10-12-2018

BloomingLeaf: A Formal Tool for Requirements Evolution Over Time

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

Marsha Chechik
University of Toronto

BloomingLeaf: A Formal Tool for Requirements Evolution over Time

Alicia M. Grubb, Marsha Chechik
Department of Computer Science
University of Toronto, Toronto, Canada
{amgrubb, chechik}@cs.toronto.edu

***Abstract*—Our previous work presented the Evolving Intentions framework, which specified how evolving qualitative goal models can be modeled and analyzed. Recent improvements to the framework allow for precise semantics of goal relationships with propagation of both evidence for and evidence against a goal's satisfaction (as in Tropos), and enables evaluation of evolution with absolute time (in addition to relative time). The reasoning is expressed as a constraint satisfaction problem. In this paper, we present BloomingLeaf, a new web-based tool that implements the new semantics. We showcase how the implementation and architecture of BloomingLeaf can be used to answer time-based questions.**

I. Introduction

Goal-Oriented Requirements Engineering (GORE) aims to help stakeholders make trade-off decisions when planning a project [1]. We extended GORE by allowing stakeholders to answer questions about their projects when the goals of the project change over time [2], [3]. We investigated using *evidence pairs* (i.e., separating evidence for and evidence against the satisfaction of a goal) and both symmetric and asymmetric links as a way to evaluate goal models [4]. By using evidence pairs, we formally defined how goals change over time in our framework, called Evolving Intentions [5]. In this paper, we introduce BloomingLeaf, a web-based tool for formal automated analysis of goal graphs. Using the motivating example that follows, we describe Evolving Intentions and BloomingLeaf in Sec. II. Sec. III discusses the architecture and Sec. IV describes ongoing and future work.

Motivating Example. Consider the task of adding bike lanes to an urban street, as shown in the goal graph fragment in Fig. 1. The stakeholder is considering the trade-off between building the bike lanes as a Temporary Construction or a Permanent Construction, and the impact of this decision on three goals of interest: Have Bike Lanes, Cyclist Safety, and Access to Parking. The stakeholder wants to ask the question: Which construction plan will satisfy Have Bike Lanes at 36 months? We can specify how each goal in the graph might change over time, and create simulations of how the evolving goals impact the model as a whole.

II. BloomingLeaf Overview

Using the running example, we describe how to model and analyze goal graphs that evolve over time with BloomingLeaf.

Goal Modeling. Goal models (or goal graphs) consist of goals and links forming directed acyclic graphs. Goals can be

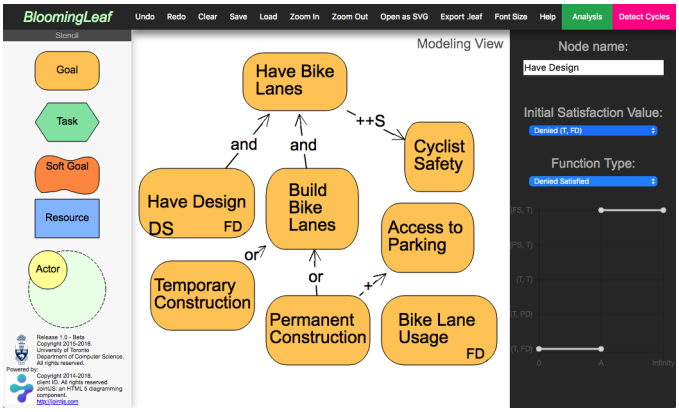

Fig. 1: Model of Motivating Example in BloomingLeaf.

decomposed into child goals requiring all (and) or one (or) child goal for satisfaction, but can also contribute to each other using contribution links (e.g., +, −, ++, −−). For example, in Fig. 1 Have Bike Lanes is and-decomposed into Build Bike Lanes and Have Design.

Each goal can be evaluated using an *evidence pair* (s, d), where $s \in \{F, P, \bot\}$ is the level of evidence for and $d \in \{F, P, \bot\}$ is the level of evidence against the fulfillment of g. Thus, goals can have one of five values: $(F, \bot) =$ FS, $(P, \bot) =$ PS, $(\bot, P) =$ PD, $(\bot, F) =$ FD, and $(\bot, \bot)$, as well as four conflicting values: (F, F), (F, P), (P, F), and (P, P). In Fig. 1, Bike Lane Usage is given the value Denied (FD) because the bike lanes are not currently in use. There is a ++S contribution link between Have Bike Lanes and Cyclist Safety. The ++S link only propagates when Have Bike Lanes has F or P for its s value of the evidence pair. See [4] and [6] for propagation details.

Evolving Intentions. With these values, we define how the evaluation of a goal (as specified through evidence pairs) can change over time. The evidence pairs form a partial order from most satisfied to most denied. Over any time interval, a goal can have a valuation that is changing stochastically, increasing, decreasing or remaining constant. We combine these to form step-wise functions. For example, we could specify Have Design as Constant over two periods where it is Denied (FD) for the first period of time and then Satisfied (FS) for the second. We identify this pattern as *Denied-Satisfied* (*DS*), and Have Design is given a *DS* label in Fig. 1. See [3] for a full list of functions.

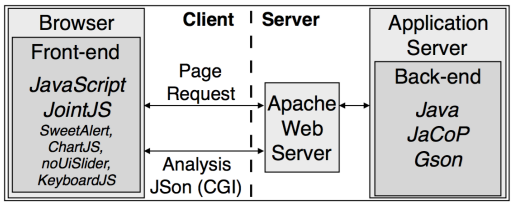

Fig. 2: Architecture of BloomingLeaf.

Simulation. Suppose we want to find a path where Have Bike Lanes is Satisfied (FS) at 36 months, to explore the question posed in Sec. I. To do this, we assign Have Bike Lanes a *Stochastic-Constant* (*RC*) evolving function, which specifies that after a time point t the evaluation of Have Bike Lanes will be Satisfied (FS) and remain Constant. Next, we define our analysis to be over months and assign $t = 36$ (not shown). By running Simulate Single Path, we generate a path where Have Bike Lanes is satisfied at 36 months (see demo video online[1]). By changing the evolving functions of goals and generating additional simulations, we can explore the best plan to satisfy Have Bike Lanes.

III. BloomingLeaf Architecture

BloomingLeaf is a web-based tool consisting of a *front-end* on the client side and a *back-end* on the server side, as illustrated in Fig. 2. The front-end is a dynamic webpage written in HTML and Javascript, which stores the goal model as a JSON string within the browser's cookies. The live demo is hosted on our Apache web-server. We built our interface on top of JointJS, a Javascript Diagramming Library, and use SweetAlert, ChartJS, noUiSlider, and KeyboardJS for interacting with and visualizing model elements. When a user makes an analysis request, the model and analysis configurations are packaged as a JSON string by the front-end and then passed to the back-end via the Common Gateway Interface (CGI). The back-end is a Java application that accepts an analysis request as a JSON string. The program uses Gson to convert the JSON string into Java objects (called Input Objects). The Input Objects are used to create Simulation Objects, which contain a collection of variables and constraints that form the constraint satisfaction problem (including forward and backward propagation rules). This collection is then passed to the Java Constraint Programming (JaCoP) solver, which returns a satisfiable set of assignments to the variables, or an error. If the solution exists then the Simulation Objects are converted to a set of Output Objects. Gson is then used to convert the Output Objects back to a JSON string which is returned to the front-end for rendering. Excluding all external libraries, the tool is approx. ∼13000 lines of source code.

IV. Discussion and Future Work

We developed BloomingLeaf for the purpose of modeling and analyzing goal models that change over time as described in [5]. The initial version of BloomingLeaf is complete and evaluation is ongoing. In this paper, we illustrated how BloomingLeaf can be used to answer a time-based question.

[1] *http://www.cs.toronto.edu/~amgrubb/archive/RE18-Demo*

BloomingLeaf is the second generation implementation of our framework and has the same look and feel as our first tool, GrowingLeaf [7]. GrowingLeaf implemented i* contribution links and qualitative evaluation labels (i.e., ✓, ✓•, ✗•, ✗, ϟ, and ?), which made it difficult to trace the propagation of labels in automated analysis. We kept the elements of the i* language in the stencil of BloomingLeaf for more explicit visualization and discussions between modelers, although these elements are not considered in the analysis. We also removed i* dependency links, but users may choose to use the ++ contribution link to represent dependencies.

We are not the only ones working on tools for evolving goal models. An alternative approach, TimedURN, enables stakeholders to consider possible evolutions and trends through visualizations of quantitative goal models and feature models [8]. While we are unable to represent quantitative evaluations in BloomingLeaf, with the addition of absolute time, BloomingLeaf can now make predictions over qualitative versions of models represented in TimedURN.

While we hope to get feedback from session participants on how to improve BloomingLeaf for wider release, we have some improvements planned. We are currently implementing an all paths analysis feature. This would give users the ability to explore all possible next states of a model and create their own simulation paths. Future work is required to add model management features as well as additional syntax checking. For example, we will give users the ability to save and load analysis configurations and results. We will explore overlaying symbolic satisfaction values (e.g., ✓ and ✗) on top of evidence pairs, as well as adding visualizations for analysis trends, similar to those in TimedURN [8].

BloomingLeaf is a public GitHub project available at *https://github.com/amgrubb/BloomingLeaf*, with a live-demo available at *http://www.cs.utoronto.ca/~amgrubb/leaf-blooming-ui* (Google Chrome is the recommended browser).

Acknowledgments. We would like to thank our development team: Hanbin Chang, Marcel Serikawa, and Yikhei Chan, as well as the members of the Modeling group in Toronto for their contributions to this work.

References

[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, J. Mylopoulos, and P. Giorgini, "Goal-Oriented Requirements Engineering: A Systematic Literature Map," in *Proc. of RE'16*, 2016, pp. 106–115.

[2] A. M. Grubb, "Adding Temporal Intention Dynamics to Goal Modeling: A Position Paper," in *Proc. of MiSE@ICSE'15*, 2015.

[3] A. M. Grubb and M. Chechik, "Looking into the Crystal Ball: Requirements Evolution over Time," in *Proc. of RE'16*, 2016.

[4] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models," *J. on Data Semantics*, vol. 1, pp. 1–20, 2003.

[5] A. M. Grubb, "Evolving Intentions: Support for Modeling and Reasoning about Requirements that Change over Time," Ph.D. dissertation, University of Toronto, 2018, forthcoming.

[6] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and Minimum-Cost Satisfiability for Goal Models," in *Proc. of CaiSE'04*, 2004.

[7] A. M. Grubb, G. Song, and M. Chechik, "GrowingLeaf: Supporting Requirements Evolution over Time," in *Proc. of i* Wrksp'16*, 2016.

[8] S. Luthra, Aprajita, and G. Mussbacher, "Visualizing Evolving Requirements Models with TimedURN," in *Proc. of MiSE@ICSE'18*, 2018.