Inventory Management System model.

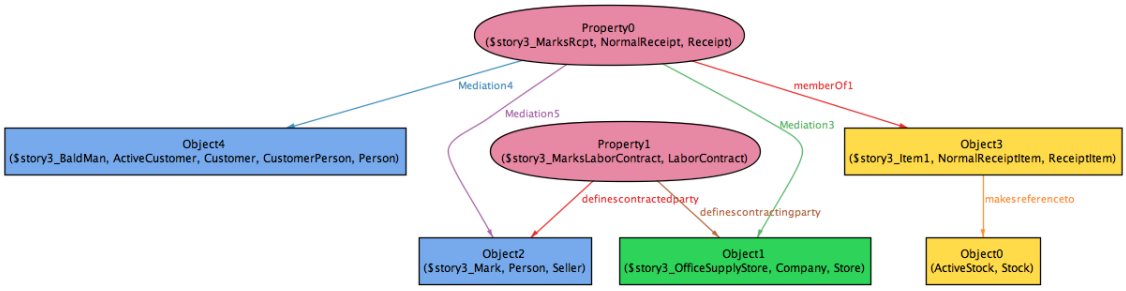

Fig. 68. Second World of the third Formal Narrative for the Inventory Management System model. Mark is selling an Item to the Bald Man.

The third Formal Narrative now behaves as we intended so we will proceed to creating the fourth Formal Story Specification to add the erasers to the receipt and later cancel them. We will also add to the specification the Stocks corresponding to the Item1 and to the erasers, since they are different. Both should be active stocks for the duration of the narrative so we add classification statements to ensure that they will. Links between the items and the corresponding stocks are also specified.

Story Elements	World0	World1	World2	World3
▼Mark	✔	✔	✔	✔
Seller	✔	✔	✔	✔
LinkMarkContract	✔	✔	✔	✔
OfficeSupplyStore	✔	✔	✔	✔
LinkStoreContract	✔	✔	✔	✔
MarksLaborContract	✔	✔	✔	✔
▼BaldMan	✔	✔	✔	✔
Seller	✘	✘	✘	✘
Customer	✘	✔	✔	✔
▼Item1	✘	✔	✔	✔
Normal Rcpt Item	✘	✔	✔	✔
LinkItemStock	✘	✔	✔	✔
LinkRcpt-rcptItem	✘	✔	✔	✔
▼MarksRcpt	✘	✔	✔	✔
NormalRcpt	✘	✔	✔	✔
▼Erasers	✘	✘	✔	✔
Cancelled	▢	▢	✘	✔
LinkEraserItemStock	✘	✘	✔	✔
▼Item1Stock	✔	✔	✔	✔
Active Stock	✔	✔	✔	✔
▼EraserStock	✔	✔	✔	✔
Active Stock	✔	✔	✔	✔

Fig. 69. Fourth Formal Narrative for the Inventory Management System model. Mark is selling an Item to the Bald Man.

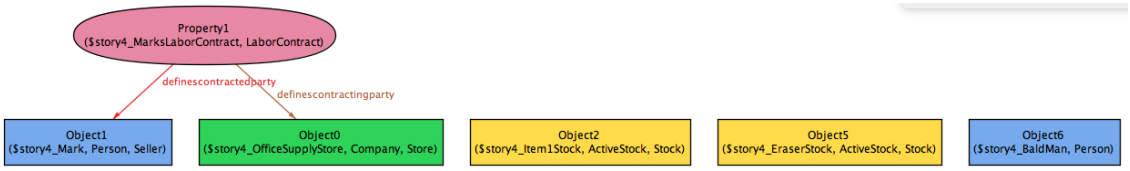

Fig. 70. First World of the fourth Formal Narrative for the Inventory Management System model. Mark is a seller in the office supply store.

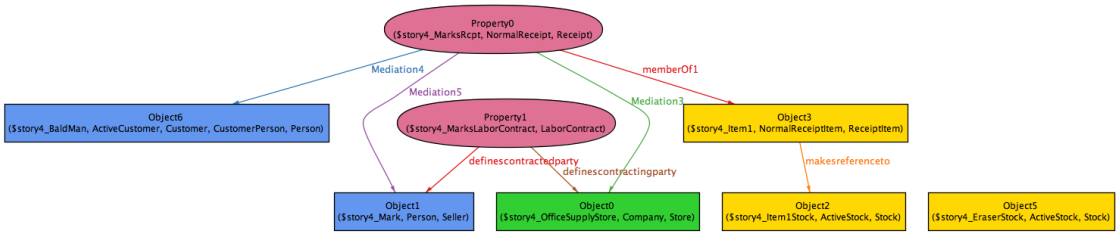

Fig. 71. Second World of the fourth Formal Narrative for the Inventory Management System model. Mark is selling an Item to the Bald Man.

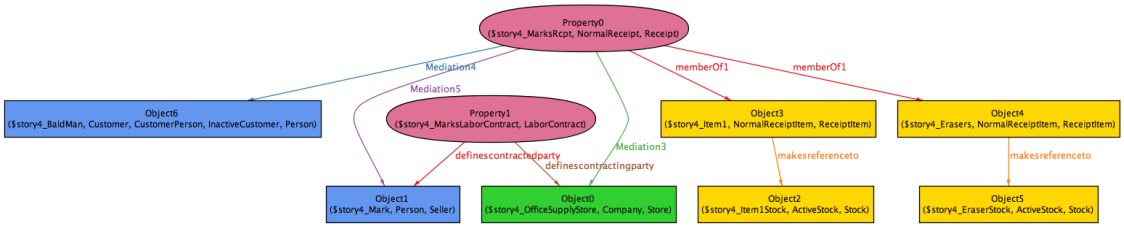

Fig. 72. Third World of the fourth Formal Narrative for the Inventory Management System model. Mark adds an eraser to the receipt.

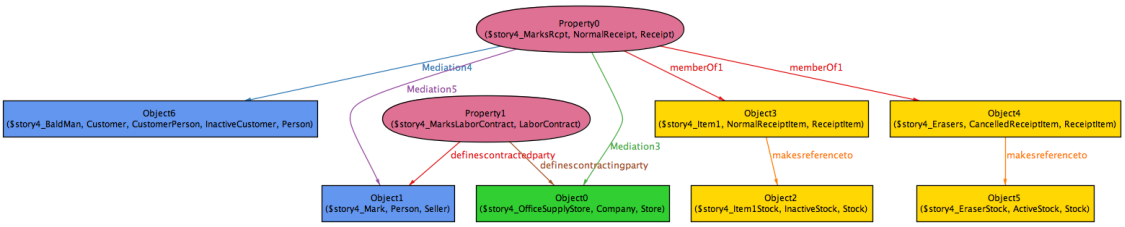

Fig. 73. Fourth World of the fourth Formal Narrative for the Inventory Management System model. Mark cancels the eraser.

In the fourth Formal Narrative (Fig. 70-Fig. 73) everything conforms to our intentions and no further changes are necessary.

The last part of Natural Language Narrative to formalize would be to switch the customer of the receipt from the Bald Man to his company. However, this is not possible according to the model. Since the receipt is modeled as a Relator and the relationship between the receipt and the customer is a mediation relation, the receipt is existentially dependent on the customer, meaning it cannot change customers. That means we probably specified in the Natural Language Narrative some state-of-affairs that were not anticipated by the original modelers.

Alternatively, maybe the modelers thought of the receipt as something that only exists after it is emitted and therefore would not change between customers (bald man and his company), but would be printed with the right customer to begin with. This would mean there are no concepts that represent the intermediate state of the purchase e.g. in the moment that Mark has scanned the first few items but has not scanned the eraser yet.

If the modelers are not interested in the dynamic aspects of the receipt, we could model the relationship between the receipt and receipt item as an existentially dependent relationship by adding the essential and inseparable meta-attributes to the relation or modeling them as mediation or characterization. Without these attributes, the model allows the situation where a receipt item from one receipt can be a part of another receipt. To show this strange situation we have created the Formal Story Specification (Fig. 74) and Formal Narrative (Fig. 75 and Fig. 76) below. The same Receipt Item (Object3) is show to be part of two different receipts (Property2 and Property1).

Story Elements	World0	World1
▼Mark	✔	✔
Seller	✔	✔
Customer	✘	✘
Receipt1	✔	✘
LinkRcpt1-item	✔	✘
Receipt2	✘	✔
LinkRcpt2-item	✘	✔
Item	✔	✔
▼BaldMan	✔	✔
Seller	✘	✘
Customer	✔	✔

Fig. 74. Formal Story Specification to show a situation where a Receipt Item could be part of two different Receipts in different points in time

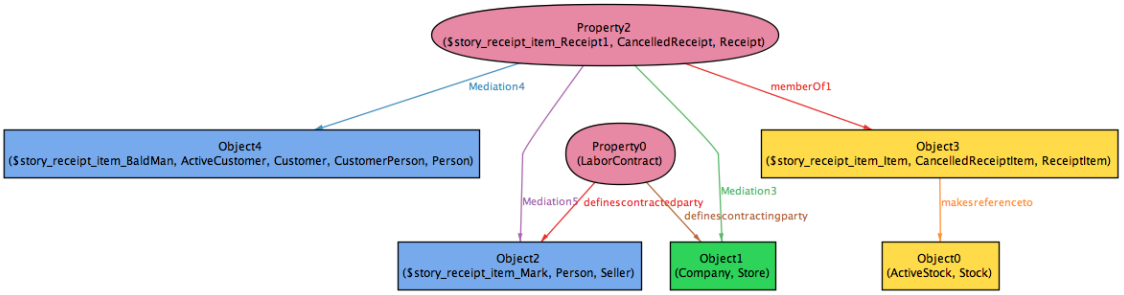

Fig. 75. First World of the Formal Narrative showing that the same Receipt Item could be used in two different Receipts. Object3 is memberOf Property2

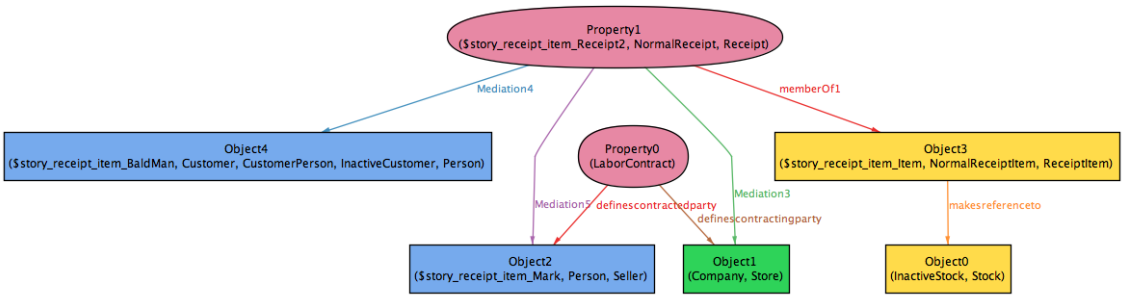

Fig. 76. Second World of the Formal Narrative showing that the same Receipt Item could be used in two different Receipts. Object3 is memberOf Property1

To avoid the situation described above, we propose two options for modeling the domain. The first would be to model the part-of relation as Inseparable and Essential (Fig. 77). The second is to change the relationship to characterization and change the stereotype of the Receipt Item to Mode (Fig. 78).

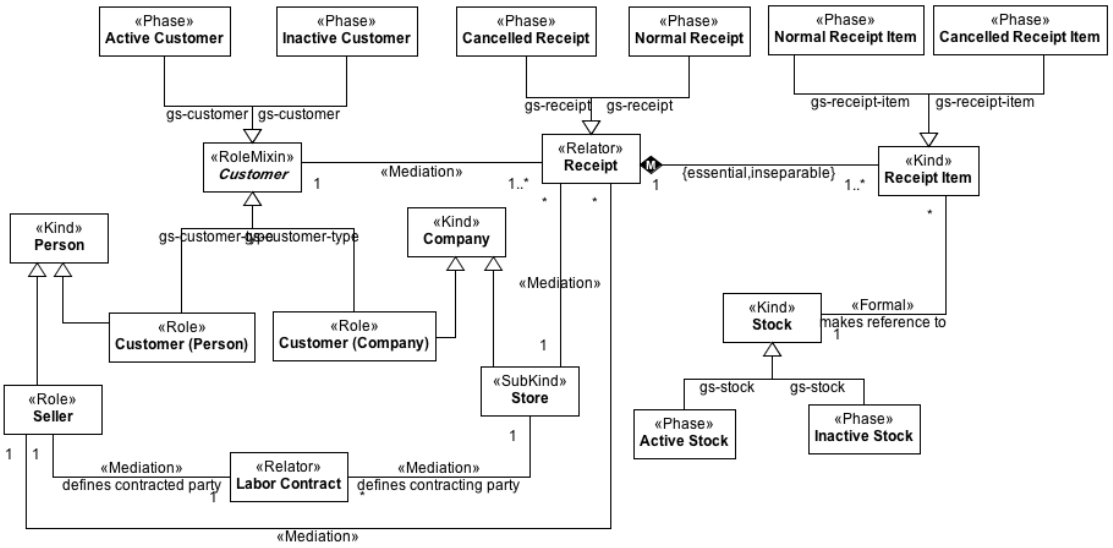

Fig. 77. Adapting the model by adding {essential, inseparable} to the parthood relationship.

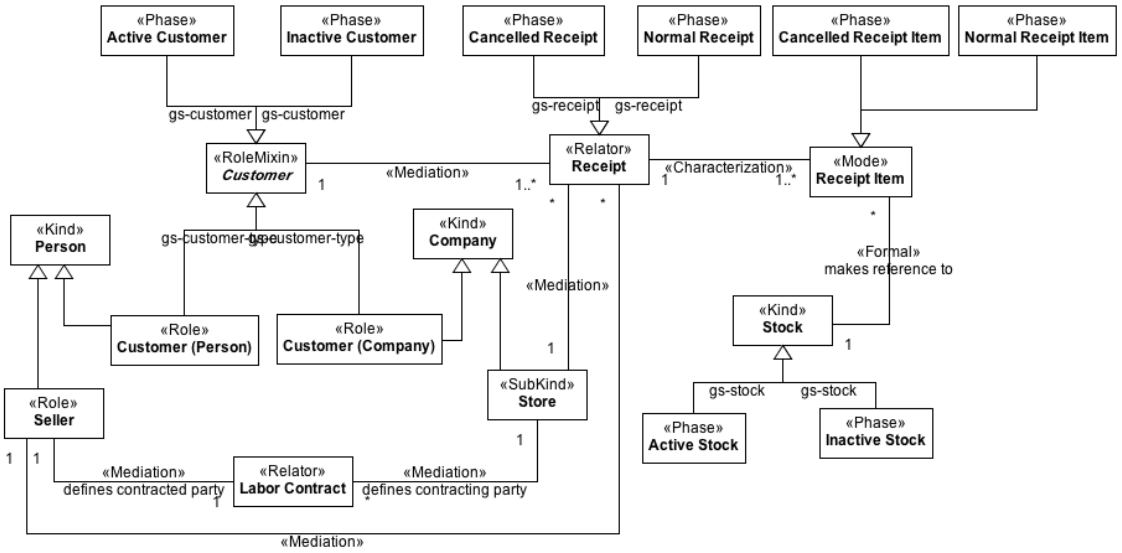

Fig. 78. Adapting the model changing receipt item to Mode

d)– Conclusion

In this annex we have exemplified intentionally creating Formal Story Specifications and Formal Narratives that show invalid state-of-affairs to motivate a change in the model.

The application of the method on this model revealed some limitations of the method. Attributes are an important part of this model and not having support for attributes limited the expressivity of our Formal Story Specifications; we could not model much of what was expressed in the Natural Language Narrative. For example, we could not exemplify “quatity” in a receipt item to show that multiple items were sold using a sin-

gle Receipt Item, or to show how the “sale_price” specified in the Stock of an item type reflects the price of the unitary value of an item, which in turn reflects the total price of the item. Another example is showing how the value of the product value may be changed using the “promotion_price”.

Appendix E– Applying the method to OntoEmerge

a) – Introduction

OntoEmerge is a well-founded core ontology for Emergency Plans. The model is intended to support the generation of Emergency Plans for organizations. The project was developed in cooperation between Brazilian universities in a cooperative project between NEMO (from UFES) and GRECO (from UFRJ) in the scope of the CNPq PRONEX program and resulted in some publications [31]. The model is very large and composed of multiple diagrams. In this appendix, the first person in the singular ("I") is used to clarify the author's participations in the process and contrast it with the participations of the modeling team. The first person in the plural ("we") refers to both the author and the modeling team.

The model uses many stereotypes that are not covered by OntoUML2Alloy such as Event, Higher Order Universal and Disposition. Nevertheless, I have engaged with the modelers to validate the models using Natural Language Narratives and discussing the possible instantiation of the Natural Language Narratives with them. This resulted in improvements of the model, which we discuss in this annex.

This annex exemplifies a way to use our method and approach which is different from the way we have performed in the rest of the work. Here we do not create Formal Story Specifications and we do not generate Formal Narratives, we use Natural Language Narratives alone to negotiate meaning between people: the narratives served as confirmation to me that I had understood the model correctly and this discussion alone led to model improvements.

We have worked with two fragments of the model: "Risk and Planned Activities" which models the notion of Risk, Hazard, Damage and events around those concepts and "Installation" which deals with the facilities that may be covered by the emergency plans. Each fragment of the model will be presented along with the corresponding Natural Language Narrative in the following sections and the whole model will not be presented. The whole model is available at http://www.menthor.net/ontoemerge.html

b)– Risk and Planned Activities diagram

In our validation sessions with the experts we started with a Natural Language Narrative about the Risks and Planned Activities diagram (Fig. 79). The narrative below, along with the discussion based on the narrative, helped consolidate the distinction between Hazard, Hazardous Event, Vulnerability, Risk and Damage. By exemplifying the concepts, we could differentiate between them and their roles in describing the phenomena. Before creating the narrative, we established a context for the story, exemplifying the class instantiation (between brackets) and then created a narrative based on such context, as follows.

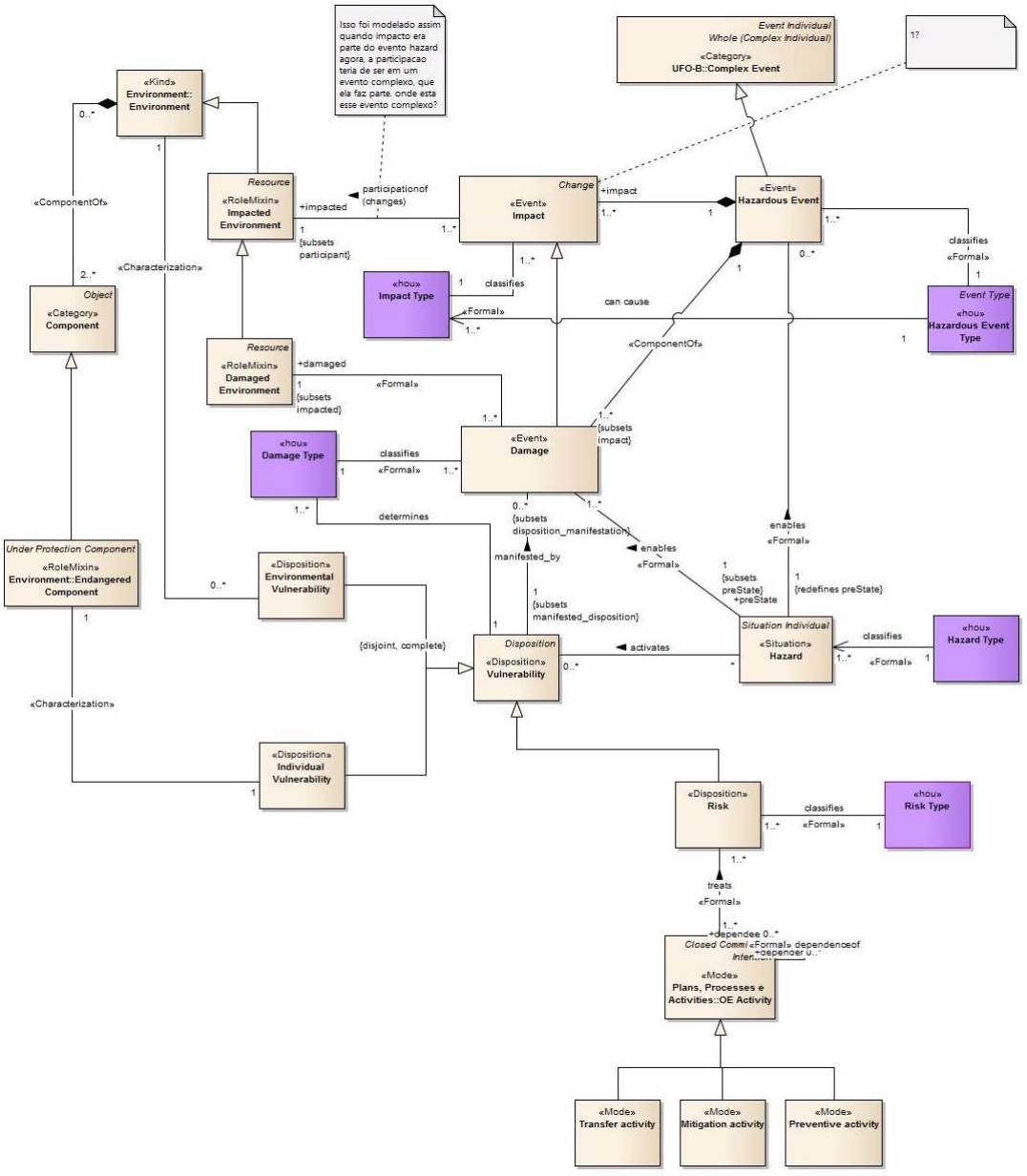

Fig. 79. Risk and Planned Activities diagram before the modifications

The university of Valencia has the *Risk of Run Over* [Risk <<Disposition>>] catalogued in their emergency plans. Whenever there is an inappropriate circulation of vehicles inside campus [Hazard<<Situation>>], the Risk of Run Over [Risk <<Disposition>>] of the university may manifest itself when a run over [Hazardous Event <<Event>>] actually happens. The Risk of Run Over is a characteristic of the university that may be manifested in a dangerous situation. When such situation entails the Running Over event [Hazardous Event <<event>>], such event is understood as the manifestation of the Risk of Run Over of the Valencia University. With this context in mind, consider the following Natural Language Narrative:

Last Friday Fred was driving in the University of Valencia and lost control of the vehicle, climbing on the sidewalk and running over Joanna and then crashing into the wall of the Informatics Department of the University of Valencia.

Joanna was severely injured; she had broken some bones. The crash was so violent it left a crack in the walls of the department.

Since Fred's vehicle was over the sidewalk, an Inappropriate Circulation [Hazard <<Situation>>] was configured, which resulted in a Run Over [Hazardous Event <<Event>>], i.e., activated the Risk of Run over of the University of Valencia [Risk <<Disposition>>]. Moreover, such situation allowed another Hazardous Event, namely the collision of the vehicle to the campus building. Each event caused a different damage.

In the first case, the damage was caused to Joanna. However, the model connects Impact and Damage to Environment i.e., it does not allow the participation of people in Damage or Impact events. Therefore, it seems to me that, in the model, the participation should be connected to Component and not to Environment, since the components are the elements that can be Under Protection (classes from the Environment diagram).

In the second case, about the damage to the building, it was unclear to me what class in the model building would instantiate. It is specified in the model that there can be Components that are built, which seems to be the category the building would instantiate. Therefore, it doesn't seem to be that the building is classified as Environment. It was not clear to me what Environment would be in this case. I imagine it would be the

University of Valencia, the totality of everything which is in risk, every component. Later, interviewing the modelers showed that was indeed the case.

In discussion with the modelers, we found that it made sense to include the components as participants of the Damage Event. We modified the model accordingly, to include this possibility (Fig. 80).

Additionally, we found a case of concept overload for Damage. In our narrative, Joanna's participation in the accident does not describe fully the damage inflicted in her. Consider that Joanna broke some bones in the crash. The damage, as it is modeled, is an Event, the manifestation of the Risk and the participation of Joanna in the accident. That is, there is nothing in the model that describes her broken bones, that "damage" that she suffered; which would probably be a Mode that inheres in her. The same could be said about the building the car crashed into. Suppose the wall was cracked in the crash. The model is not prepared to describe such "damage", only the "damaging event", which is called here "damage". The model should include some Mode that inheres in the damaged components that characterize such physical damage.

While discussing about this, we found there is an additional example of damage which is not covered by either definition. For example, consider a person which is dislodged due to a flooding. There is a physical damage to the structure (house) but there is also a damage to the human, which is being dislodged. What is such damage? Surely not an event neither a mode of the person. Is it a situation? We did not reach a conclusion on this discussion, but the Narrative and everything that was debated around it at least served to expose this limitation and opened an opportunity for improving the model.

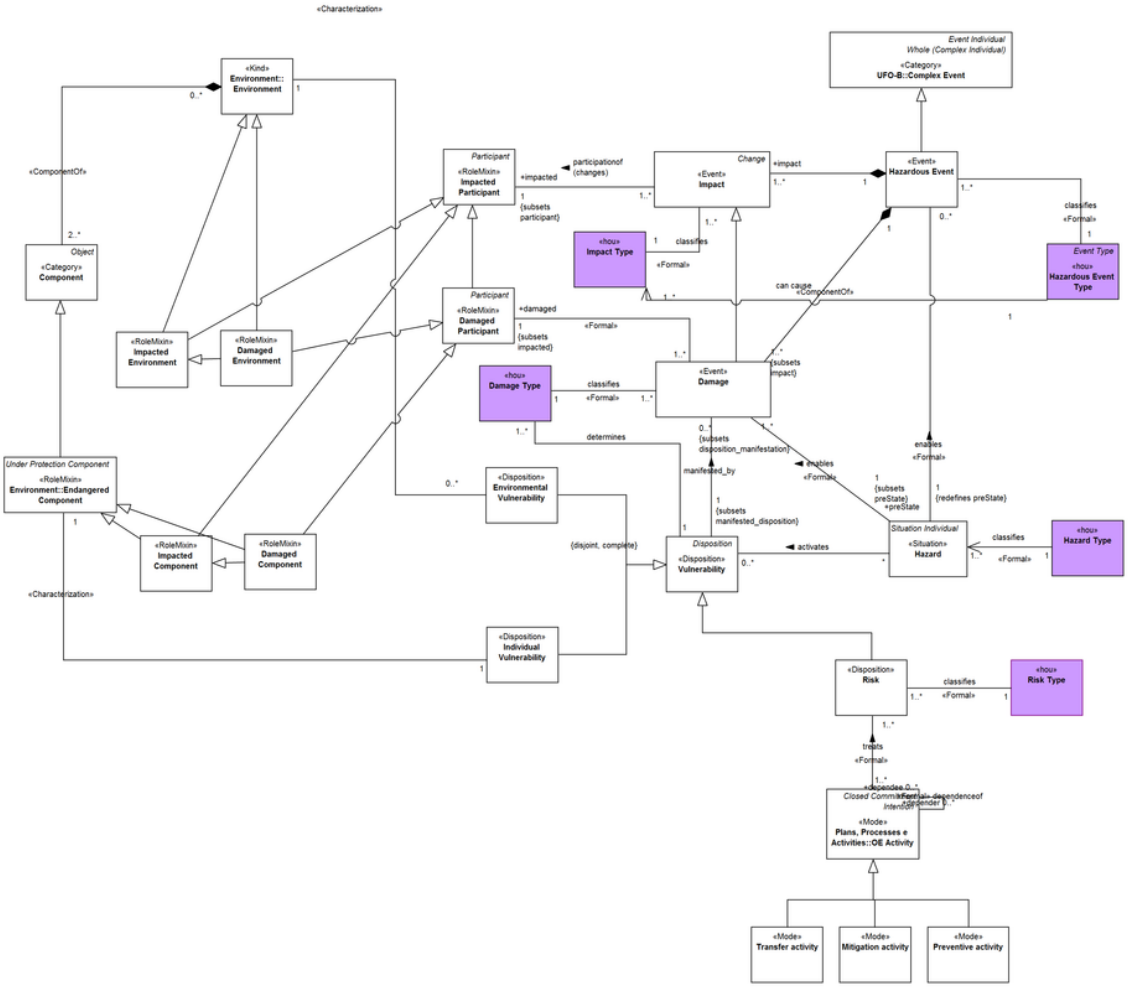

Fig. 80. Risk and Planned Activities diagram after the modifications

c) – Installation diagram

The Installation diagram models Installations i.e. places where people conduct business activities. This diagram models the people and organizations that may own or conduct business in them, as well as types of installations, the evacuation of said installations and the place where people evacuate to, among other details.

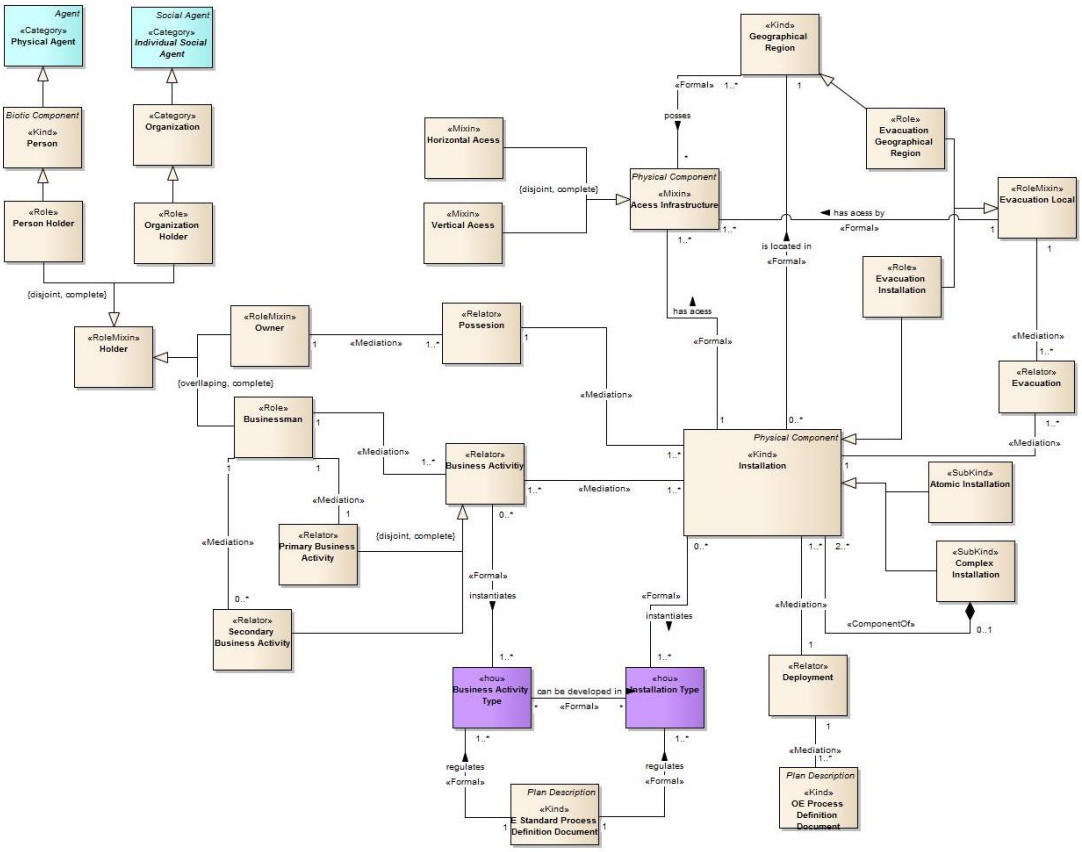

Fig. 81. Installation diagram before the modifications

To exemplify the concepts of the diagram, I created the following Natural Language Narrative:

> *In the university of Valencia there is a Datacenter (named data-center0013) inside a container (container 0025). The container is in an open field. Two technicians are conducting repairs to the electrical cables when one of them makes a mistake while plugging an equip-ment, is electrocuted and passes out. The accident starts a fire and the conscious technician drags his colleague outside through the only door of the container. He calls the firefighters and they send a team to the site. A firefighter arrives and extinguishes the fire using a chemi-cal powder fire extinguisher.*

The experience of discussing the Natural Language Narrative based on the Installation diagram with the modelers was rather unusual. It was both a failure and a success. It failed because my Natural Language Narrative turned out to be wrong, I assumed a container could be an Installation when in fact it couldn’t be. This happened even though I used a document provided by them which mentioned that a Container could be

an Installation. This revealed that the notion of Installation was not agreed on even on the original modeling team; they had documents that incorrectly stated a Container could be an installation. Revealing such false agreement can be considered a success.

We ignored this detail and kept using the example considering that the story could work the same if the Installation was some other type of building. Discussing the concept of Evacuation, I asked if the Evacuation Local (i.e. Evacuation Zone) would be anywhere the people evacuated to and they answered that no, the Evacuation Local was a place defined in the Emergency Plan. However, the model constrains that an Evacuation Local must related to an Evacuation. To reveal the problem that may arise in this case, imagine a scenario where the institution defines an Evacuation Local to an Installation where no Evacuations ever occurred. In this case the mandatory relationship between Evacuation Local and Evacuation cannot be satisfied. Therefore, I pointed out, there was probably a case of concept overload in Evacuation class. It overloads both the concept of planned evacuation and evacuation execution. Therefore we have corrected the model to introduce both concepts (Fig. 82).

Accidently, the narrative also revealed a concept that was not included in the model: when the technician removes his unconscious partner from the place of danger, that would constitute a “removal” and not an “evacuation”. Removal was not modeled.

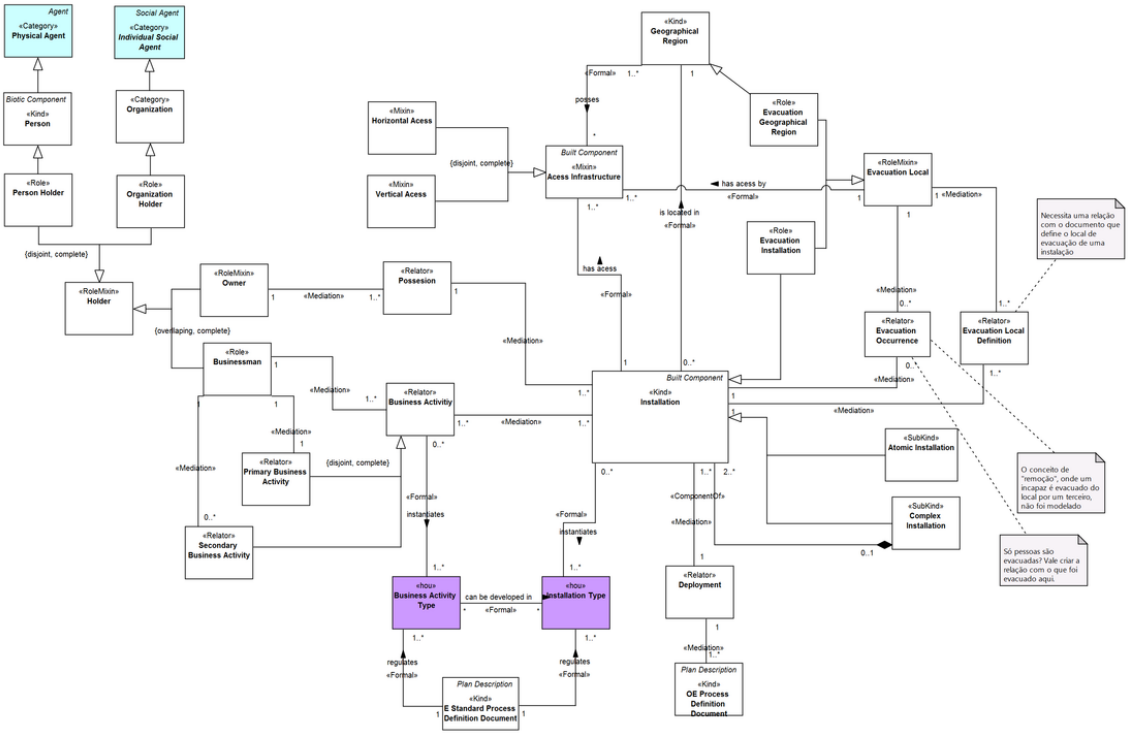

Fig. 82. Installation diagram after the modifications

d) – Conclusion

This experience with validating an existing model with the modelers and experts was very insightful. It showed that using Natural Language Narratives and discussing their formal counterparts alone can help validation. I suppose this was especially useful due to the very abstract nature of the terms in the model. The model was presented in [31] and my contributions validating the model (which were not limited to the validation session described in this annex) was significant enough to justify the status of co-author in the paper.