Formalization of the i* Mapping Rules for Class Diagram

Josenildo Melo[1], Aêda Sousa[1], Celso Agra[1], Fernanda Alencar[2]

[1]Departamento de Engenharia de Computação, Universidade de Pernambuco, Recife, Brazil
{jasm,amcs,clasf}@ecomp.poli.br
[2]Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife, Brazil
fernandaalenc@gmail.com

Abstract. The phase of requirements gathering of a project is extremely essential because it identifies all the features that the project should have. After this phase, they must be modeled to be better understood. To model solutions, UML (Unified Modeling Language) is one of the most used languages, but it is not developed to capture domain requirements for quality. To capture these requirements, models based on Goal-Oriented Requirements Engineering (GORE) are used, such as i* (iStar). This paper presents a formalization of i* mapping rules for class diagram in the context of Model-Driven Development (MDD), aiming to create more complete class diagram, where quality requirements are captured.

Keywords: Transformation between models, i*, class diagram, Model-Driven Development.

1 Introduction

Companies need to respond quickly to new market demands, building new solutions or performing maintenance on existing systems. So must update your processes and working properly, without neglecting the quality requirements [1]. It is necessary that the end of the requirements specification phase, all stakeholders is acutely aware of the features and system behavior. For this, are proposed and used various models, especially models of Unified Modeling Language (UML).

The UML is efficient to specify "what" a system does and "how" it does something, but it is not to describe the "why" it does [2]. It is not designed to capture the domain requirements (early requirements) [3]. To minimize these problems, came the Goal-Oriented Requirements Engineering (GORE) [4]. In goal-oriented approaches, requirements engineering is responsible for discovering, formulating and analyzing the problem to be solved, as well as conclude because the problem must be solved and who is responsible for solving the problem. [5]. The need to have more precise specifications of requirements that they consider the reasons, motivations and intentions captured by GORE approach led to the initial proposal of models mapping rules i* (goal-oriented) for class diagrams [6] in UML, which subsequently been extended [7].

This time, contributing to a possible automatic transformation between models could be thought. The formalization of transformation rules between these models was initialized in [8] and making it necessary to formalize and test all the rules, to allow automatic transformations between models (i* to class diagrams).

This paper aims to demonstrate a transformation between models in the context of Model Driven Development (MDD) [9], which is obtained through the formalization of mapping rules described above. For this, the present article is organized as follows: Section 2 briefly define the objectives of the research; Section 3 we discuss the scientific contributions; Section 4 provides the conclusions, and Section 5 presents ongoing and future works.

2 Objectives of the research

This work aims to demonstrate a transformation between models in the context of MDD. This will be achieved through the formalization of the guidelines proposed by [6] and extended by [7]. This guidelines was created to map i* into UML class diagram. The objective of this transformation is to keep the consistency between the desired software system and the organization objectives, as well to establish the impact that any change of objectives will be able to cause in the system and vice versa.

3 Scientific contributions

Using templates to design complex systems is standard in traditional engineering disciplines. We cannot imagine the construction of a building, a bridge or a car, without first constructing a variety of designs and simulate them. Models help us understand a complex problem (and possible solutions) through abstraction.

Currently, the Model-Driven Development (MDD) [9] has proved to be a highly reputable trend [10]. In fact, MDD aims to accelerate the development of software by automating the development of products and employing reusable models or abstractions to view the code (or the problem domain). By using the models, or abstractions, we can describe complex concepts more legibly than computer languages do. This improves communication between stakeholders, because models are often easier to understand than the code [11].

The most important contribution of this work is the development of a transformation between models that covers the MDD. This transformation will be responsible for creating the most complete class diagrams, which cover better user requirements.

This transformation will be achieved through the formalization of the guidelines proposed by [6] and extended by [7]. This guidelines are shown in the Table 1. Is not part of the scope of this study to discuss these rules, but the formalization of them.

Table 1. Mapping Guidelines

Number	i*	UML
1.1	Agents, roles or position	Class.

Number	i*	UML
1.2	Relationship ISPART-OF between positions, agents or roles.	Class aggregation.
1.3	Relationship ISA between positions, agents or roles.	Class generalization/specialization.
1.4	Relationship OCCUPIES between an agent and a position.	Class association named OCCUPIES.
1.5	Relationship COVERS between a position and a role.	Class association named COVERS.
1.6	Relationship PLAYS between an agent and a role.	Class association named PLAYS.
2.1	Tasks defined in SD model.	Methods with public visibility.
2.2	Tasks defined in SR model.	Methods with private visibility.
3.1	Resources defined in SD model.	Class if this dependence has the characteristics of an object.
3.1	Resources defined in SD model.	Attribute with private visibility in class that represents the dependee actor if this dependence cannot be characterized as an object
3.2	Resources (sub resources) defined in SR model.	Attribute with private visibility in the class that represents the actor in which the sub resource belongs (if this sub resource cannot be understood as an object).
3.2	Resources (sub resources) defined in SR model.	An independent class, otherwise.
4.1	(Soft)Goals in SD model.	Attribute with public visibility in the class that represents the dependee.
4.2	(Soft)Goals in SR model.	Attribute with visibility public in the class that represents the actor in wich the sub goal belongs.
5	Task Decomposition.	Represented by pre and posconditions (expressed in OCL) of the corresponding pUML operation.
6.1	(Soft)Goals-(Soft)Goals.	The disjunction of the means values implies the end value.
6.2	(Soft)Goal – Task, Resource-Task.	The post-condition of the means task implies the value of end.
6.3	Task – Task.	The disjunction of the post-condition of the means imply the pos-conditions of the end.

The formalization process is shown in Figure 1. The process starts with data input obtained by iStarTool [12] tool. In iStarTool, the i* element is designed and the tool generates a corresponding file XMI. In the second step ("transformation between models"), this XMI file is imported and rules described in ATL language [13] are applied. In the last step, an output model is generated containing the elements of class

diagram generated. This output model is another XMI file, that must be imported by a CASE tool for the class diagram can be viewed.

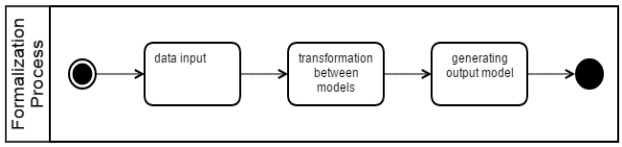

Fig. 1. Formalization process.

4 Conclusions

Requirements elicitation is essential for a system to be developed with all the features and functionality needed, and the application templates can help us visualize the system before its construction begins. Several models exist, the UML is more used. However, UML does not capture all system requirements, indicating "as" a system should be done and not "why" should be done. Among the approaches that care about the needs of the system, stands out i*.

This work presented the formalization of rules mapping i * to class diagram in order to create class diagrams that addressed user requirements more fully.

5 Ongoing and the future work

The objective of this work is to create more complete class diagrams using the MDD features, covering the features i* and based on previously created rules. For this, these rules are being formalized in the ATL language. Furthermore, we are performing a comparison between the ATL language and others three transformation languages (QVT, ETL and MOFScript).

As future work, it is planned to finalize the formalization of the rules, as well as the creation of a tool to automate the whole process. Also is planned the adaptation of XGOOD tool [14]. This tool decides which i* elements must be mapped.

References

1. T. C. Pereira, F. M. R. Alencar, J. R. F. Silva, and J. F. B. Castro, “Requisitos Não-Funcionais em Modelos de Processos de Negócio : Uma Revisão Sistemática,” *Proc. do IX Simpósio Bras. Sist. Informação*, 2013.
2. G. A. A. C. Filho, A. Zisman, and G. Spanoudakis, “A Traceability Approach for i * and UML Models,” in *2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, 2003.
3. F. M. R. Alencar, F. Pedroza, J. F. B. Castro, and R. C. O. Amorim, “New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling,” in *VI WORKSHOP DE ENGENHARIA DE REQUISITOS*, 2003.

4. A. van Lamsweerde, “Goal-oriented requirements engineering: a guided tour,” in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, 2001, pp. 249–262.
5. J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, and F. Alencar, “Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach,” *Requir. Eng.*, vol. 17, no. 4, pp. 259–281, Jun. 2011.
6. F. M. R. de Alencar, “Mapeando a Modelagem Organizacional em Especificações Precisas,” UFPE, 1999.
7. J. F. Castro, John Mylopoulos, F. M. R. Alencar, and G. A. C. Filho, “Integrating Organizational Requirements and Object Oriented Modeling,” in *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, 2001, p. 146–.
8. U. D. E. A. Oliveira, “Uma Metodologia DSDM para Integração de Requisitos Organizacionais e UML no Desenvolvimento de Sistemas de Agentes Utilizando i* (i-Star),” Universidade Federal de Sergipe, 2009.
9. B. Selic, “The pragmatics of model-driven development,” *IEEE Software*, vol. 20, no. 5, pp. 19–25, Sep-2003.
10. J. M. Vara, V. A. Bollati, Á. Jiménez, and E. Marcos, “Dealing with Traceability in the MDD of Model Transformations,” vol. 40, no. 6, pp. 555–583, 2014.
11. E. Yu, “Why Agent-Oriented Requirements Engineering,” in *4th International Workshop on Requirements Engineering: Foundations of Software Quality*, 1998, pp. 15–22.
12. Á. Malta, M. Soares, E. Santos, J. Paes, F. Alencar, and J. Castro, “iStarTool: Modeling requirements using the i; framework,” *CEUR Workshop Proc.*, vol. 766, no. iStar, pp. 163–165, 2011.
13. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, Jun. 2008.
14. F. Pedroza, F. Alencar, J. Castro, F. R. C. Silva, and V. F. A. Santander, “Ferramentas para Suporte do Mapeamento da Modelagem i * para a UML : eXtended GOOD – XGOOD e GOOSE,” *Proc. 7th Work. Requir. Eng. - WER 2004*, pp. 164–175, 2004.