

Evaluation of iStar 2.0 models using Linear Programming

Inmaculada Ayala¹, Mercedes Amor¹ and Lidia Fuentes¹

¹Universidad de Málaga, ITIS Software, Málaga, Spain

Abstract

Goal models are an effective mechanism for elicitation and analysis in early Requirements Engineering, improving communication with stakeholders. However, in real scenarios, goal models become a complex network of actors and evaluating them can be difficult. Some tools assess the level of achievement of actors and intentional elements, which alleviate this limitation. However, using tools to evaluate models has some limitations, like the difficulty of integrating goal models with other models, the training required to use these tools, and the impossibility of assessing models when tools become outdated. In this work, we propose a formalization of iStar 2.0 models in the form of linear constraints that makes it possible to evaluate and optimize models using linear programming. This formalization allows the evaluation of goal models on existing tools and facilitates integration with other approaches.

Keywords

iStar 2.0, Linear Programming, Goal Model

1. Introduction

Goal models are an effective mechanism for the elicitation and analysis in early Requirements Engineering (RE) [1]. This kind of model improves communication with stakeholders as they can express their expectations about the system that they want to achieve. Some of the most popular languages for representing goal models are *i** [2] (and its evolution iStar 2.0 [3]), GRL [4] and Tropos [5]. These languages share many features like entities' type (i.e., all of them have some agents and goals), interpretations (i.e., the meaning of the entities) or how to proceed for model evaluation.

However, in real scenarios, goal models become a complex network of actors that include intentional elements linked between them. So, providing an interpretation of the model can be difficult or even impossible. There are some tools that aid in assessing the level of achievement of actors and intentional elements through an analysis performed by different algorithms. There are mainly three types of analysis that we can perform the backward analysis, the forward analysis [1] and the optimisation analysis [6]. The forward analysis provides information about the level of achievement of intentional elements when the values of intentional elements are propagated from top-level intentional elements (i.e., the roots of the model) to bottom-level intentional entities (i.e., the leaves) through links. In a backward analysis, satisfaction values


iStar'22: The 15th International i Workshop, October 17th, 2022, Hyderabad, India*

✉ ayala@lcc.uma.es (I. Ayala); pinilla@lcc.uma.es (M. Amor); lff@lcc.uma.es (L. Fuentes)

🆔 0000-0002-5119-3469 (I. Ayala); 0000-0001-7190-0581 (M. Amor); 0000-0002-5677-7156 (L. Fuentes)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

are propagated on the contrary from leaves to roots. The third type of analysis of the level of achievement is by optimising goal models, which intends to assign values to intentional elements that maximise the model's satisfaction level.

The evaluation based on frameworks presents some issues that limit the use of goal models in practice. Firstly, existing frameworks strongly depend on specific tools that belong to academia. So, their maintenance is discontinued and they frequently become outdated; therefore, their use for evaluating goal models is rather inadvisable. In addition, these tools are usually closed systems, so integrating goal models with other approaches is almost impossible. The resolution algorithms depend on the specific implementation of the models, so it is necessary to adapt them to similar systems. Another important aspect is the training required to use these tools.

Recently, it has been proposed the formalisation of goal models using arithmetic semantics [7], linear constraints [8] and constraint satisfaction problems [6, 9]. These formalisations allow the evaluation of the models on already existing tools and facilitate the integration of goal models in other approaches [10, 8]. In this work, we propose a formalisation of iStar 2.0 models in the form of integer linear constraints that makes it possible to evaluate and optimise models using integer linear programming (ILP). ILP [11] is a field of mathematical programming consisting of various techniques and algorithms for solving particular mathematical models. These models are composed of decision variables, and the goal is to find values that maximise or minimise an objective function. This objective function is subject to a set of inequality and equality constraints. The program is linear when constraints and the objective function are linear expressions. ILP is applied when decision variables are constrained to take non-negative integer values. We extend the work of Noorian et al. [8], which does not consider inter-agent dependencies and the level of satisfaction values of the intentional elements. IStar 2.0 has an online tool pIStar [12] that facilitates the use of the standard. In addition, various general-purpose tools like Matlab, R or Excel can resolve ILP problems.

This work is structured as follows: Section 2 presents some related work; Section 3 introduces the formalization of the goal model in the form of ILP; Section 4 shows our proof of concept tool; and the paper concludes with some conclusions and future work.

2. Related work

The work presented here is related to many contributions that study how to analyse goal models using alternative representations.

GRL models use the GRL satisfaction analysis [13], a family of algorithms propagating satisfaction values in goal models. Internally, these algorithms work with mathematical expressions of goal models. The work [7] proposes arithmetic semantics based on the GRL standard. Models developed in jUCMNav are translated to SimPy. From this representation, it is possible to generate programs in Matlab, Java, JavaScript, C, C++ or R. This utility is implemented as a plugin in jUCMNav. Continuing with GRL, the work presented in [6] presents a formalisation of GRL models in the form of a Constraint Satisfaction Problem that permits the optimisation of goal models as we do. The approach is integrated into the jUCMNav tool as well.

The work in [9] addresses the multi-objective optimisation of KAOS models using constraint satisfaction. The proposal deals with Pareto-optimal solutions and uses Search-Based Software

Engineering. Multi-objective optimisation of goal models is also approached in [14] in the context of the Constrained Goal Model. Noorian et al. [8] propose a formalisation of goal model refinements as a set of linear constraints. The approach is integrated into a system that facilitates the configuration of products in product lines taking into account high-level stakeholder goals. The approach in [15] uses RELAX goal models to generate utility functions. These functions are used to monitor requirements at runtime.

3. Mapping iStar 2.0 model to ILP

The iStar 2.0 language does not specify how users have to evaluate models. Indeed, for the case of i* (the ancestor of iStar 2.0), frameworks take guidelines from other goal modelling languages like GRL for the evaluation [1]. Our idea is to provide a method to optimise the goal model and compare different model configurations. Our formalisation does not cover all the elements of iStar 2.0, we do not consider actor association links like *participates-in* and *is-a*. We define the problem as follows:

$$\begin{aligned} & \underset{ag_1, \dots, ag_n}{\text{maximize}} && \sum_{i=1}^n w * ag_i \\ & \text{subject to} && \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{B} \end{aligned} \quad (1)$$

Our objective function is the level of satisfaction of the entire model, which is the weighted sum of the satisfaction of the actors in the model (ag_i) and the $\mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{B}$ are the linear constraints derived from the model in the form of $A * x \leq b$ or $A * x = b$. We have a single weight w because we consider each actor equally important. We will use different weights if actors have different importance in the model. Constraints \mathcal{A} model the satisfaction value of an agent, which is the weighted sum of the satisfaction value of its intentional root elements. We have a linear constraint for each agent, for example for the agent k that has m root intentional elements, the constraint would be $ag_k - \sum_{i=1}^m w * ele_i = 0$.

\mathcal{R} represents constraints derived from AND and OR refinements. In previous contributions [1], refinements are evaluated using *max* (for OR contributions) and *min* (for AND contributions) functions depending on their type, which can be modeled as linear constraints as well. For example, to model the expression $x = \max(x_1, x_2)$, we define a binary decision variable y , which is 1 if $x_1 > x_2$ and 0 otherwise, and a variable M , which is bigger than x_1 and x_2 in any solution of the problem. Then we define several constraints to enforce the value of y , ($x_1 - x_2 \leq My$ and $x_2 - x_1 \leq M(1 - y)$), and the value of X ($X \geq x_1, X \geq x_2, X \leq x_1 + M(1 - y)$ and $X \leq x_2 + My$). This approach can be generalized for refinements with more than two leaf elements.

\mathcal{C} represents constraints derived from contributions to qualities. We model this as a weighted sum in which the weights depend on the kind of link. So, for *make* links the weight is 1, for *help* the weight is 0.75, for *hurt* links the weight is 0.25 and for *break* links is 0. These weights for the different kinds of contributions are aligned with the proposal of [1]. The linear constraint would be $RootQuality - \sum_{i=1}^n Cont_{make} - 0.75 \sum_{i=1}^n Cont_{help} - 0.25 \sum_{i=1}^n Cont_{hurt} = 0$.

\mathcal{D} represents constraints derived from dependencies between actors. In this kind of relationship, we have five entities: the depender, the depender element, the dependum, the dependee

and the dependee element. According to the iStar 2.0 specification, the satisfaction of the depender elements depends on the dependum. So, we use an equality constraint to enforce this issue with the form $DependerElement - Dependum = 0$. On the other hand, if the dependee element is not satisfied, the dependum cannot be satisfied as well. This is modeled using the linear constraint $Dependum - DependeeElement \leq 0$. Where there is no depender or dependee element, we use its actor's satisfaction level.

The \mathcal{E} represents these variables in the model that we want to set to a specific value. In a forward analysis, we would like to know the configuration of the model to achieve a certain level of satisfaction in an entity. We model this issue using equality constraints. Finally, \mathcal{B} represents the boundings of the model entities that should have a value between 0 and 100.

4. Proof of concept

To validate our approach, we have implemented a command-line utility in Java (available in <https://github.com/iayalavinas/istar20>) that takes models from the piStar tool¹ and generates a Matlab script. The piStar tool permits the export of models in JSON format. We use the *json-simple* library to parse piStar models. Matlab has different libraries to optimise linear programs, we use *intlinprog*².

We have validated our utility for models of different sizes. We see that even for models of more than 100 elements, Matlab and our scripts provide solutions in less than 10 seconds. This time is reasonable for a program not intended for user interaction or real-time interaction. For example, for the initial model of the piStar tool (see Figure 1), we obtain the Matlab script that appears in the screenshot of Figure 2. In line 2 appears the objective function, and as Intlinprog only supports minimisation problems, we multiply its weights by -1 . Intlinprog requires to indicate what variables of the model are integers. We do so in line 3 using the vector *intcon*. The script sets the equality and inequality constraints of the ILP using vectors *A*, *b*, *Aeq* and *beq* (lines 4-7). We set between 0 and 100 the satisfaction level of intentional elements in lines 8 and 9. The Intlinprog function is called in line 10. Finally, we print the index of the variable to interpret the results (lines 11-28).

5. Conclusions

In this work, we have presented a formalisation of iStar 2.0 models in linear constraints that allows optimising these models using ILP. IStar 2.0 models, which are modeled using the piStar tool, are then converted into an ILP problem using our proof of concept tool and resolved in Matlab. Our formalisation considers most iStar 2.0 entities such as intentional elements, agent dependencies, contributions and refinements. Although similar approaches exist for other goal modelling languages like GRL, KAOS or RELAX, they use other formalism that requires specific solvers like SMT or SAT or does not permit the model optimisation. Other work uses ILP for goal models, but it only considers a limited sub-set of the entities and does not evaluate

¹<https://www.cin.ufpe.br/~jhcp/pistar/>

²<https://es.mathworks.com/help/optim/ug/intlinprog.html>

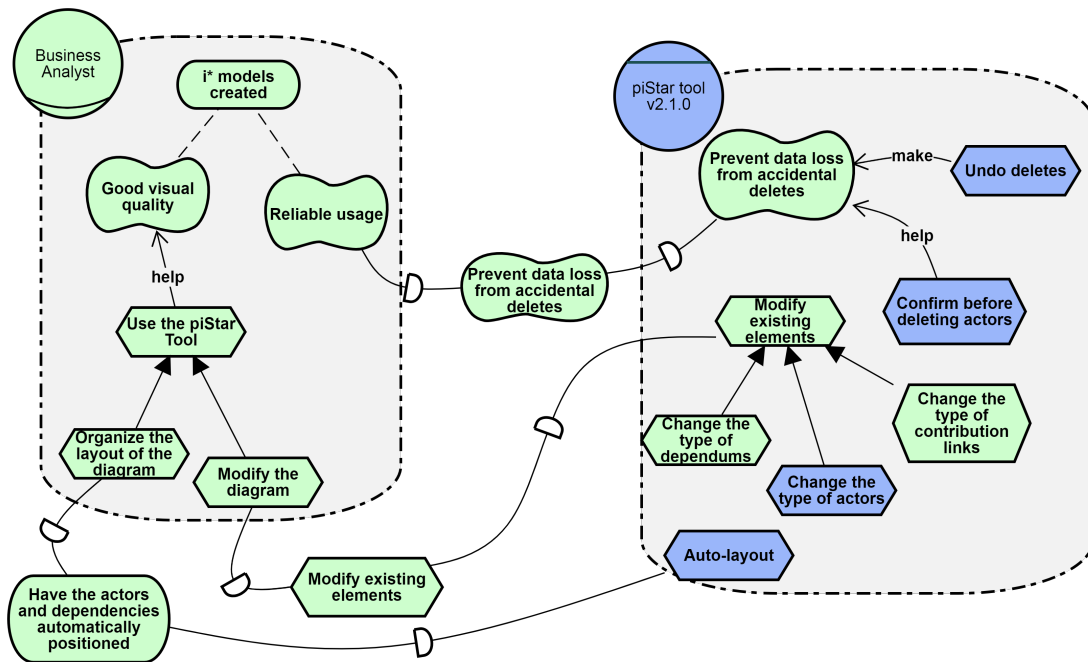


Figure 1: Goal model example of the piStar tool

the model's satisfaction level. In future work, we plan to integrate this formalisation into our approach for Proactive Dynamic Software Product Lines.

Acknowledgments

This work is supported by the European's H2020 research and innovation programme under grant agreement DAEMON 101017109, by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 and Rhea P18-FR-1081, the PRE2019-087496 grant from the MICINN and by DISCO B1-2012 12 funded by Universidad de Málaga.

References

- [1] J. Horkoff, E. Yu, Interactive goal model analysis for early requirements engineering, *Requirements Engineering* 21 (2016) 29–61.
- [2] E. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in: *Proc. of ISRE '97, 1997*, pp. 226–235. doi:10.1109/ISRE.1997.566873.
- [3] F. Dalpiaz, X. Franch, J. Horkoff, *istar 2.0 language guide*, arXiv preprint (2016).
- [4] D. Amyot, G. Mussbacher, URN: Towards a new standard for the visual description of requirements, in: *International Workshop on System Analysis and Modeling*, Springer Berlin Heidelberg, 2003, pp. 21–37.

```

prueba_1.m x +
1  % This is a automatically generated file for the iStar model basic01.txt
2  f = [ -0.5;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0; -0.5;0.0;0.0;0.0;0.
3  intcon = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23];
4  A = [0.0,0.0,-1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,-1.0E15,0.0,0.0,0.0,0.0,0.0,0
5  b = [0.0;1.0E15;0.0;0.0;0.0;1.0E15;0.0;1.0E15;0.0;0.0;1.0E15;0.0;0.0;1.0E15
6  Aeq = [0.0,0.0,0.0,-1.0,0.0,-0.75,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
7  beq = [0.0;0.0;0.0;0.0;0.0;0.0;0.0];
8  lb = zeros(23,1);
9  ub = [100;100;100;100;100;100;100;100;100;100;100;1;1;100;100;100;100;100;1
10 x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
11 % guide to interpret results (intentional_element,index)
12 % (Change the type of contribution links,7)
13 % (Confirm before deleting actors,5)
14 % (Modify existing elements,20)
15 % (Change the type of actors,2)
16 % (Reliable usage,16)
17 % (y_0,10)
18 % (Change the type of dependums,4)
19 % (Prevent data loss from accidental deletes,22)
20 % (Prevent data loss from accidental deletes,8)
21 % (y_last,19)
22 % (Business Analyst,12)
23 % (i* models created,13)
24 % (Organize the layout of the diagram,14)
25 % (Modify the diagram,15)
26 % (Undo deletes,3)
27 % (alfa_0,9)
28 % (Use the piStar Tool,17)

```

Figure 2: Screenshot of the Matlab script for the initial example of the piStar tool

[5] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An agent-oriented software development methodology, *J. AAMAS* 8 (2004) 203–236.

[6] H. Luo, D. Amyot, Towards a declarative, constraint-oriented semantics with a generic evaluation algorithm for grl, in: *Proc. of the 5 th International i* Workshop, 2011*, p. 26.

[7] Y. Fan, A. A. Anda, D. Amyot, An arithmetic semantics for GRL goal models with function generation, in: *System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering*, Springer International Publishing, 2018, pp. 144–162.

[8] M. Noorian, E. Bagheri, W. Du, Toward automated quality-centric product line configuration using intentional variability, *Journal of Software: Evolution and Process* 29 (2017).

[9] C. Ponsard, R. Darimont, Towards multi-objective optimisation of quantitative goal models using constraint programming., in: *ICORES, 2020*, pp. 286–292.

[10] A. Anda, D. Amyot, An optimization modeling method for adaptive systems based on goal and feature models, in: *MoDRE, 2020*, pp. 11–20.

[11] B. Kolman, *Elementary linear programming with applications*, Computer Science and Scientific Computing, 2nd ed. ed., Academic Press, San Diego, California, 1995.

[12] J. a. Pimentel, J. Castro, piStar tool – a pluggable online tool for goal modeling, in: *IEEE 26th International Requirements Engineering Conference (RE), 2018*, pp. 498–499.

- [13] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, E. Yu, Evaluating goal models within the goal-oriented requirement language, *International Journal of Intelligent Systems* 25 (2010) 841–877.
- [14] C. M. Nguyen, R. Sebastiani, P. Giorgini, J. Mylopoulos, Multi-objective reasoning with constrained goal models, *Requirements Engineering* 23 (2018) 189–225.
- [15] A. J. Ramirez, B. H. C. Cheng, Automatic derivation of utility functions for monitoring software requirements, in: *Model Driven Engineering Languages and Systems*, Springer Berlin Heidelberg, 2011, pp. 501–516.