piStar Tool – A Pluggable Online Tool for Goal Modeling

João Pimentel
Universidade Federal Rural de Pernambuco
Pernambuco, Brazil
Email: joao.hcpimentel@ufrpe.br

Jaelson Castro
Universidade Federal de Pernambuco
Pernambuco, Brazil
Email: jbc@cin.ufpe.br

***Abstract*—Since its proposal in the nineties, the *i** goal modeling proposal has spawned a number of complementary work, including: language extensions, automated reasoning, and transformation to/from *i** models. In order to automate their proposals, researchers need to either create a new tool from scratch or to extend an existing tool. In fact, the *i** Wiki lists 29 different *i** modeling tools. A common approach for creating such tools has been to use the Eclipse tools as a foundation (12 listed tools), which presents complexity issues of its own. The main idea of this work is the creation of an *i** modeling tool with low entry barriers for developers wishing to extend it. Our piStar tool can be extended with JavaScript, requiring no specific development tools. Alternatively, developers can adopt their language of choice (such as Java and Python), using piStar models as input either through manually downloading the model file or by sending it for processing through a REST request.**

I. Introduction

The *i** modeling language has been widely adopted by the requirements engineering academic community [1]. In 2016 it was updated to *i** 2.0, through a collaborative effort [2].

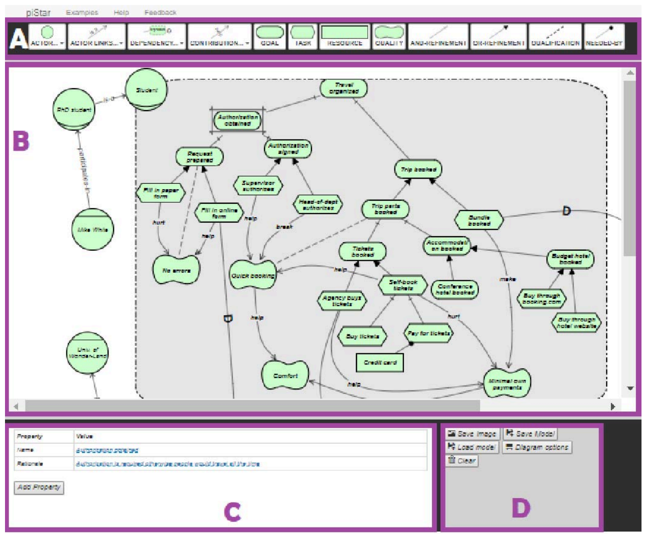
The piStar tool here presented is a modeling tool compliant with the *i** 2.0 standard. It prevents some common mistakes that would result on invalid models by not allowing its users to create invalid links, such as dependency links from an actor to itself or contribution links to tasks. Besides being usable as a tool itself, piStar has been designed as a platform to facilitate development of *i**-based tools.

Many proposals build upon the *i** language [3]. On one hand, some of them aim to extend the language itself – e.g, context annotations, behavioral expressions, cross-cutting concerns, cardinality annotations, modularization mechanisms, etc. On the other hand, some proposals do not modify the language, but use its models as input — e.g., for analysis, for automated reasoning, and for executing model transformations, where .

This paper presents the piStar tool and describes how it can be extended, for the benefit of researchers whom need to provide tool support for their *i**-based proposals.

II. The piStar tool

Figure 1 shows an overview of the piStar tool[1] user interface. Figure 1-A shows the modeling palette, inspired by the design of the OME tool[2]. The elements and links of the palette



Property | Value
Name | Authorization obtained
Rationale | Authorization is required otherwise people would travel all the time

Fig. 1. A screenshot of the piStar tool

can be added to the drawing area (Figure 1-B). The visual design of the diagram is based on the original *i** notation [1], with one conscious deviation: a rounded rectangle is used in place of a circular shape for the representation of actors' boundaries[3].

Similarly to the OME tool, actors can be collapsed and extended at will. Thus, one can create different views (Strategic Dependencies and Strategic Relationships) of the same model.

Figure 1-C shows the properties of the selected element (in this case, the "Authorization obtained" goal). Users can freely add custom attributes to their models (similarly to jUCMNav [4]), by means of the "Add Property" button. In this example, a "Rationale" property has been added to the selected element. This functionality allows researchers to easily create models to experiment with their *i** extensions – e.g., context annotations, behavior expressions, and cardinality. Nevertheless, some implementation would still be required if one needs to display such information visually.

[1]Live version: http://www.cin.ufpe.br/~jhcp/pistar/
[2]OME tool: http://www.cs.toronto.edu/km/ome/

[3]The choice of a rectangular shape intends to reduce wasted space, considering that the diagrams are usually displayed in rectangular media (e.g., computer screens and print paper).

2332-6441/18/$31.00 ©2018 IEEE
DOI 10.1109/RE.2018.00071

Last but not least, Figure 1-D displays the main menu of the tool. Models can be exported as SVG or PNG images. They can be saved and loaded as JavaScript Object Notation (JSON) files. These files can be processed with the help of JSON libraries, which are available for all major programming languages. For instance, the following Java method using the Jackson 2 library prints out the name of every actor in a specific model.

```
private static void printActorsNames() {
 ObjectMapper mapper = new ObjectMapper();
 try {
  HashMap<String, Object> map = mapper.readValue(new
       File("c:\\goalModel.txt"), new TypeReference<
      Map<String, Object>>(){});
  for (HashMap actor : (ArrayList<HashMap>) map.get(
      "actors")) {
   System.out.println(actor.get("text")); } }
 catch (Exception e) {  e.printStackTrace(); } }
```

III. Extending the Tool

The source code and documentation of the tool are publicly available on an open source repository[4]. In order to reduce the entry barriers for new developers, the project was architected as to allow extending the tool without the need for compilation, build processes, specific tools, or web servers. The only required tools are a text editor and an up-to-date web browser.

Developers can add new elements to the user interface by manipulating the HTML DOM (HyperText Markup Language Document Object Model). For instance, in order to add a new button, this JavaScript code excerpt[5] is sufficient:

```
$('#appToolbar').append('<button type="button" id="
    exampleButton"> Calculate Metric </button>');
```

A. Client-side processing

To facilitate the development of plugins that run on client-side (i.e., on the user's web browser), piStar provides a *i** 2.0 modeling API (Application Programming Interface). This API includes methods for editing the current model, such as *addActor*, *addGoal*, *clearModel*, and *element.remove*.

The tool's API also includes functions for querying the model, such as *getElements* and *getLinks* (for retrieving lists of the elements and the links of the model, respectively), *element.isGoal* (for checking whether a specific element is a Goal) and *element.isKindOfActor* (for checking if a given element is an Actor or an Agent or a Role), and *element.prop('customProperties')* (for accessing the custom attributes of a specific element).

As an example, the following JavaScript code excerpt calculates and displays the number of goals of the current model:

```
elements = istar.getElements();
goals = _.filter(elements, function(element) {
  return element.isGoal(); });
alert('Number of goals: ' + goals.length);
```

[4]Source code and documentation: https://github.com/jhcp/pistar

[5]The code presented in this section uses the *jQuery* and *lodash* libraries; the same functionality can be developed with plain JavaScript as well.

B. Server-side processing

Alternatively, models can be processed remotely by means of web services. This possibility allows developers to use their programming language of choice (e.g., Java, Python, Haskell), also enabling them to make use of legacy code.

The following code sends the current model to a web service through a POST request. In the event of success, the response from the web service is displayed to the user.

```
$.ajax({
 type: "POST", url: "www.example.com/service",
 contentType: "application/json", data: saveModel(),
 success: function(response) { alert(response); }});
```

Once the model is received by the web server, it can be processed by making use of JSON libraries as exemplified in Section II.

IV. Related Work and Conclusion

The GATO tool [5] can be considered a predecessor of piStar – the former is also an online goal modeling tool, but it does not support the social concepts of *i** (such as actors and their dependencies). The Leaf 2.0 tool[6] is similar to piStar in the sense that it is a web-based *i** 2.0 modeling tool. It constitutes the basis of three more specific tools: CreativeLeaf[6] and GrowingLeaf/BloomingLeaf [7]. Unlike piStar, it does not support the creation of dependency links.

The piStar tool is a platform for the development of *i**-based tools. It enables the creation of *i** 2.0 models that can be extended with custom properties. Moreover, it provides an *i** 2.0 modeling and querying API that can be used to create additional functionalities in a plugin-like structure. Some piStar extensions already developed include an online model sharing service[7] and a dependability analysis plugin[8].

Future work will concentrate on fleshing out the modeling capabilities of the tool while improving its user interface based on usability evaluations.

Acknowledgment

Work partially supported by CNPq and FACEPE.

References

[1] E. Yu, P. Giorgini, N. Maiden, and J. Mylopoulos, *Social modeling for requirements engineering*. Mit Press, 2011.

[2] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 Language Guide," *arXiv preprint*, no. arXiv:1605.07767 [cs.SE], pp. 1–15, 2016.

[3] E. Gonçalves, J. Castro, J. Araújo, and T. Heineck, "A systematic literature review of istar extensions," *Journal of Systems and Software*, vol. 137, pp. 1–33, 2018.

[4] D. Amyot, G. Mussbacher, S. Ghanavati, and J. Kealey, "GRL Modeling and Analysis with UCMNav," *Proceedings of the 5th International i* Workshop*, pp. 160–162, 2011.

[5] J. Pimentel, J. Vilela, and J. Castro, "Web tool for Goal modelling and statechart derivation," *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, pp. 292–293, 2015.

[6] J. Horkoff and N. Maiden, "Creative leaf: A creative iSTAR modeling tool," *CEUR Workshop Proceedings*, vol. 1674, pp. 25–30, 2016.

[7] A. M. Grubb, G. Song, and M. Chechik, "GrowingLeaf: Supporting requirements evolution over time," *CEUR Workshop Proceedings*, vol. 1674, pp. 31–36, 2016.

[6]Leaf 2.0 tool: https://github.com/amgrubb/Leaf2.0

[7]https://github.com/maxguenes/piStar

[8]https://github.com/leandrobergmann/pistargodaintegration