Translation of Requirements Engineering Models

Imad Eddine Saidi[1], Taoufiq Dkaki[1], Nacer Eddine Zarour[2], and Pierre-Jean Charrel[1]

[1] Jean Jaurès University, 5 Allée Antonio Machado, F-31058 Toulouse Cedex 9. France
[2]Ali Mendjli University, B.P.76. Constantine. Algeria
{Imad-Eddine.Saidi,Pierre-Jean.Charrel,Taoufiq.Dkaki}@Irit.fr,
Nasro_Zarour@yahoo.fr

Abstract. The globalization and the rapid development of information and communication technologies encourage organizations to work together. In software development, many works have emerged to support this cooperation using different tools and methodologies. Most of them focus on the design-stage concerns. However, very little works have dealt with cooperation during the early stage of software projects, namely Requirements Engineering (RE), despite the importance of this stage for the failure or the success of software projects. There exist different kinds of approaches to support the RE process in different contexts, based on models such as goal, viewpoint and scenario oriented. Each of these models relies on concepts which differ from one model to another. One of the difficulties for organizations that intend to work together in the upstream phases of software projects is summarized by the following question: What is the most appropriate approach every partner has to adopt? In this paper, we propose a translation process between RE models in order to ensure that organizations with different types of RE backgrounds and methodologies can work together to achieve their objectives while still using their own approach. The translation is performed using a unified meta-model issued from a semantic process of computing similarities between concepts of RE models.

Keywords: Requirements Engineering, Meta-modelling, models translation, similarities.

1 Introduction

In software engineering, many factors can be responsible for the success or the failure of projects. One of the reasons affecting the failure of these projects is the poor definition and management of requirements [1]. Hence, predicting and writing good requirements [2] is a key factor for the success or the failure of software projects. Requirements Engineering (RE) [3] is the discipline which aims at defining, managing and documenting software requirements in upstream phases of software lifecycle.

However, due to present-day globalization [4] of the business world, many organizations should cooperate in order to achieve their objectives. Cooperation is the manner of coordination that is necessary for agreeing on common objectives and for the coordinated achievement of common work among participants [5]. Unfortunately, most of the work that discusses cooperation in software engineering such as [6] and [7] focuses only on the

K. Liu et al. (Eds.): ICISO 2015, IFIP AICT 449, pp. 135–145, 2015.

design stage of software development lifecycle and do not address the RE upstream phase. Very little works have focused on the cooperation in RE. The fact that RE is important should have led to more interest in cooperation in this phase.

Different kinds of solutions have been proposed to support the RE process: goal oriented approaches such as i* [8] deal with actor dependencies, goals and intentions, viewpoint oriented approaches such as PREview [9] deal with the perception of actors, and scenario oriented approaches such as CREWS [10] describe functional behaviors by means of scenarios. This variety of solutions makes cooperation among organizations stakeholders in this phase a difficult activity due to the heterogeneity between these models.

Bendjenna et al. [11] proposed a solution which aims to integrate the three concepts of goal, viewpoint and scenario into one model in a cooperative environment. The work is embodied in the proposal of MAMIE as a new approach that should be used by all organizations stakeholders in order to allow cooperation between them. On the contrary, our work stems from the idea that preserving as far as possible the working environment of the stakeholders involved in organizations which aim to cooperate is more realistic. This leads us to propose a translator between different kinds of existing RE models that follows, in a broader way, the principle of Cares and Franch [12]. These ones have defined a "super meta-model" hosting identified variations of i* and implementing a semantic preservation oriented translation algorithm between these different variations.

Our translator allows organizations that use different kinds of RE models to cooperate continuing to use their usual approaches, without forcing them to spend time, human and financial resources in order to migrate to a unique RE model. The RE translator lies on our so called UREM unified meta-model which is issued from a semantic process of computing similarities between concepts of RE models ([8], [9] and [10]).

In this paper, we present a 3 phases process to translate RE models (Fig. 1). The following section 2 presents the two first phases of the translation process: the unified requirements engineering meta-model UREM, and how correspondences between concepts are built. In section 3, we discuss the third phase of building the RE translator which involves 'how' the translation between heterogeneous RE models is performed. Section 4 presents a case study to evaluate our work. In section 5, we conclude and draw perspectives.

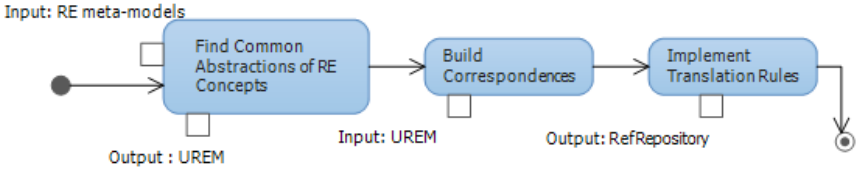

Fig. 1. RE Translator Building Process

2 UREM: Unified Requirements Engineering Meta-model

In this section, we use a unified RE meta-model to create correspondences between meta-models of 3 representative RE approaches: i*, PREview and CREWS. This meta-model represents the common abstraction of those RE meta-models. These correspondences represent the core component of the translator between RE models. The translation between models using these correspondences can be achieved by finding for each concept in a source model the most suitable correspondent concept or collection of concepts in a target model. The resulting so called UREM meta-model is illustrated in Figure 2.

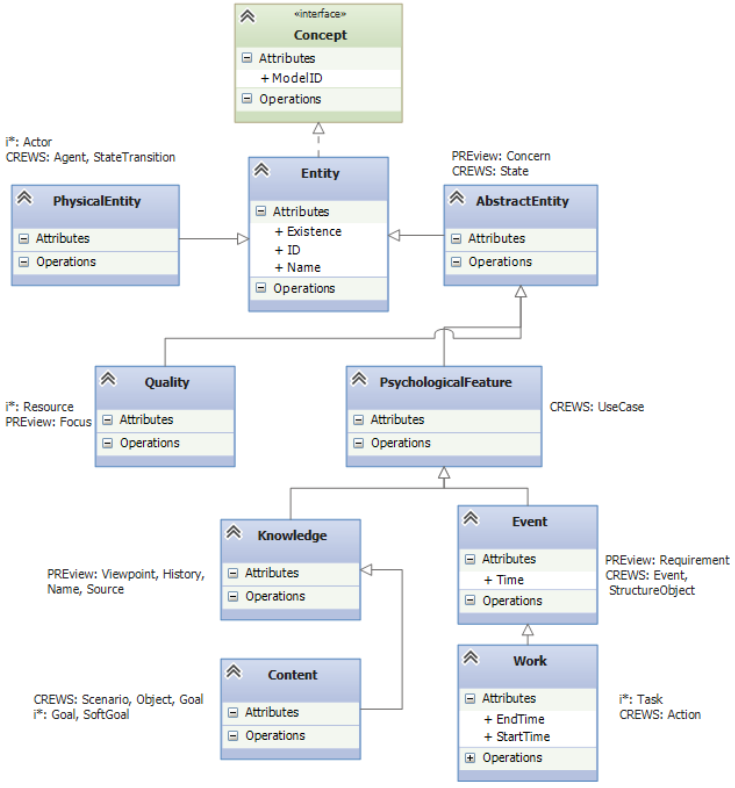

Fig. 2. UREM Meta-model

Each class (abstract concept) in UREM (Fig. 2) represents an abstraction of a set of similar RE concepts. These concepts are labeled beside each abstract concept in Fig 2. Similar concepts are concepts that share some common ground (attributes). In a previous work, we adopted a rigorous semantic process [13] based on the semantics of words which represent RE concepts. This process is performed using WordNet [14] and is composed of several steps starting with the classification of RE concepts into two categories: concepts that can be retrieved directly from WordNet (category 1) and

concepts that cannot (category 2). This categorization leads us to develop an incremental process by applying several algorithms on concepts of category 1 using WordNet: (1) Word Sense Disambiguation (WSD), (2) treat concepts as a tree in WordNet and compute distances by finding the least common parent (hypernym). Afterwards, we compare concepts of the category 2 to the tree that represents concepts of the category 1 and their parents in order to find similarities.

UREM is represented as a tree where abstract concepts are parent nodes (hypernyms) of different RE concepts (nodes). For each concept, we can find the most suitable correspondent (the most similar concept) in a target model by browsing the tree as follows:

- If several paths lead from a given concept to one concept in another model, we use least common hypernyms as described in [13] to find the shortest path between concepts. If the shortest path leads to several concepts in a same target model, we consider all these concepts as a correspondent. For example: Goal concept of i* has the same distance to several concepts in PREview: Viewpoint, History, Name and Source. If a concept has child and parent nodes, we always browse the tree according to the shortest path. Finding paths is a key factor to build correspondences.
- We build sets of correspondences between the three RE models from the results of the previous steps. For example: Goal and SoftGoal concepts in i* have Scenario, Object and Goal target concepts in CREWS, and Viewpoint, History, Name, Source target concepts in PREview. In the same time, Scenario, Object and Goal concepts in CREWS have Viewpoint, History, Name,Source target concepts in PREview. UseCase has also a short path to these concepts. We build the overall correspondence: {Scenario, Object, UseCase, Goal (CREWS), Goal, SoftGoal (i*), Viewpoint, History, Name, Source (PREview)} that is stored and used as a reference to translate these concepts from a model to another.

The resulting sets of correspondences are:

- C1 = {Scenario, Object, UseCase, Goal (CREWS), Goal, SoftGoal (i*), Viewpoint, History, Name, Source (PREview)}
- C2 = {Action, Event, StructureObject (CREWS), Task (i*), Reqirements (PREview)}
- C3 = {Agent, StateTransition, State (CREWS), Actor, Resource (i*), Concern, Focus (PREview)}.

In this section, we have built correspondences relationships among RE concepts. In the next section, we discuss the last phase illustrated in Fig. 1 to build the RE translator related to; how to perform translation between those concepts using the defined correspondences.

3 Translation between Requirements Engineering Models

In this section we describe how to translate RE concepts using correspondences created in the previous section. Figure 3 illustrates the design of the translation process between concepts.

The design illustrated in Fig. 3 is based on the factory design pattern [15] which allows users to translate and create, for a source concept, one or more target concepts without bothering them with the entire specification of these target concepts that they do not need to know, or they do not have the abilities to handle. The translation should be performed through a common interface called IConcept. The user asks ConceptsFactory for translation that needs the use of a referential repository: RefRepository.

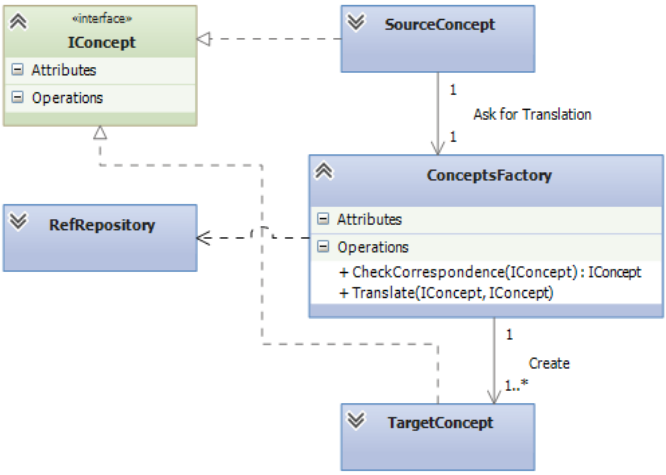

Fig. 3. Design of Translation Process

RefRepository stores all possible correspondences generated in the previous section between concepts of i*, PREview and CREWS models. For each correspondence, there exist translation rules which describe how to translate a source concept to a target model. RefRepository defines two types of translation:

- Automatic translation is used with the first and the second type of translation. This rule operates when the Concepts Factory creates automatically a new instance of a target concept from the source one by checking the most suitable correspondence in RefRepository. Afterward, the factory translates source attributes to target attributes in the new target concept by moving the value of source attributes to a target attributes. The factory uses simple naming conventions to name the new target concepts.
- Semi-Automatic translation is performed after the automatic translation if a part of source concepts cannot be translated correctly to the target model. We perform a translation aided by questions {Which, How or What}. Two lists of elements (concepts and attributes) are created, one for source elements that are not translated correctly to the target model and the other is composed of empty instances which represent target elements in the target model that are not created in the automatic translation. Fig. 4 illustrates these lists. Each list is divided in three sets according to the correspondences that are previously defined. Users can help the RE translator

using their experience by answering several questions on the source elements. Answers to these questions are a guide to find a matching (translation) between source and target elements that are not translated automatically.

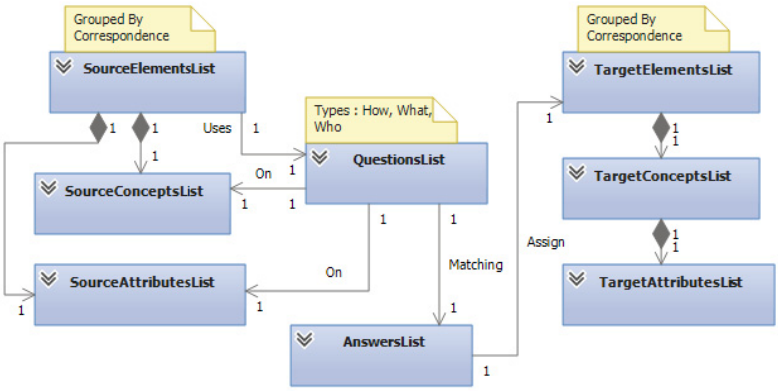

Fig. 4. Structure of Semi-Automatic Translation using Questions

RefRepository (Fig. 5) contains a set of elements. Each element is either a concept or an attribute and it belongs to a model. Each element is included in a correspondence where each one has a set of Naming Conventions and a set of questions that will be used in the semi-automatic translation.

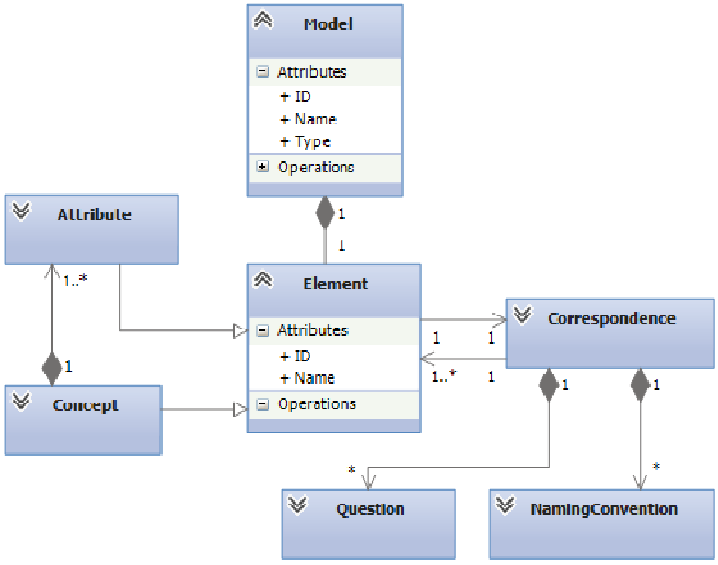

Fig. 5. Class diagram of the referential repository (RefRepository)

4 Case Study: Software Bugs Management

The evaluation of our work can be achieved by applying several case studies in order to verify the soundness of the translation process we propose. In this paper, we present one of these case studies to illustrate the translation process: a requirements specification for a software bugs management system. We uses i*, PREview and CREWS to represent requirements models. For the sake of space we only represent i* model (cf. Annex), the translation between i* and PREview and the results of translation between the three models. We compare source models for each type to the models obtained according to the translation rules defined in the previous section.

The comparison is performed using a three rows and three columns translation confusion matrix [16] to calculate the translation accuracy of our solution. Rows and columns respectively represent source models (actual classes) and target models (predicted classes): i*, PREview and CREWS. Each cell $C_{i,j}$ represents the number of concepts instances which are translated correctly from the source model i into the target model j. Afterward, we compute the accuracy of the translation A_T between each couple of models M_1 and M_2 by applying a simple formula to find the average of translated concepts ratio between any couple of models. Let C_1 and C_2 be respectively the numbers of concepts of M_1 and M_2. Let $C_{1,2}$ and $C_{2,1}$ be the number of translated concepts respectively from M_1 to M_2 and from M_2 to M_1.

$$R_T = \frac{\left(C_{1,2}/C_1 + C_{2,1}/C_2\right)}{2} \quad (1)$$

Any difference between predicted and actual concepts is considered as an error. In our case study, we apply an automatic translation which represents the most important part of translation without any expert intervention, and then we can improve the translation results by using a semi-automatic translation between RE models.

The case study involves a requirements specification for a system which aims to manage and resolve software bugs. A bug is an "Imperfections in software development process that would cause software to fail to meet the desired expectations" [17]. Therefore a bug can be defined as an abnormal behavior or a malfunction of the software system. To monitor these bugs, the use of a bug-tracker is inevitable to eliminate or at least reduce them. This system aims at providing actors with the possibility to report malfunctions, comment them, track the status of the anomaly, notify other actors of the problems encountered, and suggest solutions or opportunities for circumvention.

4.1 Application of Translation between PREview and i*

The translation between RE models of this case study is performed according to translation rules and correspondences C1, C2 and C3 defined in section 2.

To translate the PREview source model to i*, the user of PREview asks the Translator (ConceptsFactory) to translate the different 13 instances of concepts that compose the specification of the case study to target concepts in i*. ConceptsFactory checks for each instance the most suitable correspondence in the target model i*. Table 1 illustrates the results of the translation process grouped by correspondences.

Table 1. Translation from Preview to i*

Source (Preview) Concepts	Target (i*) Concepts	Translation Rule
3 instances of Viewpoint concept with their names: {BugManagementViewpoint, BugReportingViewpoint & BugFixgingViewpoint}	3 instances of Goal concept: {BugsManagementGoal, BugsReportingGoal, BugsFixingGoal}	Automatic Translation using C1.
8 instances of Requirement Concepts: BugReproducibilityDegree-Requirement, BugResolutionPriorityRequirement, BugSeverityDegreeRequirement, BugSummaryRequirement, SuggestSolutionRequirement, TestingRequirement, CommitRequirement, NotificationRequirement}	8 instances of Task Concept: AddBugReproducibilityDegree Task, AddBugResolutionPriorityTask, AddBugSeverityDegreeTask, AddBugSummaryRequirement Task, SuggestSolutionTask, TestSolutionTask, CommitSolutionTask, NotifyPersonsTask}	Automatic Translation using C2
Concern: {Bug Unavailability}	4 actors {ManagerActor, ReporterActor, QATesterActor, EngineerActor	Semi-Automatic translation using C3 and the question: 'Who is responsible for'+ Concern

The overall translation for the 13 instances of PREview is automatically performed to 11 instances in i*, and semi-automatically to 4; i.e. a total of 15 instances if the semi-automatic translation is well performed. Recall that an original source model of i* is composed of 20 instances of concepts. The 5 concepts in i* that are not translated are: Tasks (ReportBugTask, FixBugTask, ManageBugsTask), SoftGoal (Immediate-Reporting), Resource (BugResource). The Source concept (SoftwareSource) is not translated correctly to i* model.

The translation from an i* source model to PREview is performed in the same way with the same correspondences as illustrated in Table 2.

The overall translation of i* model (20 instances) are translated automatically to 11 instances of PREview concepts, and semi-automatically to one instance; that leads to a total of 12 instances if the semi-automatic translation is well performed. Knowing that an original source model of PREview is composed of 13 instances, the Resource and the SoftGoal of i* model are not translated correctly to PREview concepts.

4.2 Evaluation

In the evaluation, we compute the rate of translation successes obtained from the source models (Rows) of i*, PREview and CREWS. Table 3 presents the confusion matrix that summarizes the translation results of the models in the case study.

Proceeding from Table 3, we compute the accuracy of translation A_T of the case study when using automatic translation. We apply the formula (1) between each couple of RE models:

Table 2. Translation from I * to PREview

Source (i*) Concepts	Target (PREVIEW) Concepts	Translation Rule
3 instances of Goal concept mentioned in Table 1	3 instances of Viewpoint concept mentioned in Table 1	Automatic Translation using C1
11 instances of Task: 8 mentioned in Table 1 that represent sub tasks for ReportBugTask FixBugTask, ManageBugsTask.	8 instances of Requirement (mentioned in Table 1).	Automatic Translation using C2.
4 Actors mentioned in Table 1	Concern mentioned in Table 1	Semi-Automatic translation using C3 and the question: "What is the concern of"+ Actor.

Table 3. Automatic translation matrix for software bugs management system

	I*	PREview	CREWS
I* (20[a])		11	20
PREview (13[a])	11		16
CREWS (21[a])	15	11	

a. Numbers of concepts instances presented in the requirements specification

- PREview to/from i*: $R_T = (11 \div 20 + 11 \div 13) \times 0.5 = 70\%$
- CREWS to/from i*: $R_T = (15 \div 20 + 20 \div 21) \times 0.5 = 85\%$
- CREWS to/from PREview: $R_T = (11 \div 13 + 16 \div 21) \times 0.5 = 80\%$

The total average translation accuracy A_T among all models is 78%. We observe the best translation rate is between i* and CREWS. The translation rate can be improved using a semi-automatic translation.

The case study presents a specific part of defined correspondences. The other concepts are not applied in this case study and have to be used in a future case study.

5 Conclusion

This paper presents a solution which allows the translation between different kinds of RE models in order to improve cooperation between stakeholders issued from cooperating companies. The translation is performed using a set of correspondences between the RE models, based on a unified meta-model called UREM. UREM is composed of a set of abstract concepts that represent these correspondences. For a given correspondence, we define translation rules to ensure the translation between concepts from one RE model to another. We present a case study in order to assess the correspondences and the rules that are defined. Unfortunately, we observe that some concepts are not successfully translated. We fix to some extent a part of this problem by adjusting some correspondences such as the concept Actor of i* and Agent of CREWS that can be integrated into the attribute StakeHolder of the PREview Viewpoint concept. Recall that the overall translation accuracy among all models is 78%.

We are currently developing ReqTranslator: it is a web platform which aims to illustrate and apply our solution of RE models translation. The platform proposes several features including translation of different kinds of RE models in addition to ensure the auto-integration of new types of RE models. This integration can be easily achieved due to the structure of the data model proposed for the platform which allows extensibility. Another perspective is to enhance the visualization form tables to graphs which will simplify the representation of requirements. We will also study more complex case studies in order to improve the evaluation of our work.

References

1. McConnel, S.: Code Complete: A Practical Handbook of Software Construction, 2nd edn. (2004), ISBN-13: 079-0145196705, ISBN-10: 0735619670
2. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley (1998), ISBN: 978-0-471-97208-2
3. 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, E-ISBN 978-0-7381-0448-5 (1998)
4. Joshi, M.: International Business. Ox-ford University Press, India (June 22, 2009), ISBN-10: 0195689097, ISBN-13: 978-0195689099
5. Bauknecht, K., Mühlherr, T., Sauter, C., Teufel, S.: Computerunterstützung für die Gruppenarbeit. Addison-Wesley, Bonn (1995)
6. McChesneya, I.R., Gallagherb, S.: Communication and co-ordination practices in software engineering projects. Information and Software Technology Journal 46(7), 473–489 (2004)
7. Altmann, J., C.: Cooperative software development: concepts, model and tools, Technology of Object-Oriented Languages and Systems, TOOLS 30 Proceedings, pages 194-207, (1999)
8. Yu, E.: Modelling Strategic Relationships for Process Reengineering, Phd Thesis, University of Toronto (1995)
9. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. Wiley (1997), ISBN: 978-0-471-97444-4
10. Sutcliffe, A.G., Maiden, N.A.M., Minocha, S., Manuel, D.: Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering 24(12) (1998)
11. Bendjenna, H., Zarour, N., Charrel, P.J.: Eliciting Requirements for an Inter-company Cooperative Information System. J. Systems and IT 12(4), 305–333 (2010)
12. Cares, C., Franch, X.: A Metamodelling Approach for *i** Model Translations. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 337–351. Springer, Heidelberg (2011)
13. Saidi, I., Dkaki, E., Zarour, T., Charrel, N.E.: P.,J.: Semantic Based Process towards Unification of different Requirements Engineering Approaches. In: International Conference on Knowledge Management and Information Sharing, Vilamoura, Portugal. SciTePress (2013)
14. Miller, G., WordNet, A.: A Lexical Database for English. CACM 38(1) (1995)
15. Johnson, R., Vlissides, J., Helm, R., Gamma, E.: Design Patterns: Elements of Reusable Object-Oriented Software (1994), ISBN-13: 078-5342633610, ISBN-10:0201633612
16. Stehman, S.V.: Selecting and interpreting measures of thematic classification accuracy. Remote Sensing of Environment 62(1), 77–89 (1997)
17. Kumaresh, S., Baskaran, R.: Defect Analysis and Prevention for Software Process Quality Improvement. International Journal of Computer Applications 8(7), 875–887 (2010)

Annex

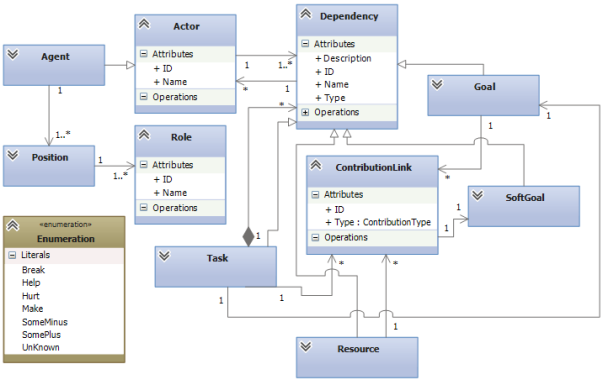

Fig. 6. i* meta-model

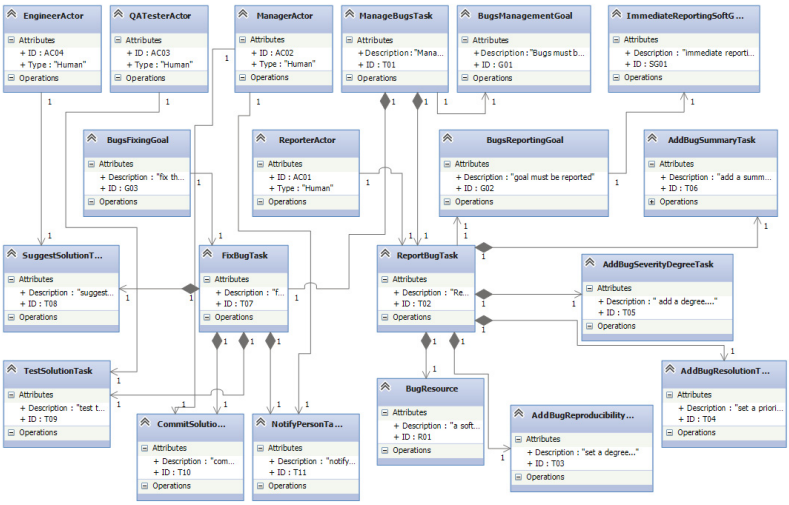

Fig. 7. i* Specification for Software Bugs Management System