American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)
ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

Insights on How to Enhance the Detection of Modeling Errors by *iStar* Novice Learners

Hajer Mejri*

Interdisciplinary Graduate School of Science and Technology, Shinshu University, Wakasato, 4 - 17 - 1, Nagano City, 380-8553, Japan
Email: mejri85@gmail.com

Abstract

When teaching a new paradigm which involves practical training to a large group of students, it often becomes time-consuming and impractical for a single instructor to give advice on an individual basis on how to correct errors being made and the need for computer-aided assistants arises. In this work, we focus on the i* (iStar) framework which is being used to teach requirements engineering and modeling techniques to undergraduate and graduate students in the classroom. We proposed and developed an online tool for automating the work of checking the design constructs used in i* diagrams so that novice learners could detect and correct errors on their own without the assistance of a human expert nearby. Although the tool was useful in showing novice learners how to edit their models to make them free of syntax errors, there were a number of situations in which they could not recognize the semantical design flaws and defects of a model using the feedback from the tool. In this paper, we give examples of the errors we observed and recommend a new tool which will automatically generate a human understandable textual annotation of the main model elements and the relationships connecting them to assist beginners as well as non-technical stakeholders involved in the requirement decisions of a system with detecting simple misrepresentations of information of this type that need to be rectified in the model.

Keywords: insight; *iStar* modeling; novice learner; table of contents; tool.

---

* Corresponding author.

1. Introduction

Since its development in 1995, the *i** (iStar) framework and modeling language [1], a goal oriented requirements engineering (RE) graphical notation, has continued to evolve into a valuable tool for teaching requirements modelling techniques in a number of graduate and undergraduate engineering curricula worldwide [2, 3, 4, 5, 6]. Although i* uses only a small set of graphical constructs to represent the actors involved in a system, their dependency relationships, and the breakdown of their goal oriented requirements, some practical training time is necessary for beginners to grasp how to use the various constructs and rules to represent this information. A number of free modeling tools and editors [7, 8, 9] have been developed to assist with drawing models on a PC screen, but none contained the functions for checking the full set of i* design rules so novice learners using these tools could construct poor quality models without realizing it, which is not desirable in a classroom setting. We addressed this problem in our previous work [10, 11] and developed the *i*Check* tool which works in tandem with existing tools to return a report-like feedback to the user which includes clear textual error messages consolidated with GIF animations showing tips on how to modify requirements in a model step by step to help students effectively work toward a solution on their own without having to rely on feedback from a nearby human expert. However, in continued observations of novice learners using tool assistants to clean up errors in their models, we found that even in the early stages of training, syntax checking support is not enough to train good modeling habits in beginners. Despite having produced a model free of syntactical errors, students sometimes failed to recognize simple semantical design flaws and defects of the models which presents a new challenge in the teaching-learning operation.

In this work, we examine i* model examples which novice learners could judge to be free of syntax errors using design rule checking feedback, but could not recognize as having components that misrepresented information in the system requirements specification and suggest how an automatic generation of a textual summary of the model contents could be used to help beginners in this next step of validation.

2. Overview of the *i** framework

The *i** framework [1] centers on a graphical modeling language for depicting the dependency relationships and goal-oriented requirements of actors in a system and the generated model is used in early stage software systems development to confirm that the requirements of a system have been documented correctly and completely before any programming work begins.

It is based on two types of diagrams: the strategic dependency (SD) model and the strategic rationale (SR) model. The former describes how actors in an organizational setting depend on one another by showing what each actor will need provided, achieved, or fulfilled by another actor (Figure 1).

The SR diagram extends the SD diagram by documenting the goals or internal intentions of each actor and how they can be reasoned or achieved (Figure 2).

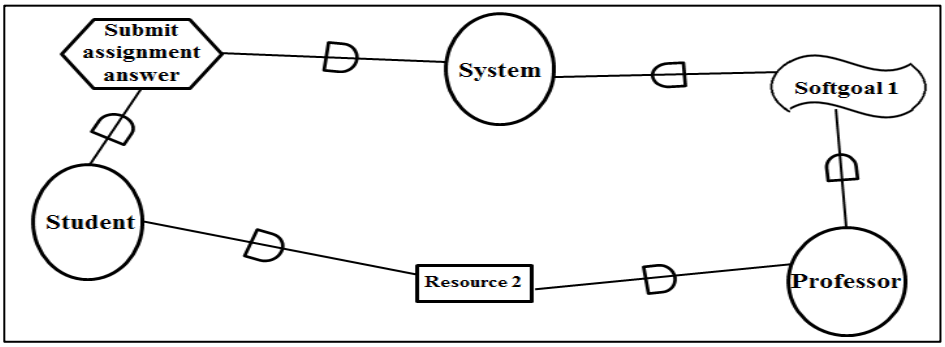

Figure 1: Example of an i* strategic dependency (SD) model showing the actors (circles) in a system and their dependency relationships with the one another.

If there is more than one way to fulfill a goal, the strategic rationale will state the different available alternatives as well as specify what should be provided in order to accomplish the tasks to achieve the goal. Thus, the SR

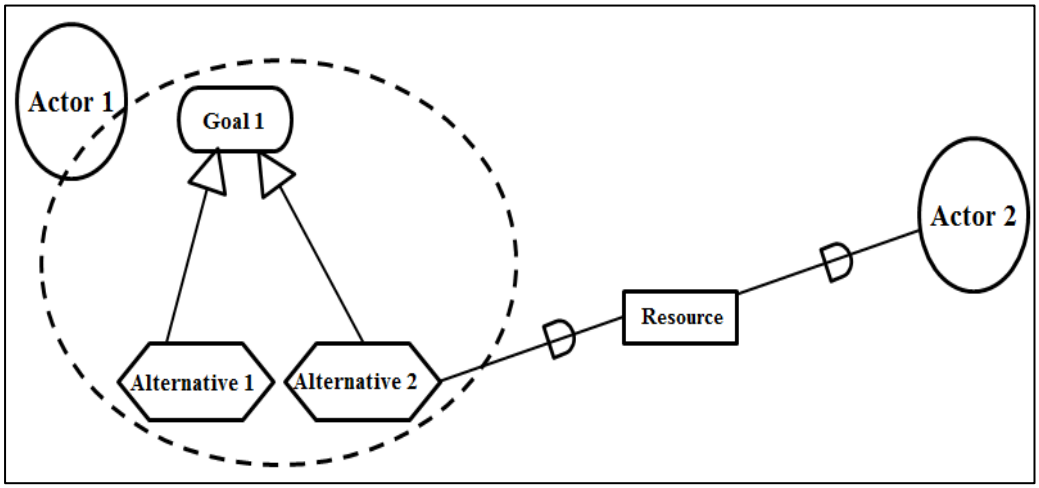

Figure 2: Example of an i* strategic rationale (SR) diagram which extends the SD diagram by depicting the internal intentions of an actor and how they can be refined using means-end (arrow) links to list the different available alternatives to achieve the high level goal.

diagram contains additional detailed graphics which are only associated with this level of modeling detail.

3. Error checking of i* diagrams

The basic i* SD and SR diagram construction rules appear in a dedicated online Wiki page [12] available to share the best practices and i* approach guidelines. A number of free modeling tools and editors allow users to sketch an i* diagram on a PC screen and offer the exportation of diagram data using an XML-based file format called istarML [13, 14]. Since none of the tools surveyed provided a full set of syntax checking features for the design rules presented in the i* Wiki page, we developed an online tool called *i*Check* [10, 11] which uses istarML data exported by available design tools and editors as input and returns individualized feedback on the

syntactical quality of the constructed model. The *i*Check* tool encourages students to perform self-correcting of the design rule violations detected and produces suggestions on how to rectify the errors with textual and GIF animation guidelines.

In continued observations of novice learners constructing and checking i* models, we found that although the *i*Check* tool feedback was effective in guiding beginners on how to correct basic syntax errors, several situations occurred in which users missed errors concerned with the informational contents of diagrams, i.e., the novices grasped the mechanics for writing requirements, but did not realize that they had misrepresented or added unnecessary information to the requirements model. After practicing how to gather requirements from a natural language description or interview and constructing an i* model of the information, the beginners needed a simple way to confirm that the information contained in the model matched their understanding of the requirements, with no omissions or unnecessary additions. In the following, we present two examples that show how an automatic summary of the contents of an i* diagram organized in an outline form would be useful for beginners who need to revisit the original descriptions of the software requirements of a system and validate that their constructed models are error-free in a semantical sense.

3.1 Misrepresentation of intended requirements

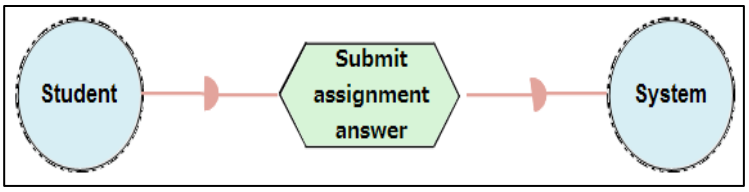

Figure 3: Example of wrong direction of the dependency relationship. There is no syntactical error in the application of the design rule, but the intended meaning ("The system depends on the student to submit an assignment answer.") is depicted in the reverse direction.

In the SD diagram for an i* model, the main job is to identify all of the actors in a system and document what each will need provided or fulfilled by others such as in Figure 1. For the beginner, the SD construction rules are not difficult to apply, but in the course of model construction, humans sometimes introduce errors.

In Figure 3, for example, a requirement stating, "The system depends on the student to submit an assignment answer" is written in the reverse direction.

The format of the dependency specification is not in error so there will be no warning by i* syntax checkers, but the meaning is incorrect.

Although this type of error could be caught with careful review, as a model grows in size, it will be easier for a human to mistakenly overlook such an error and propagate it to later stages of development. This can cause misunderstanding issues and costly corrections that the requirements engineer is particularly employed to avoid.

If this misrepresentation is not corrected at the SD diagram level, the next step of specifying and refining the goals of each actor, i.e., the SR diagram construction, will proceed and it will become more difficult to spot the error. If the SD level misrepresentation of information shown in Figure 3 is left uncorrected and the modelling process continues to formulate the expanded SR model in Figure 4(a), the chances of a human catching the error from doing just a manual check of the information will not be high. Even an examination of the human-readable istarML (XML-format) data of the model (Figure 4(b)) will not easily detect the problem since a human must often sift through and jump to different locations of the file just to recognize one piece of information, in this case that a student and the system to be developed have a dependency relationship in which the system depends on the student to take action on submitting assignment answers.

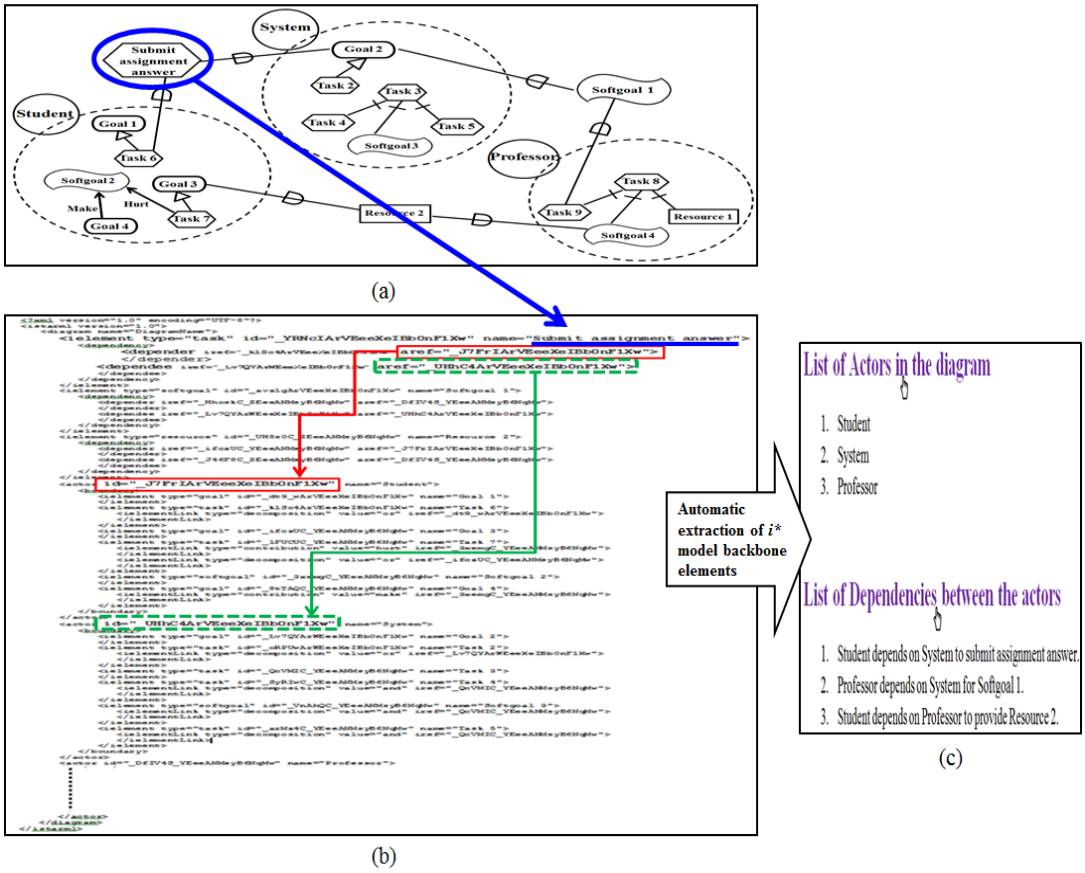

Figure 4: Checking the meaning of each piece of information in a requirements model can become tedious for a human when (a) the number of components in a model grows and (b) tracing the human-readable istarML data of the model requires jumping to different areas of the file to compile one piece of information. (c) An automatic summary generator extracting the i* model backbone elements from the XML data and listing the basic requirements information in short natural language sentences could help novices more easily find errors in the model contents.

In our observations of i* model construction and checking work by novice users, it became apparent that a simple summarization of the backbone elements of an i* model could be helpful in reducing the time and effort of checking the meaning of the information expressed. We propose in this work an automatic tool like *i*Check*

which can work online and take as input istarML data of models generated from various design tools and editors to produce short natural language descriptions of the i* model backbone elements in a simple table of contents (TOC) style (Figure 4(c)). In this way, beginners to the modeling domain can have a more human-friendly document to check for discrepancies in the model description.

3.2 Misinterpretation of design constructs

The second example of novice users having problems recognizing how to specify requirements in i* occurred in the goal refinement work of SR diagram construction. In this situation, users learned that the Means-End link is the design construct for presenting the alternative ways of achieving a goal. The examples in the tutorial lesson always showed multiple alternatives and the guidelines did not explicitly define the cardinality (1..n) of the construct so it was not completely clear to some students that it is possible to have only one means for achieving a goal as shown in the example of Figure 5.

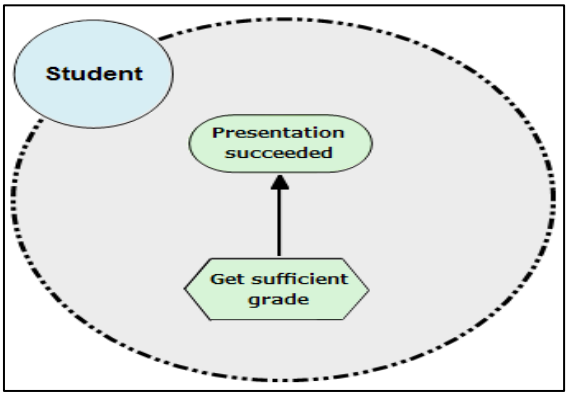

Figure 5: Example of goal refinement showing that a single means is the only (sufficient) way to achieve the goal. Beginners forcing multiple alternatives to be listed for each goal will see no syntax errors in their diagrams, but may be introducing irrelevant components without realizing it.

This resulted in some students forcing model edits to create a situation in which each goal had at least two alternatives listed, even if such information was not contained in any of the requirements of the system. As with the example in the previous section, a syntax check of the model will declare that it is free of design errors, but in fact irrelevant components exist and some human judgment will be necessary to determine whether or not the information contained in a model is enough or too much. Such misinterpretations of the design rules in a new modeling paradigm occur frequently with new users so again support beyond simple syntax checking is necessary to detect this type of error. The automatic generation of textual summaries of i* models proposed in this work can be a useful tool for other persons reviewing the models constructed by beginners to check their meanings.

4. Conclusion

In general, when learning a new modeling paradigm or approach, it is common for novice learners to overlook

errors in the syntax and semantics of their work until they get enough practice and sometimes coaching advice. If it is not practical for beginners to have on-hand experts or instructors to review their work, which is often the case in large sized classes, we need some automated tools to help them recognize bad practices on their own as much as possible before they become bad modeling habits which are hard to break. In this paper, we presented examples of situations in which novice learners were able to judge whether or not an i* requirements models contained design syntax errors and edit models to remove these errors using feedback from the online *i*Check* tool of our previous work. A closer examination of the answers by students showed that although the syntax errors could be detected and corrected, misrepresentations of information in the original system requirements description or the addition of irrelevant information were overlooked. To address this problem, we recommend a tool assistant which can be used in combination with our *i*Check* tool which can use an istarML (XML representation of an i* diagram) file as input to generate a textual summary of the backbone elements of a model in short natural language sentences organized in a table of contents style. We found a work on the development of an Eclipse based tool [15] which supports a proposed programming-like textual syntax language for Goal-oriented Requirement Language (GRL), an *i** variant, which allows for both the creation and modification of large models. However, a simple translation into natural language, rather than programming language, will suffice for enhancing the error detection ability of novices in the scenarios presented in the examples of this paper. The most important elements of the i* SD diagram include a list of the actors in a system and the dependencies between them, i.e., what each actor needs provided or accomplished by the other actors, expressed in short statements of the form "A depends on B for…". Such a summary can be a useful, complementary document to the i* diagrams for beginners to quickly review their models for semantical errors such as unintentional reversals of meaning, omissions of information, inclusions of irrelevant information, etc. As the SR diagram components add complexity to an i* model, the automatically generated textual annotations of the contents by the tool assistant proposed in this work will be even more useful. The summaries can be used not only by novice learners to check the meanings of model constructs, but also by other persons such as non-technical stakeholders who are involved in discussions and decisions concerning the requirements of a system and need to review the model contents although they are not specialists in the modeling paradigm. An important thing to consider in our future work is the release of a new version of i* (iStar 2.0) [16] in 2016 which introduces some modifications and simplifications to the original construct definitions of the initial version. In this respect, an effort, which consists in suggesting new iStarML 2.0 elements, to cope with and consider the evolution of the i* language specification has been proposed in [17]. So, one issue to deal with when proceeding with summary generator tool development is re-doing a careful study of the available tools to figure out which of them have upgraded and updated their exportation functions to adjust to the new iStarML version and subsequently decide the treatment of the newer and older versions.

References

[1] E. Yu. "Modeling Strategic Relationships for Process Reengineering." PhD thesis, University of Toronto, Canada, 1995.

[2] F. Dalpiaz. "Teaching Goal Modeling in Undergraduate Education," in Proc. iStarT, 2015, pp.1-6.

[3] J. P. Carvallo. “Teaching Information Systems: an i*-based approach,” in Proc. iStarT, 2015, pp.7-12.

[4] E. O. Svee, J. Zdravkovic. “iStar Instruction in Mixed Student Cohort Environments,” in Proc. iStarT, 2015, pp.19-24.

[5] Z. Babar, S. Nalchigar, L. Lessard, J. Horkoff and E. Yu. “Instructional Experiences with Modeling and Analysis using the i* Framework,” in Proc. iStarT, 2015, pp.31-36.

[6] E. Paja, J. Horkoff and J. Mylopoulos. “The importance of teaching goal-oriented analysis techniques: an experience report,” in Proc. iStarT, 2015, pp.37-42.

[7] HiME: Hierarchical i-star Modelling Editor. Internet : http://www.upc.edu/gessi/istar/tools/hime/index.html, [Apr. 01, 2015].

[8] OpenOME, an open-source requirements engineering tool. Internet: http://www.cs.toronto.edu/km/openome, 2011 [Feb. 14, 2015].

[9] Á. Malta, M. Soares, E. Santos , J. Paes , F. Alencar and J. Castro. “iStarTool: Modeling requirements using the i* Framework,” in Proc. iStar, 2011, pp.163-165.

[10] H. Mejri and P. N. Kawamoto. “i*Check: A Web-based Tool Assistant for Detecting Design Errors in i* Model Data,” in Proc. ISERD, 2015, pp.93-96.

[11] H. Mejri and P. N. Kawamoto. “Improving Feedback to Novice Learners on Constructing i* Requirements Diagrams,” in Proc. WCSE, 2016, pp.603-606.

[12] iStar Wiki Guide. Internet: http://istar.rwth-aachen.de/tiki-index.php?page=i%2A+Guide, Jul. 18, 2011 [May. 07, 2017].

[13] C. Cares and X. Franch, A.Perini and Angelo Susi. “iStarML: An XML-based Model Interchange Format for i*,” in Proc. iStar, 2008, pp.13-16.

[14] C. Cares and X. Franch. “iStarML: Principles and Implications,” in Proc. iStar, 2011, pp.8-13.

[15] V. Abdelzad, D. Amyot, S. A. Alwidian and T. C. Lethbridge. “A Textual Syntax with Tool Support for the Goal-oriented Requirement Language,” in Proc. iStar, 2015, pp.61-66.

[16] iStar 2.0 Language Guide. Internet: https://sites.google.com/site/istarlanguage/home, Jun. 03, 2016 [Oct. 07, 2016].

[17] C. Cares, L. Lopez. “Towards iStarML 2.0: Closing Gaps from Evolved Requirements,” in Proc. iStar16, 2016, pp.91-96.