A Flexible Approach for Validating *i** Models

Ralf Laue, Arian Storch

Chair of Applied Telematics / e-Business, University of Leipzig, Germany
laue@ebus.informatik.uni-leipzig.de

Abstract. In this article, we present a flexible approach to verify the syntactical correctness of i^* models. We translate the information that is included in an i^* model into a set of Prolog facts. Logical reasoning is applied for finding problems in a model.
Our validation method has been implemented in the *openOME* modelling tool. By using our validation add-on, modellers get feedback about problems and possible improvements of the model.

1 Introduction

Based on experiences from teaching the i^* framework to students it has been reported that beginners often misunderstand the concept of some elements of the i^* modelling language [1]. They would profit from a modelling tool that can not only locate problems resulting from such misunderstandings, but also gives feedback on correcting the model.

Furthermore, the i^* framework is open to extensions and adaptions. For this reason, a modelling tool should support various modelling styles. It should be able to check whether a model conforms to a given style or to modelling conventions that exist in an organisation.

2 Objectives of the Research

The aim of our research is to provide a mechanism for validating i^* models that has the following properties:

- The modeller gets an immediate feedback about possible problems.
- If a problem is detected in the model, the modeller has the possibility to learn about the reasons for the problem and about possibilities to model the intended meaning of the model in a correct way.
- Own validation rules can easily be added, for example based on company-wide style guidelines.
- The user is able to select the rules that should be applied in the validation.
- The validation procedure is flexible enough to allow an analysis of the textual model element labels.

3 Scientific Contributions

A modelling tool should assist its users when they have to create models that adhere to the restrictions of the modelling language. First and foremost, the model has to conform to the metamodel of the language. For the language i^*, several publications have suggested such a metamodel [2]. By enforcing metamodel conformance, an i^* modelling tool can prevent syntactical modelling errors like an actor being modelled inside another actor. However, metamodel conformance does not yet mean that a model is correct with respect to the language definition. There are additional semantic constraints such as "There should not be a cycle made from actor association links".

For checking this type of constraints, the use of the Object Constraint Language (OCL) has been suggested in [3]. When we integrated validation support into the i^* modelling tool *OpenOME*, we followed another approach that is based on logic programming: First, we translate the information that is contained in a model into a set of Prolog facts. Afterwards, logical reasoning is used for locating problems. For translating an i^* model into the corresponding Prolog facts, we use a simple XSLT transformation. An i^* model that needs to be validated is translated automatically into a set of Prolog facts by this XSLT transformation.

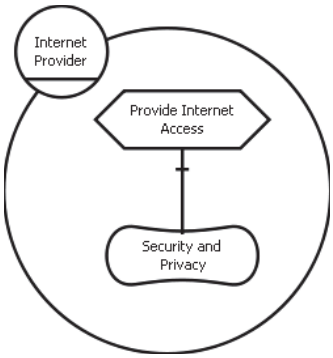

Fig. 1. the i^* model...

```
role('_nKlm').
elementname('_nKlm', 'Internet Provider').
task('_LX3l').
elementname('_LX3l', 'Provide Internet Access').
contains('_nKlm', '_LX3l0').
softgoal('_xdXW').
elementname('_vgGY', 'Security and Privacy').
contains('_nKlm', '_vgGY').
anddecomposition('_LX3l0', '_xdXW').
shape('_nKlm', 2005, 822, 1096, 924).
shape('_LX3l', 190, 423, NaN, NaN).
shape('_xdXW', 1730, 1356, NaN, NaN).
```

Fig. 2. ...and the corresponding Prolog facts

Fig. 1 and 2 show an i^* model fragment and the corresponding Prolog facts. For the sake of readability, we have shortened the unique identifiers that have been associated to each model element by Eclipse's XMI serialisation.

Once the information from the i^* model is available as Prolog facts, it is very simple to locate problems. For example, the code for searching a goal that is wrongly connected to another goal using a means-ends link can be found by the query `goal(G1),me(G1,G2),goal(G2).` In general, means-ends links used wrongly (i.e. not linking a task to a goal) can be identified by the query `me(E,Partner),(not(task(E));not(goal(Partner)))`. It is also no problem

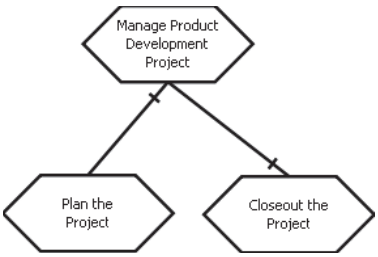

Fig. 3. Syntactical correct, but the layout of the tasks is misleading (source: i^* wiki modelling guidelines)

to detect issues like the mentioned problem of cycles within actor dependency links.

Note that the size of position of the modelling elements are available as Prolog facts as well, allowing reasoning about the layout of the model. This allows to search for problems like the one depicted in Fig. 3. While the model is syntactically correct, the direction of one of the task decomposition links should be changed such that a task that is decomposed into a sub-task is always located above the sub-task. The possibility to deal with this kind of layout problems is an advantage of our approach compared to the OCL-based method described in [3]. Another advantage is that by exporting the model as Prolog facts, we have access to a variety of methods that can analyse the textual labels of model elements. This can be used for finding violations of naming conventions. In our current implementation, we use a heuristic method that analyses the labels and tries to detect two common types of errors: goals that have wrongly been modelled as tasks and softgoals that have been wrongly modelled as goals.

We have included all rules about semantic constraints that can be found in the i^* wiki (http://istar.rwth-aachen.de) as well as some other validation rules described in the literature [3–5] or identified by our own analysis. It is possible to configure the subset of rules to be applied, which allows to use different styles for i^* models. For all i^* models validated so far (including rather large models with more than 100 elements), our validation mechanism delivered a result within one second. In order to prevent performance problems due to running the validation in background, we start the validation mechanism by executing it on demand by selecting a menu item. It should, however, be possible to use the validation mechanism as a background process without affecting the performance (as reported in [6] for a similar case).

Fig. 4 shows how detected problems are reported by the modelling tool: Each problem is shown as error or warning in the "Problems View" of the modelling tool. Also, a visual marker is added to the model element for which the problem has been found. If the modeller needs more information about the detected problem, it is possible to look up additional information about each problem class on a web site. We have already established links to the various guidelines available on the i^* wiki. A novice user can quickly learn to draw correct i^* models

by following the cycle "Model – Validate – Learn about the reported errors – Correct".

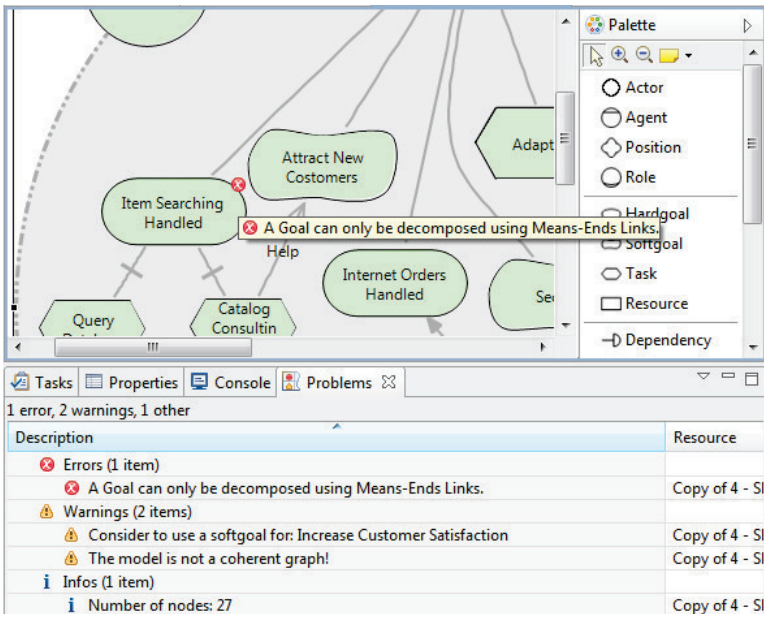

Fig. 4. Errors, Warnings and Information shown in *openOME*

4 Conclusions

The approach described in this paper has been implemented as a plugin into the Eclipse-based tool *openOME*. It is flexible enough to be used with every model editor that is based on Eclipse EMF/GMF. Previously, we have implemented the same validation mechanism into the business process model editor *bflow* Toolbox*. In a controlled experiment [7], we made the observation that providing an immediate feedback about modelling errors had a significant influence on model quality: The group provided with validation support made 6 errors in 7 models. For the control group which had to solve the same modelling task using the same tool without validation support we counted 24 errors in 6 models [8].

5 Ongoing and Future Work

We are interested in extending the approach described in this paper by analysing the texts used in the models more deeply. This would make it possible to check

for style rules such as "The infinitive of a verb together with an object has to be used for describing a task." Other interesting challenges would be to integrate the ideas of the Goal Clarification Method described in [9] and to develop recommendations for correcting a erroneous model.

Developers who are interested in using or improving our plug-in are invited to do so. More information about the plugins can be found at the Eclipse Modeling Toolbox project site `http://sourceforge.net/projects/eclipsemodeling`.

References

1. Horkoff, J., Elahi, G., Abdulhadi, S., Yu, E.: Reflective analysis of the syntax and semantics of the i* framework. In: Advances in Conceptual Modeling, Challenges and Opportunities. Volume 5232 of Lecture Notes in Computer Science. Springer (2008) 249–260
2. Cares, C., Franch, X., Lopez, L., Marco, J.: Definition and uses of the i* metamodel. In: Proceedings of the 4th International i* Workshop, Hammamet, Tunisia. Volume 586 of CEUR Workshop Proceedings., CEUR-WS.org (2010) 20–25
3. Amyot, D., Yan, J.B.: Flexible verification of user-defined semantic constraints in modelling tools. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Eesearch: Meeting of Minds. (2008) 7:81–7:95
4. Amyot, D., Horkoff, J., Gross, D., Mussbacher, G.: A lightweight GRL profile for i* modeling. In: Advances in Conceptual Modeling - Challenging Perspectives, ER 2009 Workshops. Volume 5833 of Lecture Notes in Computer Science., Springer (2009) 254–264
5. de Pádua Albuquerque Oliveira, A., do Prado Leite, J.C.S., Cysneiros, L.M.: Using i* meta modeling for verifying i* models. In: Proceedings of the 4th International i* Workshop, Hammamet, Tunisia. Volume 586 of CEUR Workshop Proceedings., CEUR-WS.org (2010) 76–80
6. Blanc, X., Mounier, I., Mougenot, A., Mens, T.: Detecting model inconsistency through operation-based model construction. In: ICSE '08: Proceedings of the 30th International Conference on Software engineering, New York, USA, ACM (2008) 511–520
7. Laue, R., Kühne, S., Gadatsch, A.: Evaluating the Effect of Feedback on Syntactic Errors for Novice Modellers. In: EPK 2009, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. CEUR Workshop Proceedings (2009)
8. Gruhn, V., Laue, R.: A heuristic method for detecting problems in business process models. Business Process Management Journal **16** (2010) 806–821
9. Jureta, I., Faulkner, S.: Clarifying goal models. In: Challenges in Conceptual Modelling. Tutorials, posters, panels and industrial contributions at the 26th International Conference on Conceptual Modeling - ER 2007. Volume 83 of CRPIT., Australian Computer Society (2007) 139–144