Incorporating Presence Conditions into Goal Models that Evolve Over Time

Xinran Bi, Alicia M. Grubb
Department of Computer Science
Smith College, Northampton, MA, USA
amgrubb@smith.edu

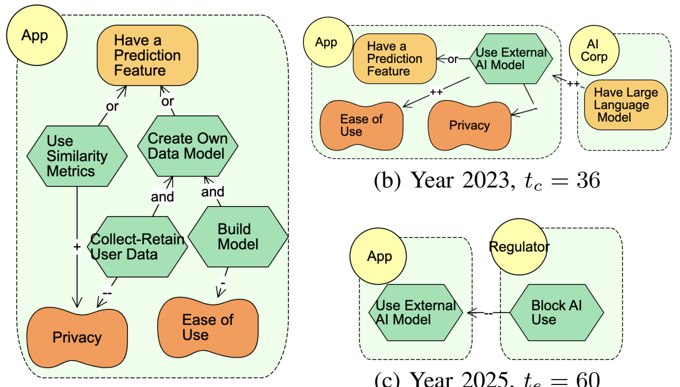
***Abstract*—Goal modeling helps analysts understand the needs and motivations of stakeholders. Recent work has investigated considering how these intentions change over time. In this paper, we investigate the problem of specifying and reasoning about goal models when some stakeholders and their intentions are not present for the entirety of a timeline under consideration. With inspiration from software product lines, we implement presence conditions to specify periods of time when an actor or intention is included/excluded. By incorporating presence conditions, goal models can more accurately represent and visualize real-world conditions and correctly propagate model information. We implement our approach in BloomingLeaf and evaluate it using a concrete example.**

I. Introduction & Motivation

Goal modeling languages have been used in the early-phases of projects to help stakeholders make decisions between alternative designs [1]. These models may then be connected downstream with other model-driven engineering approaches [2]. More recently, researchers have investigated how to analyze goal models over time [3], [4]. To validate the applicability of one time-based analysis framework, called Evolving Intentions, Grubb and Chechik investigated a road construction project over five decades [5]. As part of this work, they explored merging goal models over time when not all actors are present across the entire timeline. Grubb and Chechik described the need for representing the presence of actors and intentions in the time-based simulation of goal models; yet, they did not provide any mechanism to capture, represent, or conduct analysis in accounting for these presence conditions. We continue this line of inquiry by investigating presence conditions in time-based goal models.

Generally speaking, a *presence condition* specifies whether a given model element is present in a model fragment or slice. In model-driven engineering, presence conditions have been used for specifying software product lines [6] and uncertainty [7]. These conditions are generally specified via boolean expressions [6]. TimedURN, another time-based goal modeling approach, specifies presence conditions over model elements through *Deactivation Changes*, which specify a time were an element either becomes active or de-active [3], [8]. Our work differs from these approaches because we specify temporal ranges, rather than boolean conditions.

Illustrative Example: Predictive App (App). Consider a generic application (App), circa 2020. Fig. 1(a) illustrates a fragment of the model, created by the analytics team, who has the goal to Have A Prediction Feature in their application. In the initial model, this goal is decomposed into two possibilities: Use Similarity Metrics for predictions and Create Data Model to make predictions from historic user data. Initially, they compare the trade-offs of these possible designs, wanting to find the best approach to Have a Prediction Feature, while fulfilling Privacy and Ease of Use. Yet, they want to be open to future possibilities. Fig. 1(b) illustrates another fragment, where the team explores future options (expected in 2023) to Use External AI Models. In Fig. 1(b), a new actor called AI Corp has the goal to Have Large Language Model, which the App can use. Yet, given this controversial technology, governments may regulate the use of AI for both the public and private sector. The third model fragment, see Fig. 1(c) illustrates the Regulator actor blocking the use of AI in 2025.



(a) Year 2020, $t_0 = 0$ (b) Year 2023, $t_c = 36$ (c) Year 2025, $t_e = 60$

Fig. 1: Three model fragments of the Predictive App example.

The analytics team wants to simulate possible evolutions of Have A Prediction Feature, given the dependencies and implications of the AI Corp and Regulator actors. Prior work enables analysts to merge these model fragments and show them in the same model [9]. In the resulting simulations, absentee actors are shown with meaningless fulfillment labels for their intentions, which confuses users and leads to erroneous decisions. Thus, we present a novel technique to capture and visualize the presence and absence of actors and intentions, which improves the versatility and interpretability of the time-based simulations in the Evolving Intentions framework.

Contributions. In this MoDRE workshop paper, we incorporate presence conditions into the Evolving Intentions frame-

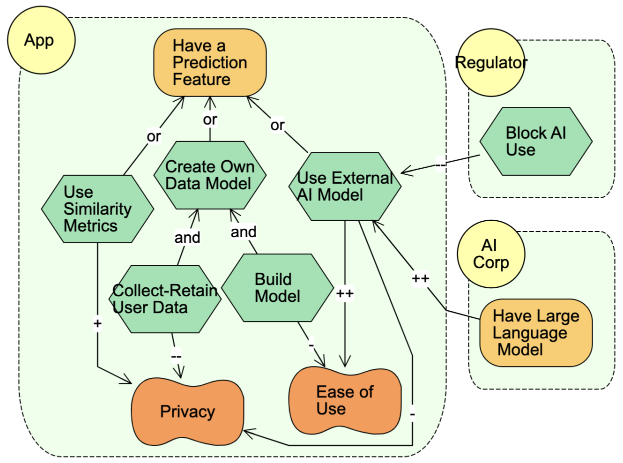


Fig. 2: App model with all fragments from Fig. 1 combined.

work. We enable the automatic analysis and simulation of goal models where some actors and intentions are absent for a portion of the model timeline.

Organization. Section II uses the App example to give an overview of the Evolving Intentions framework. In Sect. III, we describe our specification and implementation of presence conditions. Section IV discusses our evaluation and ongoing work. Section V compares our approach to related work. We conclude in Sect. VI.

II. Background

In the Evolving Intentions framework, an *evolving goal model* is a tuple $M = \langle A, G, R, EF, MC, maxTime\rangle$, where A is a set of actors, G is a set of intentions (e.g., goals, tasks, soft-goals), R is a set of relationships over intentions, EF is a set of evolving functions, MC is a set of constraints over time points in the model, and *maxTime* is the maximum absolute time over which any function or constraint is defined by the user [4].

Fig. 2 shows the model after the three model fragments from Fig. 1 have been merged [9]. Fig. 3 provides the specification of the model in Fig. 2 according to the definition of an evolving goal model M. For example, App's actor boundary in Fig. 2 illustrates the same eight intentions listed for App in the set A of M_{App} (see Fig. 3). Each intention (e.g., Privacy) is listed under the set G with the associated type (e.g., *soft-goal* for Privacy). Similarly, all relationship shown in Fig. 2 are listed as the specification of R (see Fig. 3). For example, the link between Build Model and Ease of Use in Fig. 2 is written as Build Model $\xrightarrow{-}$ Ease of Use in the specification.

Within the framework, the fulfillment of intentions are evaluated using five *evidence pairs*: *Satisfied* $(F, \perp)$, *Partially Satisfied* $(P, \perp)$, *Partially Denied* $(\perp, P)$, *Denied* $(\perp, F)$, and *None*, $(\perp, \perp)$. Additionally, four conflicting evidence pairs may result from propagation: (F, F), (F, P), (P, F), and (P, P). These valuations can be used to specify how the fulfillment of each intention may change over time using step-wise function specified within EF. For example, the EF set in Fig. 3 specifies that Have a Prediction Feature will behave randomly from $t_0 = 0$ to t_{hpf} using the STOCHASTIC function, then will be *Satisfied* $(F, \perp)$ from t_{hpf} to $t_{maxTime}$ as specified by the CONSTANT function. See [4] for a complete overview.

Once the time-based elements of a model are specified, we can generate a *simulation path* of how the model could evolve over time. Thus, the definition of M also includes timing information. Simulation paths are generated from a start time $t_0 = 0$ until a user specified maximum time *maxTime*, where each increment of time (i.e., a *tick*) is specified by the user. In our illustrative example, the modeler created the model for January 2020 and wants to simulate it over seven years were each tick is a month. Thus, the *maxTime* of M_{App} (see Fig. 3) is 84. We also specify time points with which we intent to sample the evaluation of intentions in the model. We include January of each year in the MC set by specifying t_a–t_f (see Fig. 3).

III. Results

In this section, we introduce how we specify presence conditions within the Evolving Intentions framework and then we describe our implementation in BloomingLeaf.

A. Specification of Presence Conditions

As described in Sect. I and Sect. III-B, we allow front-end users to identify presence conditions as the period of inclusion for an element over the timeline. However, to enable backwards compatibility of prior models, we store this information as the *period of exclusion*. We extend the definition of an actor and an intention (see Sect. II) to allow for the specification of an absence of an element.

Actor Definition. Let an actor $a \in A$ be the tuple $\langle name, type, intention\text{-}set, exclusion\text{-}set\ (optional)\rangle$, where *exclusion-set* is a set of closed intervals for which the actor is not present. For example, Fig. 1 (b) illustrates the AI Corp to be present in the model starting at $t_c = 36$. Thus, we update the specification of the AI Corp actor in Fig. 3 as $\langle$AI Corp, $actor$, {Have Large Language Model}, $\{[0, 35]\}\rangle$. Similarly, for Regulator (see Fig. 1 (c)), which appears in the timeline in 2025 (i.e., $t_e = 60$), we update the definition as $\langle$Regulator, $actor$, {Block AI Use}, $\{[0, 59]\}\rangle$.

Intention Definition. Let an intention $g \in G$ be the tuple $\langle name, type, exclusion\text{-}set\ (optional)\rangle$, where *exclusion-set* is a set of closed intervals for which the intention is not present. For example, the Use External AI Model is only present in Fig. 1 (b) and Fig. 1 (c), but is absent from Fig. 1 (a); thus, Use External AI Model should only be present in the timeline at $t_c = 36$. We update the specification of the Use External AI Model intention in Fig. 3 as $\langle$Use External AI Model, $task$, $\{[0, 35]\}\rangle$.

Inclusion/Exclusion Periods. We specify presence conditions as the exclusion period rather than the inclusion period to allow for backwards compatibility of prior models, analysis, and tooling. We acknowledge this deviation from the literature (see Sect. V). In the illustrative example, we do not need to modify the definition of the App actor or the intentions shown in Fig. 1 (a) as they are present throughout the entire timeline.

The goal model M_{App} is $\langle A, G, R, EF, MC, \textit{maxTime} \rangle$ where,

$A = \{\ \langle \text{App}, \textit{actor}, \{\text{Have a Prediction Feature, Use Similarity Metrics, Create Own Data Model, Collect-Retain User Data, Build Model, Privacy, Ease of Use, Use External AI Model}\} \rangle, \langle \text{AI Corp}, \textit{actor}, \{\text{Have Large Language Model}\} \rangle, \langle \text{Regulator}, \textit{actor}, \{\text{Block AI Use}\} \rangle\ \},$

$G = \{\ \langle \text{Have a Prediction Feature}, \textit{goal} \rangle, \langle \text{Use Similarity Metrics}, \textit{task} \rangle, \langle \text{Create Own Data Model}, \textit{task} \rangle, \langle \text{Collect-Retain User Data}, \textit{task} \rangle, \langle \text{Build Model}, \textit{task} \rangle, \langle \text{Privacy}, \textit{soft-goal} \rangle, \langle \text{Ease of Use}, \textit{soft-goal} \rangle, \langle \text{Use External AI Model}, \textit{task} \rangle, \langle \text{Have Large Language Model}, \textit{goal} \rangle, \langle \text{Block AI Use}, \textit{task} \rangle\ \},$

$R = \{\ (\text{Use Similarity Metrics, Create Own Data Model, Use External AI Model}) \xrightarrow{\text{or}} \text{Have a Prediction Feature}, (\text{Collect-Retain User Data, Build Model}) \xrightarrow{\text{and}} \text{Create Own Data Model}, \text{Use Similarity Metrics} \xrightarrow{+} \text{Privacy}, \text{Collect-Retain User Data} \xrightarrow{--} \text{Privacy}, \text{Build Model} \xrightarrow{-} \text{Ease of Use}, \text{Use External AI Model} \xrightarrow{++} \text{Ease of Use}, \text{Use External AI Model} \xrightarrow{-} \text{Privacy}, \text{Block AI Use} \xrightarrow{--} \text{Use External AI Model}, \text{Have Large Language Model} \xrightarrow{++} \text{Use External AI Model}\ \},$

$EF = \{\ \langle \text{Have a Prediction Feature}, \{\langle \textsc{Stochastic}, (\bot, \bot), t_0, t_{hpf} \rangle, \langle \textsc{Constant}, (F, \bot), t_{hpf}, t_{maxTime} \rangle\} \rangle\ \},$

$MC = \{\ t_a = 12, t_b = 24, t_c = 36, t_d = 48, t_e = 60, t_f = 72\ \}$, and

$\textit{maxTime} = 84.$

Fig. 3: Specification of the App model shown in Fig. 2.

The exclusion set can contain more than one time period. For example, to specify that an actor will be present in the middle of the timeline, say from 24 to 48, then the exclusion set would be $\{[0, 23], [49, 84]\}$. Additionally, if an intention belongs to an actor (i.e., is listed in the actor's *intention-set*), then the actor's exclusion period applies to that intention; nevertheless, the intention can have a more restrictive exclusion period. For example, we specified the exclusion set for the Regulator actor as $[0, 59]$. This implies that Block AI Use could have an exclusion set of $[0, 62]$, meaning the Regulator actor is present in the timeline before Block AI Use appears.

Analysis with Presence Conditions. Once presence conditions have been added into model elements, they can be incorporated into analysis. The Evolving Intentions framework allows for path-based simulations over time, where at each time point in the path the model is evaluated (i.e., each intention is assigned an evidence pair) given the evolving functions in EF and propagation of values across the links in the model. When an intention is not present (i.e., during an exclusion period), the intention is excluded from analysis and all links connected to the intention are excluded. When an actor is excluded, all intentions within the actor boundary and their associated links are excluded.

Generally, decomposition links (i.e., and/or) have more than one source intention. For example, Create Own Data Model in Fig. 2 has two source intentions. Suppose Build Model had a presence condition and was excluded from analysis, then the only source intention would be Collect-Retain User Data. In this case the decomposition would effectively propagate as a ++ link (see [4] for further details on propagation).

B. Visualizations and Tooling

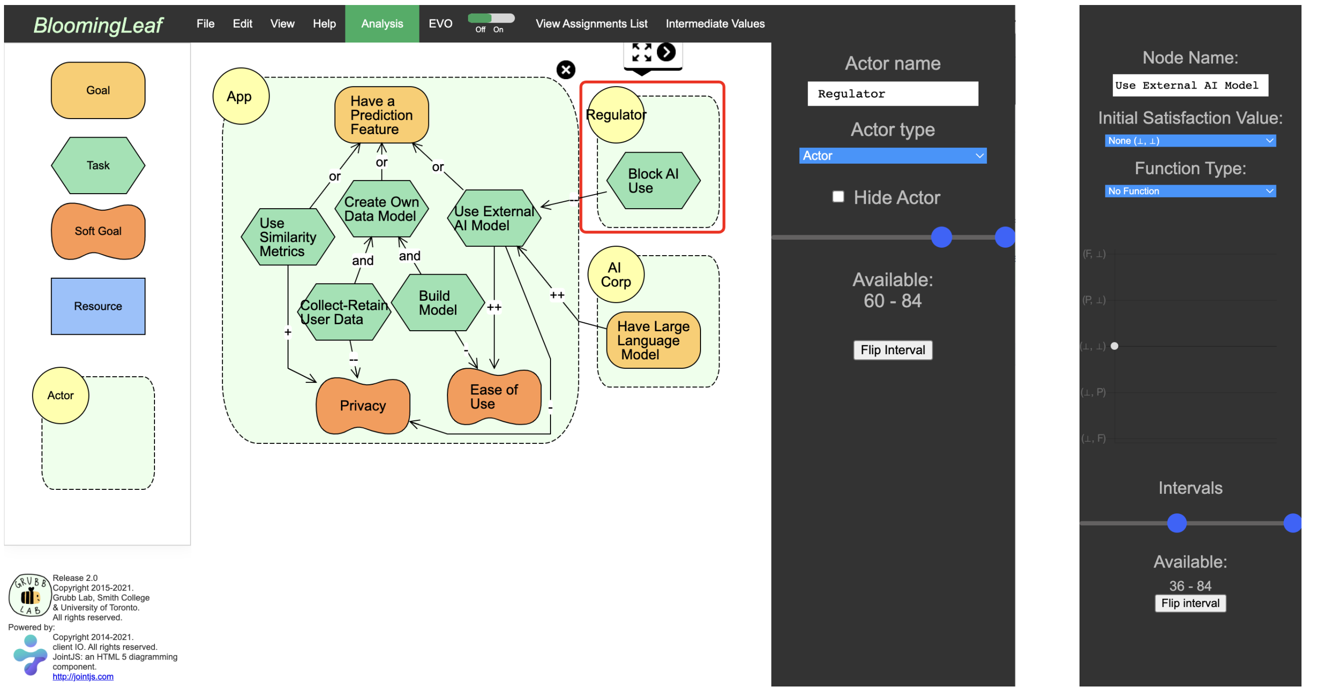
We implemented presence conditions as part of BloomingLeaf, a goal modeling tool that implements the Evolving Intentions framework (see Release 2.6 at *https://github.com/amgrubb/BloomingLeaf/releases/tag/v2.6*). When creating a model, users can specify presence conditions using the actor and intention panels.

Actor Panel. Fig. 4(a) shows BloomingLeaf with the App on the center canvas. Since the Regulator actor is selected, the actor panel appears on the right-hand side. We appended this panel to allow users to specify when the actor is present over the timeline. Recall that *maxTime* for the App model is 84. In Fig. 4(a), the actor slider has been updated to select $60 - 84$, showing when the Regulator is present. As already discussed in Sect. III-A, although the user selects when the actor is present, we store when the actor is excluded.

Intention Panel. Fig. 4(b) shows the intention panel, when Use External AI Model is selected on the center canvas. Again, we allow users to specify when the intention is present over the timeline. In this case, Use External AI Model is present from 60 to 84 (see Fig. 4(b)).

Model-level View. By opting to store presence conditions in the actor and intention panels, users may have trouble keeping track of each assignment or may be confused why the range for an intention is more restrictive than zero to *maxTime*. To increase visibility of presence conditions, we extended the Assignments List pop-up to include presence conditions, which can be accessed by clicking View Assignments List in the top toolbar of BloomingLeaf (see Fig. 4(a) for toolbar icon). The updated assignments window shown in Fig. 5 has the presence conditions for the App model listed. We list all actors, as well as any intentions with ranges that deviate from their actor.

Simulation. Finally, we updated the path simulation viewer in BloomingLeaf to show or hide elements based on their presence conditions. Recall, at each time point in a simulation, evidence pairs are propagated to all intentions in the model based on the evolving functions and evaluation labels assigned by the user. For example, in the App example, at $t_a = 12$ and $t_b = 24$, the model shown in the simulation is the same as what appears in Fig. 1(a) with evidence pair assignments for each intention. While at $t_e = 60$ and $t_f = 72$, the Fig. 2 model is shown with evidence pair assignments. We omit images of this simulation for space considerations.

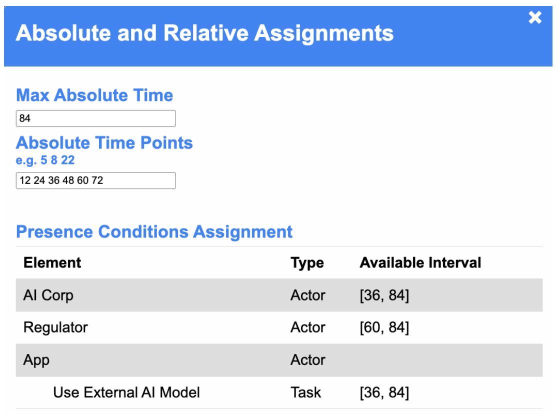


Actor name: Regulator
Actor type: Actor
Hide Actor
Available: 60 - 84

Node Name: Use External AI Model
Initial Satisfaction Value: None (⊥, ⊥)
Function Type: No Function
Available: 36 - 84

(a) BloomingLeaf showing App model on the center canvas. Since the Regulator intention is selected, the Actor panel (with presence conditions) appears on the right. (b) Intention panel when Use External AI Model selected.

Fig. 4: Screenshots of BloomingLeaf showing our presence conditions extensions to the Actor and Intention Panels.



Absolute and Relative Assignments

Max Absolute Time

84

Absolute Time Points

e.g. 5 8 22

12 24 36 48 60 72

Presence Conditions Assignment

Element	Type	Available Interval
AI Corp	Actor	[36, 84]
Regulator	Actor	[60, 84]
App	Actor	
Use External AI Model	Task	[36, 84]

Fig. 5: Assignments window showing *maxTime*, absolute time points and presence conditions for the App example.

IV. Evaluation & Ongoing Work

Next, we describe our initial evaluation and ongoing work.

Evaluation. As described in Sect. III, we implemented presence conditions in the front-end of BloomingLeaf. Using the App example, we demonstrated the feasibility of analysts using BloomingLeaf to specify and simulate a model with presence conditions. As an initial test of the scalability of our implementation, we return to the road construction project introduced in Sect. I. To visualize presence conditions, Grubb and Chechik drew bubbles over fragments of the full model of the road project [5]. We found it easy and fast to add the presence conditions to model elements. Our approach is scalable for this large model (>100 intentions), which we demonstrate with a video online at *https://doi.org/10.35482/csc.004.2023*. We need to complete further validation with large models to confirm scalability. Upon reflection, our implementation can be improved by allowing users to select and apply a single presence condition to a collection of intentions.

Ongoing Work. With the ability to visualize presence conditions in BloomingLeaf, we can improve the backend analysis to take this information into account. Recently, Hablutzel et al. [9] created a semi-automatic algorithm for the manual merge presented by Grubb and Chechik [5]. We aim to extend this merge algorithm, which is implemented in BloomingLeaf, to accommodate presence conditions in the source models to be merged. Doing so involves resolving conflicting information about the presence of actors and intentions.

Additionally, we explore how users interpret and reason about goal model simulations. Using our tooling for presence conditions, we allow users to create a model slice by hiding actors. The actor panel in Fig. 4(a) shows a Hide Actor button that when clicked the actor disappears from the modeling canvas. We envision a similar interface for the analysis view, allowing us to study how users explore large models.

V. Related Work

Presence conditions have been extensively studied in the context of model-driven engineering (MDE) for software product lines (i.e., variability aware systems). Czarnecki and Antkiewicz contributed presence conditions as annotations over model elements to represent variability and evaluate a particular feature configuration [6]. They represented presence conditions as boolean formulas in disjunctive normal form and XPath expressions. In subsequent work, Czarnecki and Pietroszek represented valid configurations of a software product line using presence conditions expressed over features as propositional variables [10].

Based on the feature-based model templates proposed by Czarnecki and Antkiewicz, Voelter and Visser demonstrated the application of domain specific languages as a mapping between feature models and code, within an MDE framework [11]. During the mapping, a model transformation is performed to remove program elements whose presence conditions evaluate to false based on the current feature configuration. See Jézéquel for a survey of variability modeling, including presence conditions [12].

More recently, work has focused on analyzing and simplifying presence conditions. Von Rhein et al. [13] observed that presence conditions can often contain redundant information and investigated approaches to simplifying formulaic representations. Finally, Hentze et al. used presence conditions and SAT-based analysis to quantify the number of possible products that can be derived in a product line [14]. Our treatment of presence conditions for time-based goal model deviates from the traditional notion of a presence condition because we focus on variability within the temporal aspect of a single model rather than possible combinations within the goals, as analogous to features.

We are not the first to consider presence conditions in the context of goal models. Horkoff et al. annotated iStar models with *May* presence conditions to identify and resolve design-time uncertainties [7]. Alwidian et al. annotated Goal-oriented Requirement Language (GRL) models in order to track changes in the model elements over time [15]. While, the representations used by Alwidian et al. differ from our work and that of Czarnecki and Antkiewicz [6], they are still representing variability as the presence of an entity in their union model [16].

Finally, as mentioned in the introduction, we build on the prior work of Grubb and Chechik, which investigated merging models across multiple time periods where not all actors and intentions were present for the entirety of the timeline [5]. In a similar approach, called TimedURN, Aprajita and collaborators enable the evaluation of intentions in GRL models to change over time and link these changes with use case maps [3], [8]. They implemented presence conditions as an instantaneous change, called a *DeactivationChange* for intentions/actors and a *LinkDeactivationChange* for links. For example, a *DeactivationChange* in our App example would be to make AI Corp active at January 2023 ($t_c = 36$). In the case of an actor present from 24 to 48 (see Sect. III-A), then this would be represented as three *DeactivationChanges*, two to hide the intention at 0 and 49, and another to make it appear at 24. While neither approach is better, we chose specifying presence conditions over ranges because it allowed for easier analysis and model slicing within our existing framework.

VI. Summary

In this paper, we described our extension to the Evolving Intentions framework to give stakeholders the ability to specify presence conditions and reason about simulations where one or more actors and intentions were absent for part of the model timeline. We implemented these extensions in BloomingLeaf and demonstrated their usage using an illustrative example of an application adding a prediction feature. We evaluated our implementation using a large model from the literature.

Our ongoing work focuses on incorporating presence conditions with model merging. Using presence conditions as model slices allows us to empirically observe how stakeholders interpret large models. Future work will investigate generalizing this approach to other goal modeling languages.

Acknowledgments. We thank Venus Nguyen for her leadership in tool development. Thanks to Emily Kung, Karenna Kung, Thu Tran, and Caroline Zouloumian for their assistance in tool development. This material is based upon work supported by the National Science Foundation under Award No. 2104732.

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] J. Horkoff, T. Li, F.-L. Li, M. Salnitri, E. Cardoso, P. Giorgini, J. Mylopoulos, and J. Pimentel, "Taking Goal Models Downstream: A Systematic Roadmap," in *Proc. of RCIS'14*, May 2014, pp. 1–12.

[3] Aprajita and G. Mussbacher, "TimedGRL: Specifying Goal Models Over Time," in *Proc. of MoDRE'16*, 2016, pp. 125–134.

[4] A. M. Grubb and M. Chechik, "Formal Reasoning for Analyzing Goal Models that Evolve over Time," *Requirements Engineering*, vol. 26, no. 3, pp. 423–457, 2021.

[5] ——, "Reconstructing the Past: The Case of the Spadina Expressway," *Requirements Engineering*, vol. 25, no. 2, pp. 253–272, 2020.

[6] K. Czarnecki and M. Antkiewicz, "Mapping Features to Models: A Template Approach based on Superimposed Variants," in *Proc. of GPCE'05*, 2005, pp. 422–437.

[7] J. Horkoff, R. Salay, M. Chechik, and A. D. Sandro, "Supporting Early Decision-making in the Presence of Uncertainty," in *Proc. of RE'14*, 2014, pp. 33–42.

[8] Aprajita, S. Luthra, and G. Mussbacher, "Specifying Evolving Requirements Models with TimedURN," in *Proc. of iStar'17*, 2017, pp. 26–32.

[9] K. R. Hablutzel, A. Jain, and A. M. Grubb, "A Divide & Concur Approach to Collaborative Goal Modeling with Merge in Early-RE," in *Proc. of RE'22*, 2022, pp. 14–25.

[10] K. Czarnecki and K. Pietroszek, "Verifying Feature-Based Model Templates against Well-Formedness OCL Constraints," in *Proc. of GPCE'06*, 2006, pp. 211–220.

[11] M. Voelter and E. Visser, "Product Line Engineering Using Domain-specific Languages," in *Proc. of SPLC'11*, 2011, pp. 70–79.

[12] J.-M. Jézéquel, "Model-driven Engineering for Software Product Lines," *International Scholarly Research Notices*, vol. 2012, 2012.

[13] A. Von Rhein, A. Grebhahn, S. Apel, N. Siegmund, D. Beyer, and T. Berger, "Presence-condition Simplification in Highly Configurable Systems," in *Proc. of ICSE'15*, 2015, pp. 178–188.

[14] M. Hentze, C. Sundermann, T. Thüm, and I. Schaefer, "Quantifying the Variability Mismatch between Problem and Solution Space," in *Proc. of MODELS'22*, 2022, pp. 322–333.

[15] S. Alwidian and D. Amyot, ""Union is Power": Analyzing Families of Goal Models Using Union Models," in *Proc. of MODELS'20*, 2020, pp. 252–262.

[16] S. Alwidian, D. Amyot, and Y. Lamo, "Union Models for Model Families: Efficient Reasoning over Space and Time," *Algorithms*, vol. 16, no. 2, 2023.