1-1-2016

Understanding Challenges and Tradeoffs in iStar Tool Development

Tong Li
Beijing University of Technology

Alicia M. Grubb
University of Toronto, amgrubb@smith.edu

Jennifer Horkoff
City, University of London

Understanding Challenges and Tradeoffs in iStar Tool Development

Tong Li[1], Alicia M. Grubb[2], and Jennifer Horkoff[3]

[1] Beijing University of Technology, Beijing, China
litong@bjut.edu.cn
[2] University of Toronto, Toronto, Canada
amgrubb@cs.toronto.edu
[3] City University London, London, UK
horkoff@city.ac.uk

Abstract. Many iStar-based modeling and analysis tools have been developed for specific iStar-related purposes (e.g., tools enumerated in iStar wiki). Despite the proliferation of tools, new tools keep being built, taking resources and time and possibly "reinventing the wheel" in terms of tool functionality. To gain an in-depth understanding of this situation, we are interested in the challenges and tradeoffs in creating iStar tools. In this paper, we examine three diverse tools as case studies, illustrating the challenges and tradeoffs from the developer's perspective. Based on such studies, we establish a goal model to capture the collected knowledge, which can help researchers to make informed decisions and optimize their tool development.

1 Introduction

The iStar modeling language is suitable for modeling and analyzing stakeholders' requirements in the early phase of system development. Over the past decade, many researchers have extended the syntax and created new analysis techniques for a variety of purposes (e.g., TROPOS, PRiM, and RiSD [7]). Researchers have built and continue to build a variety of tools in order to validate these new extensions. The iStar wiki [1] list 26 tools, yet many recent tools (i.e., MUSER [15], Leaf [8], and pistar[13]) are not yet listed.

When embarking on tool development, it is important for researchers to consider development technologies that will satisfy their requirements and mitigate common challenges for modeling tools (i.e., usability, model scalability, installation, and maintenance). Related work has reviewed iStar tool functionality, evaluating the coverage of syntactic constructs [2], to facilitate tool selection from the user point of view. However, such work does not discuss the underlying development of these tools.

In this paper, we examine technical tradeoffs in tool development choices from a developer point of view. As active tool developers, we (the authors of this paper) are intensely involved and have in-depth expertise. We examine our tools, profiling their development rationale, to understand how technology tradeoffs impact tool qualities. We contribute a goal model to explore the tradeoffs of

tool development and an initial list of desirable qualities for iStar tools, with the intention of helping researchers make informed decisions about adopting existing tools or developing their own new tools. We hope that this work helps to create a discussion in the iStar community about tool development, sharing, and reuse.

2 Examination of iStar Tools

In this section we introduce and discuss selected iStar tools. We cover tools in which the authors have particular expertise. Future versions of this study should have systematic inclusion and exclusion criteria, covering a wider range of tools. For each tool, we first briefly describe its features; and then present its architecture, design, and used technologies; finally, we discuss pros and cons based on qualities desired by its developers.

OpenOME is an Eclipse-based tool [12] for the creation and analysis of iStar models using the iStar Framework syntax described in the iStar Wiki [1]. The tool allows users to create iStar models graphically using a palette of shapes. OpenOME imports and exports models in the GMF .ood and .oom format, the Q7 textual modeling language, and the iStarML interchange format [3]. OpenOME supports the forward and backward interactive, qualitative iStar analysis procedures described in [11]. Visualizations have been added to highlight model leaves and roots (potential starting points for analysis), areas of human judgment, and the intentions involved in a conflict in backward analysis.

OpenOME uses the Eclipse platform, taking advantage of the Eclipse Modeling and Graphical Modeling Frameworks (EMF & GMF). Use of these frameworks allows us to automatically generate code from an iStar metamodel. This code has been customized and expanded to support features specific to iStar modeling. OpenOME architecture takes advantage of the Eclipse package environment, allowing for extension or customization with the addition of new packages.

We can use our experience to examine the pros and cons of OpenOME development. On one hand, the Eclipse platform (including EMF & GMF) provides *metamodel support*, and supports automatic code generation *reducing development effort*. In theory, one should be able to use Eclipse as a modeling tool without coding. In practice, the iStar modeling capability provided by the default GMF functionality is non-optimal: no support for collapsing actors, special behaviour and look of links and softgoals. As a result much of the code had to be customized, and since the auto-generated code was difficult to understand, further customization proved difficult. When the metamodel was updated (*maintainability*) bugs were often introduced, affecting tool *stability*, and the interface between the generated and custom code had to be manually analyzed and fixed.

Users had to download a full version of Eclipse with OpenOME embedded, having a negative effect on *ease of installation* (the alternative was to make OpenOME available as a downloadable Eclipse project, requiring users to already have Eclipse installed and understand project installation). For users with a technical background, this was fine, but less technical users were forced to use a heavy-weight, large and complex tool, impeding *usability*.

MUSER is a prototype tool which is designed to support three-layer security requirements analysis [15]. Specifically, it helps analysts to model security requirements with an extended iStar language, and supports automatic security requirements refinement and operationalization (as specified in [14]).

By extending OmniGraffle[4], a specialized and powerful diagramming application, the development of MUSER focused exclusively on investigating model reasoning. A Java and AppleScript-based program was developed on top of OmniGraffle to manage graphical models in the canvas using corresponding APIs. Specifically, the program transforms graphical models into Datalog and performs inferences. All inference results are reflected in the canvas, e.g., highlighting critical security goals, and creating new refinements of security goals.

We examine the pros and cons of MUSER development. The loosely-coupled architecture helps to improve *extensibility*, allowing developers to incrementally add new modeling notations or inference tasks to corresponding modules. In addition, the tool inherits good *usability* from OmniGraffle, such as automatic layout, batch selection of similar elements, and copy/paste, facilitating the modeling practices. OmniGraffle also provides comprehensive and detailed manuals, contributing to *learnability* of the tool. The tool supports model *scalability*, allowing modelers to create and browse large-scale models. This is the most desired quality by the developer, as the tool is used to model and analyze security requirements for large Socio-Technical Systems (STSs) which normally involve several hundred elements. On the other hand, OmniGraffle, as an indispensable component of MUSER, is a commercial application which only functions in Mac OS, impairing *ease of installation* and the desire to be *cross-platform*. Moreover, the metamodel is embedded in the tool restricting users from making edits.

Leaf (Beta) is intended to be a simple web-based iStar modeling tool [8]. The basic tool contains a palette of iStar concepts, and a canvas where elements can be drawn. The developers have plans to update the palette to conform to the latest version of the iStar 2.0 core [4].

Leaf (beta) uses both JointJS and the Rappid diagramming framework[5]. JointJS is available via an open license, while Rappid is a commercial product, but provides free academic licenses. Code is written in JavasScript, using CSS and HTML files for formatting and interfacing with the Web. The Leaf code has been expanded and adapted for two separate projects: Creative Leaf and GrowingLeaf.

Creative Leaf aims to combine the benefits of established creativity techniques with iStar modeling for enhanced requirements engineering [10]. The Leaf code has been modified and adapted, e.g., to add ideas and assumptions, and to add a creativity panel to allow users to use structured creativity techniques within the tool. Creative activities are aimed to support either divergent creativity, generating ideas, or convergent creativity, selecting, combining and developing ideas.

[4] `http://www.omnigroup.com/omnigraffle`

[5] `http://www.jointjs.com`

GrowingLeaf models the evolution of goal models over time. Using iStar Strategic Rationale diagrams it extends the static notion of iStar by allowing the qualitative evaluation of intentional elements (i.e., goals, tasks, qualities) to change over time (as described in [9]). In addition to enabling users to model dynamicity, GrowingLeaf allows users to create simulations based on either the initial states of the model (using a random simulation), or desired intermediate or final states of the model (by encoding the model as a Constraint Solving Problem).

We consider the pros and cons of the Leaf-family of tools together. Leaf tools are *lightweight* in that they are used in the browser and do not contain many lines of auto-generated code. They are simpler and have better *usability* compared to more heavy-weight tools like OpenOME. *Installation* is trivial. On the downside, there is no explicit *support for metamodel* representation or extension. From a developer point of view, development is easy or hard depending on the level of familiarity with JavaScript, CSS and HTML. The use of JointJS and Rappid, with their accompanying documentation, gives a good starting point for general modeling functionality, but customizations and extensions are still needed. Like all browser-based tools, there is the challenge of making the tool *compatible* with all browsers. Thus far, functionality has only been carefully tested in Chrome.

3 Analysis Model for Tool Development

Based on our experiences in developing and working with the previously described tools, we built an initial goal model (using the iStar 2.0 language guide [4]), illustrating challenges and tradeoffs of alternative technical choices, which is shown in Fig. 1. In particular, each tool is modeled as a *task*. The desired qualities of tools are modeled as *qualities*, which can be positively/negatively influenced by alternative tools. Note that all such qualities were identified and highlighted in italic in Sec. 2, and we leave the detailed explanation of these qualities and contributions to future work. Finally, specific technologies/languages that were required in developing each tool (e.g., Java and Datalog) are modeled as resources. The qualities we have identified are not independent (e.g., a Lightweight tool helps Reduce Development Effort). Thus, we omit the interactions between qualities to keep the model simple, in this version.

The qualities in Fig. 1 can be used as a checklist by future iStar tool developers, helping them to elaborate their tooling requirements. Once developers decide what qualities they desire in their tool, they can check whether existing tools can adequately satisfy such qualities. Developers can either incrementally build their tool on top of those existing tools, inheriting all of the tool's characteristics, or develop a new tool from scratch. When building new tools, developers may find it helpful to review the particular techniques used by existing tools and understand their tradeoffs, in order to choose an optimal technical solution.

4 Related Work

The iStar wiki tool list, currently summarizes 26 iStar tools [1]. Each tool has been examined in terms of features, e.g., whether the tool allows SR/SD model-

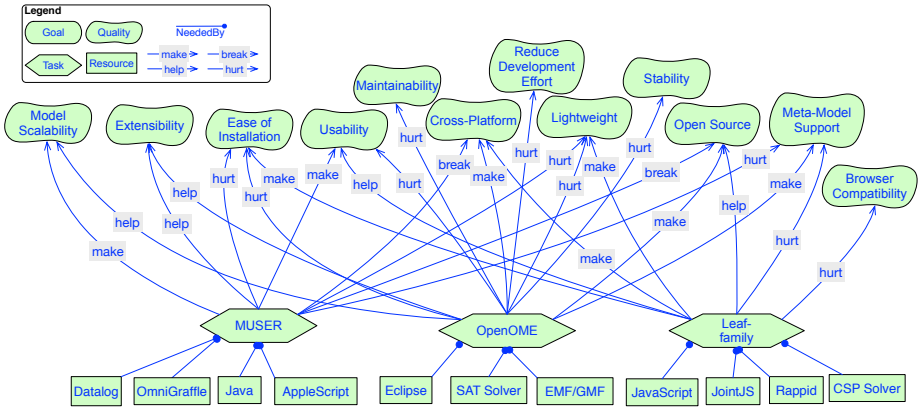

Fig. 1: Tradeoffs among different techniques

ing. In addition to this summary, Almeida et al. surveyed six iStar tools, examining the coverage of syntactic constructs of each tool [2]. Both of these studies focused on the features of iStar tools from the user's perspective, which are intended to help researchers to find a suitable iStar tool to use. De Gea et al. surveyed existing Requirements Engineering (RE) tools [5], assessing how and to what degree each tool supports the RE process by means of concrete capabilities. Our work departs from these previous studies in that we focus on the tradeoffs and challenges in developing iStar tools from the developer's perspective, with the aim of facilitating the development of new iStar tools.

iStarML offers an XML compliant format to represent iStar diagram, enabling the interoperability among iStar variants [3]. With the help of iStarML, researchers can easily exchange models that are built using different meta-models. When it comes to tool development, iStarML, as a common file interface, can contribute to collaborative tool development.

A diverse collection of tools can be a sign of a prosperous research community, but we have also witnessed the "death" of iStar tools. According to the iStar wiki, 8 out of 25 tools are no longer available. This situation also appears in other research communities. For example, Eichelberger et al. performed a comprehensive survey starting with an initial set of 200 existing UML tools, and found that only 68 of them were available for use [6]. The continuous creation and "death" of iStar tools might be an inevitable part of a maturing field. After enough competition, the surviving tools should become de facto baselines, on top of which new tools can be incrementally built, promoting the maturity of iStar tooling. In order to accelerate this procedure, iStar researchers should try to consolidate iStar 2.0 tools, establishing an open iStar tool that can be extended by the community.

5 Conclusions and Future Work

In this paper, we examine a selection of technical choices for creating iStar tools from the developer's perspective, understanding the challenges and tradeoffs.

This is ongoing research, we are planning to study further existing iStar tools in depth (e.g., jUCMNav[6]), mapping out a full picture of available techniques. Specifically, we intend to evaluate our study with potential tool developers, for example, PhD students who work on iStar models and need tooling support in their thesis. We believe this and future studies can help researchers justify their iStar tool development choices, avoid reinventing the wheel, and optimize their development process.

Acknowledgments. Jennifer is supported by an ERC Marie Skodowska-Curie Intra European Fellowship (PIEF-GA-2013-627489) and a Natural Sciences and Engineering Research Council of Canada Postdoctoral Fellowship (Sept. 2014 - Aug. 2016).

References

1. i* tool wiki. `http://istarwiki.org/tiki-index.php?page=i%2A+Tools`.
2. C. Almeida, M. Goulao, and J. Araújo. A systematic comparison of i* modelling tools based on syntactic and well-formedness rules. In *Proc. iStar'13*, pages 43–48, 2013.
3. C. Cares, X. Franch, A. Perini, and A. Susi. Towards interoperability of i* models using istarml. *Computer Standards & Interfaces*, 33(1):69–79, 2011.
4. F. Dalpiaz, X. Franch, and J. Horkoff. iStar 2.0 Language Guide. *arXiv:1605.07767*, 2016.
5. J. M. C. De Gea, J. Nicolás, J. L. F. Alemán, A. Toval, C. Ebert, and A. Vizcaíno. Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10):1142–1157, 2012.
6. H. Eichelberger, Y. Eldogan, K. Schmid, and M. Platz. A Comprehensive Survey of UML Compliance in Current Modelling Tools. *Software Engineering*, 143:39–50, 2009.
7. G. Grau, C. Cares, X. Franch, and F. Navarrete. A comparative analysis of i* agent-oriented modelling techniques. In *Proc. of SEKE'06*, pages 657–663, 2006.
8. A. M. Grubb. Leaf (beta): An istar modeling tool. `http://www.cs.toronto.edu/~amgrubb/leaf`, 2015. Accessed: 2016-07-28.
9. A. M. Grubb and M. Chechik. Looking into the Crystal Ball: Requirements Evolution over Time. In *Proc. of RE'16*, 2016.
10. J. Horkoff and N. Maiden. Creative Leaf: A Creative iStar Modeling Tool. In *Proc. of iStar'16*, 2016.
11. J. Horkoff and E. Yu. Interactive goal model analysis for early requirements engineering. *Requir. Eng.*, 21(1):29–61, 2016.
12. J. Horkoff, Y. Yu, and E. Yu. OpenOME: An Open-source Goal and Agent-Oriented Model Drawing and Analysis Tool. In *Proc. of iStar'11*, pages 154–156, 2011.
13. João Pimentel. pistar. `http://www.cin.ufpe.br/~jhcp/pistar/`, 2016. Accessed: 2016-07-28.
14. T. Li and J. Horkoff. Dealing with security requirements for socio-technical systems: A holistic approach. In *Proc. of CAiSE'14*, pages 185–200, 2014.
15. T. Li, J. Horkoff, and J. Mylopoulos. A prototype tool for modeling and analyzing security requirements from a holistic viewpoint. In *Proc. of CAiSE'14 (Forum/Doctoral Consortium)*, pages 185–192, 2014.

[6] `http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome`