A Divide & *Concur* Approach to Collaborative Goal Modeling with Merge in Early-RE

Kathleen R. Hablutzel, Anisha Jain, Alicia M. Grubb
Department of Computer Science
Smith College, Northampton, MA, USA
amgrubb@smith.edu

***Abstract*—Goal modeling enables the elicitation of stakeholders' intentionality in the earlier stages of a project. Often, approaches are limited by the effort required to create an initial goal model. In this paper, we investigate the problem of model merging for Tropos goal models. Specifically, we propose a formal approach to the problem of automatically merging the attributes of intentions and actors, once these elements have been matched. Additionally, recent approaches have investigated answering questions about future evolutions of stakeholders' projects with goal models. In this work we consider both static models, as well as those with timing information, using the principles of gullibility, contradiction, and consensus. We study our implementation and validate the merge operation on a variety of models from the literature.**

I. Introduction

Goal-Oriented Requirements Engineering (GORE) aims to help stakeholders make trade-off decisions when planning a project [1], [2]. GORE frameworks also provide analysis capabilities to solve problems facing stakeholders, understand project evolution, and connect their models with downstream development activities [1], [3]. Using GORE approaches in the early-phases of projects, modelers elicit the intentions and dependencies of stakeholders and users within the project domain, and document these requirements using a central visual artifact, called a *goal model* [4]. Modelers can ask trade-off questions of their goal models, and help stakeholders visualize alternatives and make project decisions.

Recent work suggests that given the same initial documentation, requirements analysts focus on different aspects of a project with some leaning toward refinements and others creating new features [5], which suggests that requirements specifications can be improved through the multiple perspectives of different analysts. Others found matching and merging goal models leads to better requirements [6]. Yet, there is a significant cognitive barrier in merging the work of multiple modelers. We aim to automate merging Tropos model syntax and semantics to reduce the burden of merging, and allow modelers to have more focused discussions during elicitations.

State of the Art in Model Merging. Model merging has been extensively studied in the literature. Researchers have proposed approaches for behavior and state-based models [7], [8], [9], [10]. Within GORE, early work by Sabetzadeh and Easterbrook described stakeholder views, in terms of i* models, as annotated graphs [11]. Feng et al. looked at merging decomposition patterns (e.g., and/or decomposition)

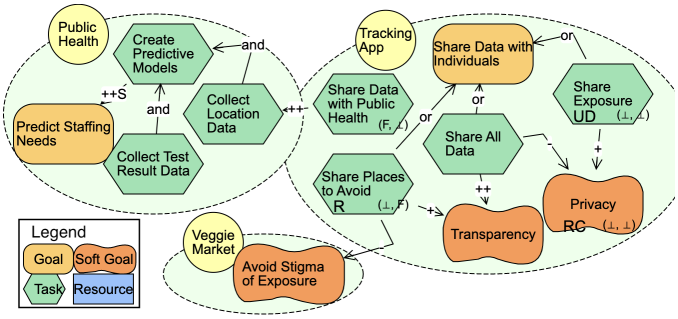


Fig. 1: Model-A showing the perspective of state officials.

in goal models [12]. More recently, Baslyman and Amyot explored merging model fragments into system models within UNR [13]. Peng et al. worked towards building goal models piecemeal by adapting fragments of a model for another domain based on a goal in the subject domain of the model under consideration [14]. Grubb and Chechik proposed a manual process to enable the piecemeal creation of Tropos goal models with evolutionary information [15]. Alwidian and Amyot created a merged representation of families of goal models in URN [16]. The vast majority of this work considers the problem of model merging in terms of matching model elements, creating a reference model, and merging (i.e., a *three-way merge* [17]); thus, the problem of model matching with traceability is well understood. However, once model elements are matched, prior work does not give sufficient intuition into how to resolve conflicts between the attributes of each element in Tropos. In this paper, we extend the work of Grubb and Chechik [15]. In their approach, whenever a conflict is detected, the algorithm prompts the user to manually resolve the conflict. In reality, this approach is repetitive and requires users to interact with every step of the merge process; instead, we aim to automatically merge the majority of model attributes once elements are matched, prompting user involvment only at the end of the merge process in the case of an unresolvable conflict. To illustrate the types of conflicts that may exist in this approach, we introduce a motivating example.

Illustrative Example: Contact Tracing App (CT-App). We consider the example of a COVID-19 contact mobile phone tracing app for a fictitious client CSoft, who is evaluating options and implications of sharing data. Modelers meet with stakeholders to elicit and document their needs. Modeler-A meets with state officials and app developers looking at

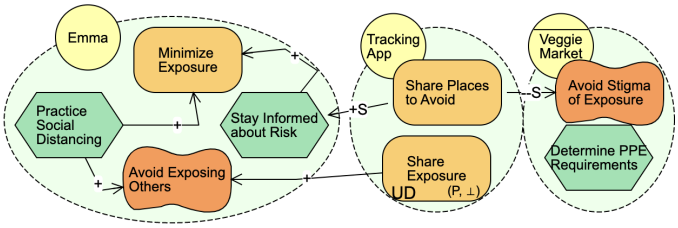


Fig. 2: Model-B showing the views of individual stakeholders.

the technical aspects. Modeler-B wants to understand the benefits to users. Once goal models are constructed (see Fig. 1 and Fig. 2, see also Sect. II for an overview of the notation), CSoft wants to answer the question: "What are the impacts of each of the alternatives for Share Data with Individuals?" Analyzing either model results in incomplete answers because neither model sufficiently describes the intentions of the stakeholders and Tracking App. To fully answer questions in this scenario, we need to merge the models in Fig. 1 and Fig. 2, resulting in the model shown in Fig. 3. Although this is trivial in our CT-App example, this is an arduous task for realistic goal models due to the conflicts among element attributes.

For example, Share Exposure and Share Places to Avoid are modeled as tasks in Model-A (see Fig. 1) and goals in Model-B (see Fig. 2). Share Exposure is assigned an initial valuation of *Partially Satisfied* $(P, \bot)$ in one model, but assigned the value *None* $(\bot, \bot)$ in the other. Both models contain a link from Share Places to Avoid to Avoid Stigma of Exposure, but this is modeled as a $--$ link in Model-A and a $--$S link in Model-B. Finally, Veggie Market is modeled as an actor in Model-A and an agent in Model-B. In the manual algorithm presented in [15], each of these decisions would require prompting the user to make a choice. Other approaches from the literature require this information to be added to the reference model. In order to save time and minimize possible errors, we aim to create an algorithm for goal model merging that automates or streamlines the majority of these decisions.

Contributions. In this technical solution paper, we support modelers' ability to collaborate in GORE activities, specifically the creation of goal models. We are motivated by our central research question: (**RQ**) Given a minimal reference model (i.e., matched element names), to what extent can we automatically merge the remaining attributes to create a single goal model? We investigate static models and those with evolutionary information (see Sect. II for an overview). For static models, element attributes include intention and actor types, relationship types, and intention valuations. Evolving models contain additional attributes for how the valuations of intentions and types of relationships change over time.

We contribute a semi-automated approach for merging Tropos goal models with and without evolutionary information. Our implementation has traceability of the decisions, which enables stakeholders to verify the automated portions of the merge. We demonstrate the effectiveness and scalability of our approach by evaluating it with a variety of models.

Organization. In Sect. II, we introduce relevant goal modeling background, including the manual algorithm from [15].

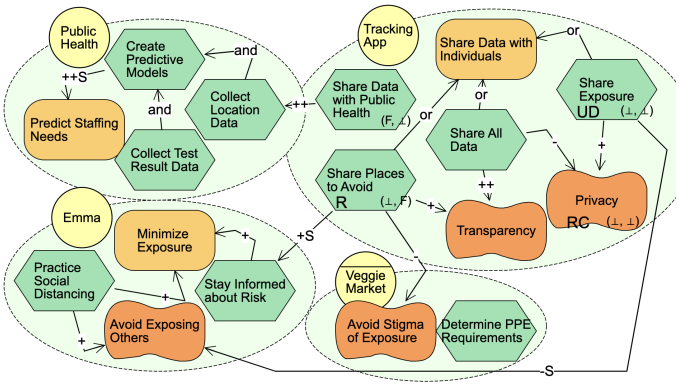


Fig. 3: Results of merging Model-A and Model-B.

Sect. III lays a foundation for our approach and defines the concepts of gullibility and consensus. Sect. IV and Sect. V describes our merge algorithm for static and evolving models, respectively. Sect. VI reports on the effectiveness and scalability of our approach and tooling. Sect. VII connects our approach to related work. We conclude in Sect. VIII.

II. Background

In this section, we review the modeling notation and the manual algorithm we extend and automate.

Goal Modeling. In Tropos, goal models are directed graphs of intentions and links [18], [19]. Intentions may belong to an *actor* and are one of four types: goals, tasks, resources, and soft-goals (see legend in Fig. 1). Intentions can be decomposed, where a node requires the satisfaction of all (and) or one (or) of its children. In Model-A of the CT-App example (see Fig. 1), the top-level *goal* of the Tracking App actor is to Share Data with Individuals, which is or-decomposed into three alternative *tasks*. Intentions can also contribute to each other using contribution links (e.g., +, −, ++, ++S, −−S, etc.).

Each intention 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 an intention, with $\bot \leq P \leq F$. Thus, goals can have one of five values: *Satisfied* $(F, \bot)$, *Partially Satisfied* $(P, \bot)$, *Partially Denied* $(\bot, P)$, *Denied* $(\bot, F)$, and *None* $(\bot, \bot)$; as well as four conflicting values: (F, F), (F, P), (P, F), and (P, P). F [resp. P] means there is full [resp. partial] evidence for or against the fulfillment of an intention, while $\bot$ represents null evidence. In Fig. 1, Share Data with Public Health is assigned by the user the value *Satisfied* $(F, \bot)$ because CSoft has already formed an agreement with public health officials. In the Public Health actor, there is a ++S contribution link between Create Predictive Models and Predict Staffing Needs. The ++S link only propagates when Create Predictive Models has F or P for its s value of the evidence pair. See [18] and [19] for additional propagation details.

Specifying Evolving Information. We use the Evolving Intentions framework [20] to explore merging models with evolution. Evidence pairs in Tropos form a partial order from most satisfied to most denied. In this framework, evolution is specified using step-wise functions (i.e., *evolving functions*),

where over any time interval, the valuations of a intention can INCREASE, DECREASE, remain CONSTANT, or not be constrained by a defined pattern, which is known as STOCHASTIC. In the CT-App example, Modeler-A specifies Share Exposure as STOCHASTIC for a period while the app is being developed, then CONSTANT at Denied $(\perp, F)$ for the next period of time, followed by an INCREASE function up to the value of Satisfied $(F, \perp)$. Fig. 6(a) illustrates this function for Share Exposure.

An *evolving goal model* is a tuple $M = \langle A, G, R, EF, MC, \textit{maxTime} \rangle$, where A is a set of actors, G is a set of intentions, R is a set of relationships, 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 [20].

Prior Manual Merge Algorithm. As part of the Evolving Intentions framework, Grubb and Chechik proposed a manual algorithm for merging goal models with evolutionary information [15]. The procedure consists of six steps: (GC1) Create the timeline for the merged model. (GC2) Update absolute values assigned to the symbolic constants in each model. (GC3) Union merge actors and intentions based on element names. (GC4) Merge relationships in the model. (GC5) Prompt the user to define presence conditions over the model. Update evolving functions for regions with presence conditions. (GC6) Merge evolving functions. This algorithm is a helpful starting point for merging goal models with evolution; however, as demonstrated with the CT-App example in Sect. I, the algorithm does not merge any conflicts in attributes and would instead repeatedly prompt the user to intervene throughout GC3, GC4, and GC6. We extend the algorithm in terms of merging the goal model elements and their attributes (i.e., GC3 and GC4), and merging the evolutionary information in the Evolving Intentions framework (i.e., GC1, GC2, and GC6). Since the goal of this work is to merge models created by different stakeholders, usually over similar time periods, we exclude the consideration of presence conditions (i.e., GC5), which was specific to merging a single modeler's perspective over vastly different time periods where actors did not exist in the time period of some models [15].

The existing portions of the manual procedure [15] are essentially gullible, which means the goal set G of the merged model is equal to the union of the candidate goal sets. Alternatively, a consensus approach would instead take only the goals found in both models—the intersection of their goal sets. A consensus approach defeats the purpose of merging, which is to paste together mostly discrete sub-models into a larger model. In our simplified match, we take the union of goals and actors (based on element name).

With regards to merging evidence pairs and evolving functions, the algorithm essentially uses *gullibility-up-to-contradiction*. This means that for intentions only present in one model, or attributes only given values in one model, the algorithm accepts the values assigned to them. The previous manual algorithm is not yet able to assign values to intentions that have user-assigned values in both source models—values that may disagree. Our strategy for merging element attributes is to use gullibility-up-to-contradiction and then require *consensus*. In the next three sections, we further define the meaning of this strategy and describe how it applies to each kind of attribute, enabling us to automate the merge process.

III. MERGE PROCEDURE & PRINCIPLES

In this section, we give an overview of our approach.

High-level Approach. We list the high-level steps of merge in Algo. 1. *Merge* unifies information from two evolving goal models at the same level of abstraction, specified over two absolute time periods. The inputs to merge are two (evolving) goal models $M_1, M_2 = \langle A, G, R, EF, MC, \textit{maxTime} \rangle$ (see Sect. II). The output of the merge process is an evolving goal model M_M. Our approach also produces a list of deleted elements, and an intermediate list of model constraints to ensure that the timelines are matched appropriately.

The process begins with a pre-computation step (see Lines 1-3 in Algo. 1) whereby we determine the updated timeline, as described in Sect. V-A. Since the model timeline affects the rest of the process, we compute the new timeline and have the user verify its correctness and update any timeline information before proceeding with the rest of the merge process.

To keep our investigation focused on merging element attributes, our approach simply matches intention and actor names to create a union of model elements (see Line 4 in Algo. 1). Other approaches (see Sect. VII) use a reference model to make these connections. We consider the matching portion of the algorithm to be outside the scope of our investigation in this paper. Other matching functions can be inserted into our algorithm, if desired, to match elements with different naming conventions (see Sect. VIII).

Once the new timeline has been created and the actors and intentions are matched, we proceed with merging the remaining attributes in the model. First, on Line 5 of Algo. 1, we merge the actors and their relationships (i.e., attributes of A), as described in Sect. IV-A. Second, we merge the attributes of the intentions on Lines 6-8. We initially merge only the intention types (see Sect. IV-B), then we merge the evolving functions associated with each intention, as described in Sect. V-B and Sect. V-C. Additionally, we resolve any intention valuations not included in evolving functions (see Sect. IV-C). There is no dependency between the ordering of merging actors (Line 5) and intentions (Lines 6-8 in Algo. 1). Finally, we merge intention relationships, which include both static (see Sect. IV-B) and evolving (see Sect. V-D) links (Line 9). After this process completes, the modeler may review each automated decision in the merge algorithm and manually resolve any remaining conflicts.

To merge *static* models, EF and MC are empty sets and *maxTime* is assigned to zero. In this case, only Lines 4, 5, 6, 8, and 9 are required in Algo. 1. All the details required to merge static models are described in Sect. IV.

We map the steps in our merge process with the manual algorithm (see Sect. II). GC1–2 map onto Lines 1–3 in Algo. 1, while GC3 maps onto Lines 4–6, & 8. GC4 is accomplished

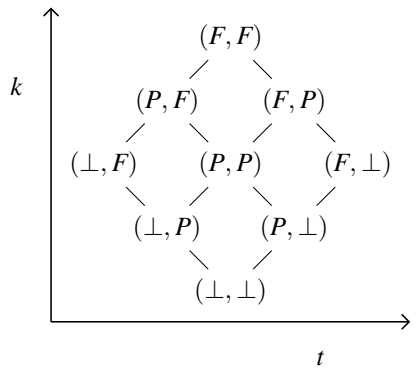


Fig. 4: Bilattice of knowledge (k) and truth (t) orderings of $s \odot d$.

Algorithm 1: High-level Merge Process

Input:
Models $M_1, M_2 = \langle A, G, R, EF, MC, \textit{maxTime} \rangle$ ▷ See Sect. II.
Initial start values: $M_1.\textit{start} = 0$, $M_2.\textit{start}$
Output:
Merged Model $M_M = \langle A, G, R, EF, MC, \textit{maxTime} \rangle$
Log files with deletions list.
Timing file with updated model constraints MC.
1: Pre-compute Model Timeline ▷ See Sect. V-A
2: User verifies timing file and updates any model constraints in MC, R, EF.
3: Update Timeline Information MC ▷ See Sect. V-A
4: Match Actors A, and Intentions G based on element name.
5: Merge Actors & Actor Relationships A ▷ See Sect. IV-A
6: Resolve Intentions Types G ▷ See Sect. IV-B
7: Merge Evolving Functions EF ▷ See Sect. V-B-V-C
8: Resolve Intention Valuations G ▷ See Sect. IV-C
9: Merge Intention Relationships R ▷ See Sect. IV-B & Sect. V-D

by Line 9 and GC6 is completed by Line 7. As mentioned in Sect. II, we do not consider GC5 (i.e., presence conditions).

Our strategy to merge element attributes is to use gullibility-up-to-contradiction and then require consensus. To implement this strategy, we need to first explicitly define what it means for intention valuations to be gullible, or be in consensus.

Defining Gullibility and Consensus for Evidence Pairs. Next, we define operators for gullibility and consensus for evidence pairs used in Tropos. Evidence pairs are used as valuations for intentions (i.e., an attribute of $g \in G$). Additionally, they are used to describe how the valuation of an intention evolves or changes over time (e.g., used in EF). We use these new operators in Sect. IV and Sect. V when merging intention valuations and evolving functions, respectively.

Using the *evidence pair* (s, d) introduced in Sect. II, we create a bilattice of the product of s and d (i.e., $s \odot d$), by considering the knowledge (k) and truth (t) orderings. The set of this bilattice is the Cartesian product $s \times d = \{(\bot, \bot), (\bot, P), (\bot, F), (P, \bot), (P, P), (P, F), (F, \bot), (F, P), (F, F)\}$, which are the values in Sect. II. Its t and k orderings are defined for any two elements (s_1, d_1) and (s_2, d_2) in $s \times d$,

$$(s_1, d_1) \leq_k (s_2, d_2) \text{ iff } s_1 \leq s_2 \text{ and } d_1 \leq d_2$$
$$(s_1, d_1) \leq_t (s_2, d_2) \text{ iff } s_1 \leq s_2 \text{ and } d_2 \leq d_1$$

For example, $(\bot, \bot) \leq_k (\bot, P)$ means that $(\bot, P)$ represents more evidence than $(\bot, \bot)$, while $(\bot, P) \leq_t (\bot, \bot)$ means that $(\bot, \bot)$ indicates more satisfaction/less denial than $(\bot, P)$. If for two values a and b in $s \times d$ it is neither the case that $a \leq_k b$ nor that $b \leq_k a$, then the two values are considered *incomparable* k-wise—(*P*, *F*) and (*F*, *P*) are two such values. The same rule holds in the t ordering. Comparing all pairings in $s \times d$ in this way yields the bilattice pictured in Fig. 4. A lattice is a partially ordered set in which any two elements within it have a greatest lower bound (*glb*), or meet, and a least upper bound (*lub*), or join. With this bilattice, we define $\wedge$ and $\vee$ as the meet/*glb* and join/*lub* of the t-ordering, while $\otimes$ and $\oplus$ are defined as the meet and join of the k-ordering. We describe these operations in terms of the meet ($\sqcap$) and join ($\sqcup$) of each component lattice, s and d [21]. For all (s_1, d_1) and (s_2, d_2),

$$(s_1, d_1) \wedge (s_2, d_2) = (s_1 \sqcap s_2, d_1 \sqcup d_2) \ (\textit{Min-Sat})$$
$$\langle s_1, d_1 \rangle \vee \langle s_2, d_2 \rangle = \langle s_1 \sqcup s_2, d_1 \sqcap d_2 \rangle \ (\textit{Max-Sat})$$
$$\langle s_1, d_1 \rangle \otimes \langle s_2, d_2 \rangle = \langle s_1 \sqcap s_2, d_1 \sqcap d_2 \rangle \ (\textit{Consensus})$$
$$\langle s_1, d_1 \rangle \oplus \langle s_2, d_2 \rangle = \langle s_1 \sqcup s_2, d_1 \sqcup d_2 \rangle \ (\textit{Gullibility})$$

In English, $\wedge$ returns the minimum s value and the maximum d value, while $\vee$ returns the maximum s value and the minimum d value. $\otimes$ returns the minimum s and d values (which we define as *consensus* in this paper), while $\oplus$ returns the maximum s and d values (defined as *gullibility*). Prior work used *Max-Sat* and *Min-Sat* for forward and backward propagation [18], [19]. The evolving functions EF use the t ordering to describe how the valuations of intentions change over time [20]. The bilattice structure in Fig. 4 provides us with two metaphorical and technical approaches to information conflict: consensus, represented by $\otimes$ (i.e., meet over k ordering); and gullibility, represented by $\oplus$ (i.e., join over k ordering).

IV. Merging Static Goal Models

In this section, we describe how, using the principle of gullibility-up-to-contradiction, we merge the attributes in static goal models. At this point we assume all actors and intention elements have been matched.

A. Merging Actors & Actor Relationships

Actors may be assigned one of three types (i.e., Actor, Role, and Agent), and may be connected to each other via actor links (i.e., participates-in or is-a relationship). Using gullibility-up-to-contradiction, if an entity (i.e., actor) exists in only one model, we accept that entity through gullibility. In merging actors, a contradiction may occur in two attributes: (a) a matched pair of actors with different types, or (b) a matched pair of actor links with different link types. Through resolving these contradictions, we may generate an invalid combination of actor type and link type (see Table I for a full list).

In elicitation, some modelers may refine the scenario more than others and we want to take this in to account when

TABLE I: Invalid Actor Links

Link Type	Source	Destination
is-a	Actor	Role
is-a	Role	Actor
is-a	Agent	*any type*
is-a	*any type*	Agent
participates-in	Actor	Agent
participates-in	Role	Agent

merging actor types. Roles are an abstract refinement over actors. Agents are concrete instantiations of an Actor or Role. Since Roles and Agents are refinements of actors, we gullibly accept the maximum type within this refinement relationship:

$$Actor < Role < Agent \tag{1}$$

For example, if two actors have the same name but one is of type Actor and the other is of type Role, then it is acceptable to assign the merged actor as a Role. In the case of different actor link types, we choose to prioritize participates-in over the is-a relationship, because it is a more general relationship, presenting any kind of association (other than generalization) [22].

Completing these steps separately may result in one of the invalid actor relationships listed in Table I; thus, there are two possible solutions to this conflict: (1) prioritize links over actor types, and (2) prioritize actor types over links.

If we prioritize actor links, we update the actor type to result in a valid link. When an invalid relationship is created (see Table I), then we identify the actor type that results in the invalid relationship and revert the actor type of the conflicting element(s) to that of the source model.

If we prioritize actor types, we remove conflicting links. When an invalid link is created (see Table I), then the rule presented in Equation 1 holds and we delete the conflicting link, adding it to the list of deleted entities for the stakeholder to review upon completion of the algorithm.

We use the second option in our implementation because every actor has an actor type, but not all actors are connected with an actor relationship.

B. Merging Intentions & Intention Relationships

Next, we consider merging intentional elements and their relationships. Given the principle of gullibility-up-to-contradiction, if an intention or intention attribute exists in only one model then we gullibly accept this information into the merge model. Once intentions are matched, conflicts may arise between the type of the intention, as well as between the link types that connect intentions. Unlike actors, intentions have no formal restriction between their type and the type of link by which they are connected; thus, we can merge the intention types and relationships separately.

Resolving Intention Types. As introduced in Sect. II, intentions can have one of four types. Unlike actors, there is no refinement relationship between intention types in our goal model language. In this case, we accept the maximum type:

$$Goal < Task < Softgoal < Resource \tag{2}$$

Equation 2 is based on heuristics of how each type is used [23], [22]. We establish this equation considering the possible choices made by modelers. For example, *Goal*s are sometimes considered the default type, meaning that they are chosen in cases where the type is not well articulated, often to be updated later as the model is developed. Resources, on the other hand, are only selected in narrowly defined cases where modelers want to identify resources.

TABLE II: Consensus rules for positive and negative contribution links.

	++	+	++S	+S	++D	+D
++	++					
+	+	+				
++S	++S	+S	++S			
+S	+S	+S	+S	+S		
++D	++D	+D	**++** or **no**	**+** or **no**	++D	
+D	+D	+D	**+** or **no**	**+** or **no**	+D	+D

	−−	−	−−S	−S	−−D	−D
−−	−−					
−	−	−				
−−S	−−S	−S	−−S			
−S	−S	−S	−S	−S		
−−D	−−D	−D	**−−** or **no**	**−** or **no**	−−D	
−D	−D	−D	**−** or **no**	**−** or **no**	−D	−D

Resolving Contribution Relationships. In forward propagation, contribution links define how evidence pairs are propagated via evidence predicates [18]. A contradiction between the contribution types of two links means that there are opposing rules over the evidence predicates in propagation. For example, in forward propagation, a + link propagates partial satisfaction in the source intention as partial satisfaction in the destination intention, where as a − propagates partial satisfaction in the source intention as partial denial in the destination intention. These links are in direct conflict, with no consensus between them, and cannot be resolved. Using the rules of propagation, we defined the meaning of consensus between contribution links for when a contradiction is found.

Consider two generic models: M_1 and M_2. Given a pair of intentions $M_1.g1, M_1.g2$ from the first model and the same pair of intentions $M_2.g1, M_2.g2$ from the second model, there may exist at most two intention relationships $M_1.g1 \xrightarrow{M_1.r} M_1.g2$ and $M_2.g1 \xrightarrow{M_2.r} M_2.g2$. In this scenario there may exist a conflict between the relationship types (i.e., $M_1.r$ and $M_2.r$). If a relationship exists in only one model, we accept that relationship through gullibility. If a relationship exists

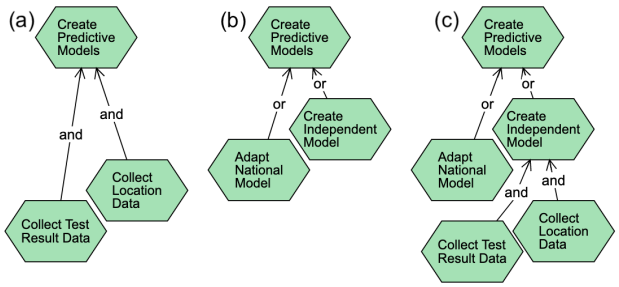


Fig. 5: Model fragments of a conflicting decomposition.

in each model, we use a consensus approach to merge the relationships. If $M_1.r$ equals $M_2.r$, then the relationships are already in consensus. If the signs are opposite (e.g., $M_1.r$ = + and $M_2.r$ = −), then there is an unresolvable conflict. In this case, the relationship is kept, but the type is changed to the binary type no, which indicates to the user that a conflict must be resolved in the relationship type without deleting the entire link. For two relationships with the same sign (e.g., $M_1.r$ = +S and $M_2.r$ = ++), consensus gives us the minimal operator both links agree on. We list these combinations for the positive and negative relationships in Table II. Merging a +S link with a ++ link results in a +S link.

An interesting property emerges with respect to merging a 'S' and 'D' relationship with the same sign. For example, as listed in Table II, +S and +D have no consensus in propagation, so they should result in a no relationship. However, given that they do not conflict, a semi-gullible approach would allow for merging them into a + link, which we implement, see Table II.

Resolving Decomposition Relationships. Decomposition (i.e., n-ary) relationships are used to refine intentions into subcomponents (and) or alternatives (or). Each intention can have at most one incoming n-ary relationship type. Merging results in a conflict when a target intention becomes connected via both and and or links. Conflicts between n-ary relationship types cannot be resolved automatically and require modeler intervention. In many cases, further elicitation and specification is required. For example, in Fig. 5(a), Create Predictive Model is decomposed into Collect Test Result Data and Collect Location Data using and, but is also or decomposed into Adapt National Model and Create Independent Model in Fig. 5(b), resulting in a conflict. Choosing one of the decomposition types will not result in a meaningful model. Instead, after the merge algorithm is complete, the modelers may update the resulting model according to the needs of stakeholders, which may require further elicitation. In the case of the model in Fig. 5(c), an intermediate intention is manually added by the user to resolve this conflict. Thus, we leave all source intentions connected and change the n-ary relationship type to no.

C. Merging Intention Valuations

Finally, we merge intention valuations, in terms of evidence pairs. Recall that we introduced the gullibility $\oplus$ and consensus $\otimes$ operators for evidence pairs in Sect. III. When two intentions are matched, but only one is assigned an evidence pair, we gullibly accept the assignment. When both intentions are assigned an evidence pair, we determine the consensus between the two values by applying the $\otimes$ operator.

Again consider two generic models M_1 and M_2 that contain the intention g. The value assigned to intention g in M_1 is defined as $M_1.g_v$, and in M_2 it is $M_2.g_v$. Then the merged value, $M_M.g_v$, will be $M_1.g_v \otimes M_2.g_v$. In English, the sat and den values assigned to g in the merged model will be the minima of the sat and den values assigned to g by the source models. These are the maximum satisfaction and denial values about which we can be reasonably certain, since the two models agree that at least that level of evidence is present.

For example, if $M_1.g_v = (\bot, F)$ and $M_2.g_v = (\bot, P)$, then $M_M.g_v$ will be $(\bot, F) \otimes (\bot, P) = (\bot, P)$. M_1 and M_2 agree that there is no evidence indicating that g is satisfied; so, the same holds true in M_M. M_1 and M_2 disagree about the exact amount of evidence indicating the g is denied, but they agree that there is at least partial evidence; thus, M_M will indicate that there is partial evidence against g's satisfaction.

Using $\otimes$, the merged model is more likely to result in intentions with a valuation of None $(\bot, \bot)$. Instead, a gullibility approach (i.e., $\oplus$) results in conflicting intention valuations (e.g., (P, F)). While conflicting values are built into the language, in practical terms, they are intended to be the result of analysis. Given the intent of the merge operation is to create a large model for further elaboration and analysis, an approach that results in these conflicting valuations is less optimal. Instead, we use a consensus approach to merge valuations.

V. Merging Goal Models with Evolution

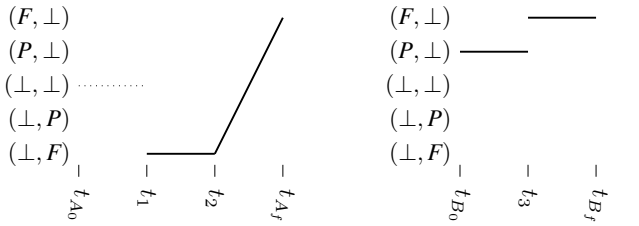
In this section, we describe how, using the principle of gullibility-up-to-contradiction, we merge the evolutionary elements in the model. Specifically, we demonstrate how to merge multiple timelines and how to resolve conflicts between evolving functions and between relationships.

A. Merging Timeline Information

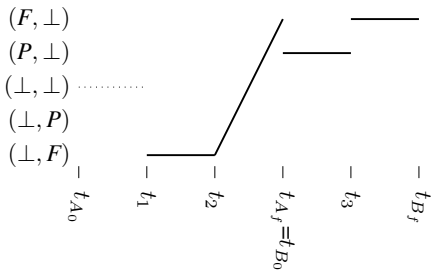
When two models are merged, they may be defined over different time intervals. Before we can merge the evolutionary information for each element, we require a consistent timeline between them. The proposed manual algorithm in [15] assumed that the models were defined over separate time scales. We relax this constraint to allow for models defined over the same time period. Note that both models must use the same scale (i.e., semantic meaning of a tick in time) to give the resulting merged model any real-world meaning.

As inputs to the algorithm, the user specifies absolute *start* time points for each model based on the real-world values. The start time for one of the models must be zero. For each model, we define *stop* = *start* + *maxTime*. Using these values, the timeline for the merged model is defined as $M_M.start = 0$ and $M_M.stop = \max(M_1.stop, M_2.stop)$.

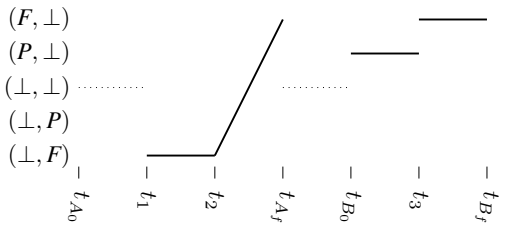
With a merged timeline, we can update absolute values assigned to the symbolic constants in each model. Symbolic constants are used in MC, EF, and R (see Sect. II). Prior to merging, all absolute assignments to symbolic constants were



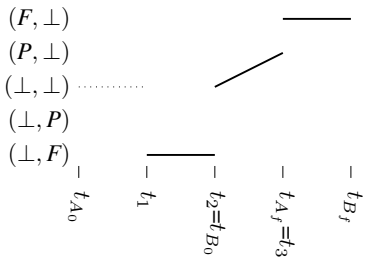
(a) Function from Model-A (b) Function from Model-B



(c) Case 1: The timelines are contiguous.



(d) Case 2: There exists a gap between the timelines.



(e) Case 3: The timelines overlap.

Fig. 6: Proposed evolving functions from Model-A and Model-B for Share Exposure in the CT-App example, and the resulting merged function given the three possible timelines.

specified relative to 0 and *maxTime* of the source models. We update any absolute time assignments to make them relative to the *start* and *stop* times for M_M. Finally, we assign the *maxTime* value for the merged model to be M_M.*stop*.

B. Merge Evolving Function Timelines

As introduced in Sect. II, evolving functions are step-wise atomic functions over disjoint neighboring intervals, where the atomic functions are CONSTANT, INCREASE, DECREASE, and STOCHASTIC. Suppose we are merging the functions for Share Exposure, which we describe in Fig. 6. In Model-A, see Fig. 6(a), Share Exposure is defined by a STOCHASTIC function until t_1, followed by a period of *Denied* $(\perp, F)$ (CONSTANT) and then an INCREASE function. In Model-B (see Fig. 6(b)), Share Exposure is defined by two CONSTANT functions, assigning the intention a *Partially Satisfied* $(P, \perp)$ value until t_3 followed by a *Satisfied* $(F, \perp)$ value. Specifying the resulting model depends on the underlying timeline over which each model is defined. For example, if $t_{A_f} < t_{B_0}$ (see Fig. 6(a) and Fig. 6(b)), then there exists an unspecified gap in the function. If $t_{A_f} > t_{B_0}$ then there exists an overlap in the function, which may result in a conflict.

Let us consider two generic models, M_1 and M_2, and assume that both models contain an evolving function for intention g. The timeline for g is defined in both models such that there is a total order over the time points used to define g's evolution. g's timeline in model M_1 covers the time interval $[M_1.t_0, M_1.t_f)$. In model M_2, g's timeline covers $[M_2.t_0, M_2.t_f)$. There are three possibilities for the interactions between these intervals:

- Case 1: $M_1.t_f$ and $M_2.t_0$ are simultaneous, meaning that the timelines are contiguous.
- Case 2: $M_1.t_f$ occurs before $M_2.t_0$, meaning that there is a gap between the timelines.
- Case 3: $M_1.t_f$ occurs after $M_2.t_0$, meaning that the timelines overlap (and may overlap completely).

Cases 1 and 2 are straightforward because there is no points at which the timelines actively disagree. We propose that the merge algorithm continue the gullibility-up-to-contradiction approach and simply accept both timelines. In Case 1, the timelines are contiguous; thus, g's timeline in the merged model M_M is the timeline in M_1, followed by its timeline in M_2. An example of this case for Share Exposure is shown in Fig. 6(c), where the source timelines are copied into the merge timeline verbatim. In Case 2, the merged timeline contains a gap. Thus, g's timeline in the merged model will begin with its timeline in M_1, then an empty interval is added via a STOCHASTIC function, followed by the timeline from M_2. This case is shown for Share Exposure in Fig. 6(d), where a STOCHASTIC function is added over the period $[t_{A_f}, t_{B_0})$.

Case 3 is more challenging in part because there are many ways for the timelines in M_1 and M_2 to overlap. It may be that only the terminal and initial intervals, for M_1 and M_2 respectively, overlap; or their functions may overlap across multiple intervals. In the latter case, the user must specify which intervals overlap through the timing file we generate, see Line 2 of Algo. 1 discussed in Sect. III. In the next subsection, we describe how to merge individual conflicting atomic functions. If more than one interval overlaps, each interval is considered separately. Following our policy of gullibility-up-to-contradiction, the non-overlapping intervals on the timeline are included verbatim in the merged model. For example, in Fig. 6(e), the function definitions for Share Exposure overlap in the interval $[t_2{=}t_{B_0}, t_3{=}t_{A_f})$. Thus, the functions from M_1 are copied verbatim over the interval $[t_{A_0}, t_2)$ and the last function from M_2 is copied for the final interval $[t_3, t_{B_f})$. We explain the function for the overlapping interval in the next subsection.

TABLE III: Rules for merging individual atomic evolving functions for a given intention g between two generic models M_1 and M_2. *Note that the order of M_1 and M_2 is interchangeable. If M_1 has an* INCREASE *function and M_2 has a* CONSTANT *function, the resulting model M_M will use the rule on Line 5 with the model numbers reversed.*

	Model M_1 $M_1.g_x - M_1.g_y$	Model M_2 $M_2.g_x - M_2.g_y$	Merged Model M_M $M_M.g_x - M_M.g_y$			
1	STOCHASTIC	STOCHASTIC	STOCHASTIC			
2	CONSTANT	CONSTANT	CONSTANT	$M_1.g_x \otimes M_2.g_x$		
3	INCREASE	INCREASE	CONSTANT	$M_1.g_x \otimes M_2.g_x$	iff	$M_1.g_x \otimes M_2.g_x = M_1.g_y \otimes M_2.g_y$
			INCREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_y \otimes M_2.g_y$		
4	DECREASE	DECREASE	CONSTANT	$M_1.g_x \otimes M_2.g_x$	iff	$M_1.g_x \otimes M_2.g_x = M_1.g_y \otimes M_2.g_y$
			DECREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_y \otimes M_2.g_y$		
5	CONSTANT	INCREASE	CONSTANT	$M_1.g_x \otimes M_2.g_x$	iff	$M_1.g_x \otimes M_2.g_x = M_1.g_y \otimes M_2.g_y$
			INCREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_x \otimes M_2.g_y$		
6	CONSTANT	DECREASE	CONSTANT	$M_1.g_x \otimes M_2.g_x$	iff	$M_1.g_x \otimes M_2.g_x = M_1.g_y \otimes M_2.g_y$
			DECREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_x \otimes M_2.g_y$		
7	INCREASE	DECREASE	CONSTANT	$M_1.g_x \otimes M_2.g_x$	iff	$M_1.g_x \otimes M_2.g_x = M_1.g_y \otimes M_2.g_y$
			INCREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_y \otimes M_2.g_y$	iff	$M_1.g_x \otimes M_2.g_x <_t M_1.g_y \otimes M_2.g_y$
			DECREASE	$M_1.g_x \otimes M_2.g_x - M_1.g_y \otimes M_2.g_y$	iff	$M_1.g_y \otimes M_2.g_y <_t M_1.g_x \otimes M_2.g_x$
8	STOCHASTIC	CONSTANT	CONSTANT	$M_2.g_x$		
9	STOCHASTIC	INCREASE	INCREASE	$M_2.g_x - M_2.g_y$		
10	STOCHASTIC	DECREASE	DECREASE	$M_2.g_x - M_2.g_y$		

C. Merge Conflicting Evolving Functions

As introduced above, over any individual time interval, if the evolving function for an intention is specified in one input model but not specified in the other, we gullibly accept the specified function. When matched intentions have evolving functions defined in both models, we need to resolve any contradiction. Recall from Sect. II that there are four types of change in the valuation of an intention over an interval, called atomic evolving functions. In order to create generic rules that apply to any time interval, we first update the definition of an atomic function by adding a reference value for the start of the interval. With two reference values we determine the consensus of the start and end of the interval and discern the updated atomic function accordingly. We list all combinations of merging atomic evolving functions in Table III.

Throughout this discussion, we consider two generic models, M_1 and M_2, and assume that both models contain the intention g, which has evolving functions defined over the interval $[t_i, t_f)$. The resulting merge function depends on the function type from each of the source models, M_1 and M_2.

Updated Atomic Function Definitions. The lack of a starting value in the INCREASE and DECREASE functions makes the intervals vague. If a user enters an INCREASE interval with a maximum value of *Satisfied* $(F, \perp)$ followed by a DECREASE interval with a minimum value of *Denied* $(\perp, F)$ for intention g, presumably they mean that g becomes fulfilled during the first interval, then is unfulfilled. Yet a situation where g stayed at *None* $(\perp, \perp)$ throughout follows this pattern. In a timeline that begins with an INCREASE or DECREASE function, it is unclear from where the valuation of g is increasing or decreasing. These ambiguities are theoretically problematic and cause technical problems during merging.

We propose that the INCREASE [resp. DECREASE] functions now contain two reference points, g_x, a closed starting value, and g_y, an open maximum [minimum] value. For an INCREASE function $g_x <_t g_y$ must hold, and $g_x >_t g_y$ must hold for a DECREASE function. This revised definition better corresponds with their intended meanings, as the valuation of g must actually increase or decrease by at least one step during the interval, and the initial valuation of g is no longer unclear when the interval occurs at the beginning of a goal's timeline. The initial and final value of a CONSTANT function is already known and is the same as the reference point, thus $g_x = g_y$.

Merging Reference Values for Functions. Before we merge the types of the atomic functions, we first find the consensus among the reference evidence pairs. Since we updated the definitions for the atomic functions, each interval contains a starting evidence pair g_x, and an ending evidence pair g_y. Thus, we use the $\otimes$ operator defined in Sect. III, which determines the consensus between two evidence pairs. In each of these cases, $M_M.g_x = M_1.g_x \otimes M_2.g_x$ and $M_M.g_y = M_1.g_y \otimes M_2.g_y$, meaning that the new starting and ending reference pair for the merged model are the meet (or minimum available knowledge) from each of the input models.

Functions with the Same Source Type. With updated reference values, we determine the resulting atomic function. We start with the cases where both source models have the same function type—both M_1 and M_2 define g as evolving with the same function type. If the type is STOCHASTIC, nothing further is needed (see Line 1 in Table III); otherwise, we use $\otimes$ to find a consensus from the given reference values. If both source models, M_1 and M_2, contain a CONSTANT function, then the resulting model M_M will contain a CONSTANT at $M_1.g_x \otimes M_2.g_x$ (see Line 2 in Table III). If both models contain an INCREASE [resp. DECREASE] function, then in the merge model M_M, g will have an INCREASE [resp.

DECREASE] function from $M_M.g_x$ to $M_M.g_y$ (see Line 3 [resp. Line 4] in Table III). In the rare case that the new starting and ending values are equal (i.e., $M_M.g_x = M_M.g_y$) for an INCREASE or DECREASE function, we change the function type to be CONSTANT (see again Lines 3 and 4).

Resolving Different Source Types. Next, we consider the cases where source models have different types. If the function type in M_1 is different from the function type in M_2, we must consider their reference points first. As mentioned above, the resulting starting and ending reference evidence pairs for the merged model M_M will be $M_M.g_x = M_1.g_x \otimes M_2.g_x$ and $M_M.g_y = M_1.g_y \otimes M_2.g_y$. Using these values, we determine the resulting functions for each combination (see Lines 5-7 in Table III). If $M_M.g_x = M_M.g_y$, then the merged function type for g is CONSTANT. Similarly, if $M_M.g_x < M_M.g_y$, then the type is INCREASE, and if $M_M.g_x < M_M.g_y$, then the type is DECREASE. When the function for g is CONSTANT in one of the models, the resulting function for g in the merged model will typically have the function type from the other model, although in a minority of cases it will be CONSTANT.

The scenario where the function for g is STOCHASTIC in M_1 and is a different type in M_2 has two possible solutions. First, the merged model could be assigned a STOCHASTIC function because we do not know how g evolves in M_1 during that period; thus, the consensus between the models is unknown (i.e., STOCHASTIC). Second, the merged model could be assigned the function from M_2 because a STOCHASTIC function is equivalent to no data. This is supported by gullibility-up-to-contradiction to accept the only definite information about g, that is, the information provided by M_2. We choose the second approach, where the non-stochastic function dominates (see Lines 8-10 in Table III).

Resolving the Merge Conflict in the CT-App Example. In Fig. 6(e), Share Exposure is specified as having a conflict in the overlapping interval [t_2=t_{B_0}, t_3=t_{A_f}). In Model-A, Share Exposure is specified by an INCREASE function from *Denied* to *Satisfied*: $M_A.g_x = (\perp, F)$ and $M_A.g_y = (F, \perp)$. In Model-B, Share Exposure is specified by a CONSTANT function at the level of *Partially Satisfied*: $M_B.g_x = M_B.g_y = (P, \perp)$. Applying the $\otimes$ operators to each of the g_x and g_y values gives the merged model the reference values $M_M.g_x = (\perp, \perp)$ and $M_M.g_y = (P, \perp)$. From Line 5 of Table III, we see that the resulting function is INCREASE, shown in Fig. 6(e).

D. Merging Evolving Links

Finally, we merge evolving relationships between intentions. In Sect. IV-B, we merged static links and considered two generic links: $M_1.g1 \xrightarrow{M_1.r} M_1.g2$ and $M_2.g1 \xrightarrow{M_2.r} M_2.g2$. The intuition for resolving evolving relationships is the same as for static links. The main difference between static and evolving links is that for evolving links one or both of $M_1.r$ and $M_2.r$ may include evolutionary information or be "Not Both" links (see [20] for details). Since evolving relationships are limited to two time intervals, we merge each interval separately using the rules described in Sect. IV-B. These rules apply for contribution and decomposition links. If there is a conflict where the link cannot be resolved (i.e., mixing decomposition types), we change the link type to ⊘ and document the conflict for the user to resolve after the completion of the algorithm. If both $M_1.r$ and $M_2.r$ are evolving links, there may also be a conflict between their defined intervals. In this case, we create a new symbolic constant to represent the merged interval, assigning a new time point for this transition.

Not Both links are unique to the Evolving Intentions framework and used exclusively to describe an evolutionary decision. They have a limited use case and appear with low-frequency in the models in the literature. In our approach, if a Not Both link is merged with any other link or evolving function and creates a conflict, the Not Both link is removed; otherwise, we preserve it in the merged model.

VI. VALIDATION & TOOLING

In this section, we validate the effectiveness of our merge algorithm and implementation. To support the independent verification of our work, we provide a replication package: *https://doi.org/10.35482/csc.001.2022*

Tooling. Our implementation of Algo. 1 is a Java program that takes as input two models created using BloomingLeaf (Rel. 2.0) and the timing offset between start times. The output is a model that can be loaded into BloomingLeaf for visualization. First, *PreMerge* takes the two models and the timing offset as input to examine intentions with ambiguous evolving function timelines. Cases where intentions have multi-segmented overlapping evolving functions are outputted to a timing file for the user to update (see Lines 1-2 in Algo. 1). Second, *Merge* takes the two models and the timing file as input and outputs the merge of the two models and a traceability file detailing the source models for each element, deleted elements, and conflict messages (see Lines 3-9 in Algo. 1).

Validation. We collected a set of models from the GORE literature, listed in Table IV. Models 1-8 were collected from a study by Cebula et al., where, for a given scenario, one model was created by study subjects and one model was created by researchers [24]. We updated the names of some elements in Models 1-8 to enable matching (e.g., self vs. student). Models 9-14 were taken from other papers in the literature [15], [20], [25], [26], [27]. To establish ground truth, we asked eight non-author researchers in our lab, called *inspectors*, to manually merge together the models for each scenario. Inspectors were trained modelers familiar with evolving goal models but had no information about the proposed merge algorithm. Each scenario had up to two inspectors, and each inspector reviewed three to four scenarios. When only one model existed, inspectors created a second model from the scenario.

Table IV lists counts of actors, intentions, and relationships for the source models (and their matches) and the merged model, as well as evaluation criteria. We successfully merged all models. The Conflicts/Deletions column of Table IV lists the number of elements that must be updated by the user after merge completes. With values less than five, we greatly

TABLE IV: Validation Study Data. Correctness & Completeness Scale: All, Almost all, Most, Half, Some, Few, None.

Model	Model A				Model B				Matched		Merged				Conflicts/	Evaluation		
Value	A	G	R	EF	A	G	R	EF	A	G	A	G	R	EF	Deletions	Correctness	Completeness	Runtime (*ms*)
1. S1	2	17	16	0	2	20	21	0	2	6	2	30	36	0	0	All	Almost All	2
2. S2	2	18	16	0	6	28	33	0	1	8	7	38	45	0	1	Almost All	Almost All	2
3. S3	3	19	20	0	2	29	44	0	2	6	3	42	60	0	1	Almost All	Almost All	2
4. S4	1	16	17	0	2	22	31	0	1	8	2	30	46	0	0	All	All	2
5. S5	2	15	18	0	3	24	26	0	2	4	3	35	43	0	0	Almost All	All	2
6. S6	1	16	19	0	2	29	56	0	1	3	2	42	74	0	0	All	All	2
7. S7	4	17	13	0	3	24	37	0	2	3	5	38	50	0	0	Almost All	All	2
8. S8	3	30	37	0	2	17	16	0	2	8	3	40	50	0	1	Most	Almost All	2
9. Bike-lanes	0	29	26	19	0	22	22	8	0	10	0	41	42	25	0	Almost All	Almost All	4
10. GRAD	3	19	20	10	2	14	14	9	2	13	3	20	25	13	0	Almost All	All	5
11. Scheduler	3	18	13	9	3	14	13	8	3	7	3	25	25	14	0	Most	Almost All	4
12. WME	2	14	11	10	2	18	23	13	2	10	2	21	31	15	2	Most	Almost All	5
13. Spadina-plan-pro	5	43	34	18	5	28	15	13	2	4	8	67	47	31	0	All	Almost All	3
14. Spadina-pro-opp	5	28	15	13	6	37	28	16	3	10	8	54	41	25	1	Almost All	Almost All	4

reduced the effort required by users over the approach in [15], even for large-scale models (see Models 13-14 in Table IV). Common conflicts included matched intentions belonging to different actors in the source models and unresolvable decomposition link types.

Models 13 and 14 in Table IV were created using the three scenario models (i.e., plan, pro, and opp) from [15]. We found merge to be associative by comparing two versions of the Spadina model: (1) a merge between the plan and the pro model and (2) a merge between the pro and the opp model.

Correctness: For any given pair of models M_1 and M_2 and a set of matched elements in those models, a correct merge operation is one that produces a series of attribute merge answers for which an inspector produces the same results. Table IV lists the inspectors' ratings of correctness, using a common rubric of how many decisions were in agreement: all, almost all, most, half, some, few, none. As shown in Table IV, most models were rated as *Almost All*. Inspectors expected all links to be merged and deemed no links as incorrect.

Completeness: For any given pair of models M_1 and M_2 and a set of matched elements in those models, a complete merge will produce all *reasonable* merges of the model attributes, in the resulting model M_M, where reasonable is defined by inspectors expectations of which attributes should be merged. Using the same scale as correctness, inspectors answered the question, "for all the attributes you merged manually, how many did the automated procedure merge?". From the results listed in Table IV, all models were found to be complete or almost complete. Both correctness and completeness were measured relative to the inspectors' manually created models without knowledge of the merge algorithm. Incomplete models were the result of inspectors' expectations that the procedure would remove redundant links and add additional links.

Scalability: Algo. 1 terminates and has a theoretic worst case runtime of $\mathcal{O}(n^4)$, where n is the number of intentions in M_M. This runtime is dominated by the match operation (Line 4 in Algo. 1), where for every intention $g \in M_1.G$, we loop over every intention $g \in M_2.G$ and then loop over every link $r \in M_2.R$ (where $|R| < n^2$) for each matched intention. We list runtimes measured on a 2.7GHz Quad-Core Intel i7 processor (16GB RAM) in Table IV. With runtimes $<$10ms, our tool is scalable for human-in-the-loop operations.

Threats & Limitations: There is a risk that the inspectors, as lab members, were biased in-favor of our approach. Our initial validation of effectiveness is limited by our inability to resolve all merge conflicts. We used a coarse rubric for inspectors to evaluate correctness and completeness. Future work can mitigate these threats through empirical validation with more precise metrics.

VII. Related Work

As introduced in Sect. I, software and model merging has been extensively studied in the literature. Merge is one of a few operators, identified by Bernstein [28], to manage and maintain design models. In most cases, merge is performed by combining two models using a reference model (i.e., a *three-way* merge [17]).

Early work can be found within the view points literature [29]. Richards investigated matching view points within UML models [30]. Sabetzadeh and Easterbrook proposed a framework for merging *views*, by describing connections between models as annotated graphs [11]. They demonstrated their work by merging i* goal models. Khatwani et al. completed a replication of the work by Sabetzadeh and Easterbrook and found that view point merging, with i* models, leads to better requirements, and that merge traceability focused discussions [6]. In our approach, we take this once step further by resolving inconsistencies between model attributes.

The Model Driven Engineering community has made a concerted effort to generalize model management operators. Brunet et al. generalized definitions of model operators, including merge, based on algebraic properties [31]. Dam et al. built on this work to merge models with arbitrary syntactics given a strict meta-model [32]. More recently, Schultheiß et al. proposed a solution to n-way model matching, which would enable merging n-models [33]. In future work, we can investigate incorporating our merge procedure as a domain specific operator for merging attributes.

Within GORE, Feng et al. considered merging decomposition patterns (e.g., and/or) in goal models [12]. In our work

we do not merge contradicting decomposition links. Feng et al.'s work provides insights for resolving these conflicts in the future. Baslyman and Amyot investigated merging for the purpose of evaluating the impacts of adopting new technology, by merging UNR model fragments describing new features into preexisting project models [13]. In later work, Alwidian and Amyot created a merged representation of goal model families, called *union models*, which enabled tracking model element changes as design-time artifacts and automating analysis across multiple versions of a scenario [16]. In many respects, Alwidian and Amyot built on the work of Sabetzadeh and Easterbrook [11], but across both the time and space dimensions. In this paper, we look at the initial creation of a goal model through merge, rather than tracking the models evolution over time through a merged representation.

Grubb and Chechik proposed a manual procedure for the piecemeal creation of models from different time periods [15]. We extend their work to automatically merge model attributes. Peng et al. looked at building goal models piecemeal by adapting fragments of a model from another domain based on a goal in the subject domain of the model under consideration [14]. This work differs from our investigation in that it merges a model fragment with a single candidate goal; nonetheless, their model repository may be beneficial for evaluating our work.

Finally, La Rosa et al. investigated merging business process models [9], which offers insights for improving element matches. Dhaouadi et al. used an uncertainty framework to connect goal models with Bayesian networks [34]. This work differs from our approach in that it manually connects elements of different model types rather than merging models of the same type; hence, a complementary approach.

VIII. Conclusions, Limitations, and Future Work

In this paper, we presented a semi-automated approach for merging Tropos goal models with and without evolutionary information. We return to our central research question: given a minimal reference model (i.e., matched element names), to what extent can we automatically merge the remaining attributes to create a single goal model? Using a minimal reference model, we automatically merge most model attributes. For the few that we are unable to merge automatically (e.g., decomposition), we report these required updates to the user. Our approach is complementary to the state of the art in model merging, as we propose a novel approach to merging attributes in goal models, once elements have been matched. Our implementation allows for the traceability of each decision, which enables stakeholders to verify the automated portions of the merge. We demonstrated the effectiveness and scalability of our approach on a variety of models from the literature.

Limitations. Since we used the Evolving Intentions framework [20], our approach is limited to this version of Tropos. This approach could be adapted to other GORE languages (e.g., i*, URN), which have the same types for actors and intentions. The main difference between languages is the link types and intention valuations; thus, our approach can be applied with the addition of rules for the language specific link types and valuations. As introduced in Sect. II, the manual merge algorithm originally contained a mechanism for denoting presence conditions in the model. Since the approach was originally created for merging models across vastly different time scales, we did not include the investigation of presence conditions in this paper and leave this as potential future work.

Future Work. We intend to connect our approach with an automatic layout algorithm to improve the resulting visualization of the merged model. Once this is complete, we will conduct further empirical validation as described in Sect. VI. As discussed in Sect. III, our merge operation is based on matching element names. In future work, we will explore replacing Line 4 in Algo. 1 with various state of the art matching algorithms (e.g., NLP-based approaches) to merge elements with different naming systems based on semantic meaning.

Other future work includes implementing presence conditions and exploring language specific versions of our approach as described above. To improve users' confidence in the algorithm, we could provide automated proofs of correctness for each merge operation and facilitate the post-hoc review of merge decisions by adding a visualization for the output logs. Finally, future work could explore using *learning approaches* to enable our algorithm to resolve more conflict types and give users suggestions on how to update the model.

Acknowledgments. We thank Isabel Montesanto and Yilin Lucy Wang for their early contributions to this work, as well as the other researchers in our lab who participated as inspectors in our validation. This material is based upon work supported by the National Science Foundation under Award No. 2104732, as well as the Smith College STRIDE Scholarship program.

References

[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented Requirements Engineering: An Extended Systematic Mapping Study," *Requirements Engineering*, vol. 24, no. 2, pp. 133–160, Jun 2019.

[2] A. van Lamsweerde, "Goal-oriented Requirements Engineering: A Guided Tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[3] 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.

[4] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating Goal Models Within the Goal-Oriented Requirement Language," *Int. J. of Intelligent Sys.*, vol. 25, no. 8, pp. 841–877, 2010.

[5] S. Debnath, P. Spoletini, and A. Ferrari, "From ideas to expressed needs: an empirical study on the evolution of requirements during elicitation," in *Proc. of RE'21*, 2021, pp. 233–244.

[6] C. Khatwani, X. Jin, N. Niu, A. Koshoffer, L. Newman, and J. Savolainen, "Advancing viewpoint merging in requirements engineering: a theoretical replication and explanatory study," *Requirements Engineering*, vol. 22, no. 3, pp. 317–338, 2017. [Online]. Available: https://doi.org/10.1007/s00766-017-0271-0

[7] M. Sabetzadeh, S. Nejati, S. Easterbrook, and M. Chechik, "A Relationship-Driven Framework for Model Merging," in *Proc. of MiSE'07*, 2007.

[8] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 54–64.

[9] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman, "Business Process Model Merging: An Approach to Business Process Consolidation," *ACM Transactions on Software Engineering Methodologies*, vol. 22, no. 2, Mar. 2013. [Online]. Available: https://doi.org/10.1145/2430545.2430547

[10] S. Ben-David, M. Chechik, and S. Uchitel, "Merging partial behaviour models with different vocabularies," in *International Conference on Concurrency Theory*. Springer, 2013, pp. 91–105.

[11] M. Sabetzadeh and S. Easterbrook, "View merging in the presence of incompleteness and inconsistency," *Requirements Engineering*, vol. 11, no. 3, pp. 174–193, 2006.

[12] Z. Feng, K. He, R. Peng, J. Wang, and Y. Ma, "Towards merging goal models of networked software." in *Proceedings of the Twenty-First International Conference on Software Engineering & Knowledge Engineering*, 2009, pp. 178–184.

[13] M. Baslyman and D. Amyot, "Goal model integration: Advanced relationships and rationales documentation," in *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0*, P. Fonseca i Casas, M.-R. Sancho, and E. Sherratt, Eds. Cham: Springer International Publishing, 2019, pp. 183–199.

[14] Y. Peng, B. Li, J. Wang, and Z. Liu, "An approach of crossover service goal convergence and conflicts resolution," in *2020 IEEE World Congress on Services (SERVICES)*, 2020, pp. 225–230.

[15] A. M. Grubb and M. Chechik, "Reconstructing the past: the case of the Spadina Expressway," *Requirements Engineering*, vol. 25, no. 2, pp. 253–272, 2020. [Online]. Available: https://doi.org/10.1007/s00766-019-00321-0

[16] S. Alwidian and D. Amyot, ""Union is Power": Analyzing Families of Goal Models Using Union Models," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS '20)*, 2020, pp. 252–262.

[17] T. Mens, "A State-of-the-Art Survey on Software Merging," *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 449–462, May 2002.

[18] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models," *Journal of Data Semantics*, no. 1, pp. 1–20, 2003.

[19] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and Minimum-Cost Satisfiability for Goal Models," in *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'04)*, 2004, pp. 20–35.

[20] A. M. Grubb and M. Chechik, "Formal Reasoning for Analyzing Goal Models that Evolve over Time," *Requirements Engineering*, 2021.

[21] M. Fitting, "Bilattices and the Semantics of Logic Programming," *Journal of Logic Programming*, no. 11, pp. 91–116, 1991.

[22] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 Language Guide," *arXiv:1605.07767*, 2016.

[23] M. Santos, C. Gralha, M. Goulao, J. Araújo, A. Moreira, and J. Cambeiro, "What is the Impact of Bad Layout in the Understandability of Social Goal Models?" in *Proc of RE'16*, 2016, pp. 206–215.

[24] N. Cebula, L. Diao, and A. M. Grubb, "A Preliminary Investigation of the Utility of Goal Model Construction," in *Proceedings of the 13th International i* Workshop*, 2020, pp. 67–72.

[25] B. C. Hu and A. M. Grubb, "Support for User Generated Evolutions of Goal Models," in *Proc. of MiSE'19*, 2019, pp. 1–7.

[26] A. M. Grubb and M. Chechik, "Looking into the Crystal Ball: Requirements Evolution over Time," in *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*, 2016.

[27] R. Salay, M. Chechik, J. Horkoff, and A. D. Sandro, "Managing Requirements Uncertainty with Partial Models," *Requirements Engineering*, vol. 18, no. 2, pp. 107–128, Jun. 2013.

[28] P. A. Bernstein, "Applying model management to classical meta data problems." in *CIDR*, vol. 2003, 2003, pp. 209–220.

[29] B. Nuseibeh, J. Kramer, and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 760–773, 1994.

[30] D. Richards, "Merging Individual Conceptual Models of Requirements," *Requirements Engineering*, vol. 8, no. 4, pp. 195–205, 2003.

[31] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A Manifesto for Model Merging," in *Proceedings of the 2006 International Workshop on Global Integrated Model Management*, ser. GaMMa '06. Association for Computing Machinery, 2006, pp. 5–12. [Online]. Available: https://doi.org/10.1145/1138304.1138307

[32] H. K. Dam, A. Egyed, M. Winikoff, A. Reder, and R. E. Lopez-Herrejon, "Consistent Merging of Model Versions," *Journal of Systems and Software*, vol. 112, pp. 137–155, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121500134X

[33] A. Schultheiß, P. M. Bittner, L. Grunske, T. Thüm, and T. Kehrer, "Scalable n-way model matching using multi-dimensional search trees," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021, pp. 1–12.

[34] M. Dhaouadi, K. M. B. Spencer, M. H. Varnum, A. M. Grubb, and M. Famelis, "Towards a Generic Method for Articulating Design-time Uncertainty," *Journal of Object Technology*, vol. 20, no. 2, 2021.