



GRLMerger: an automatic approach for integrating GRL models

Nadeen AlAmoudi¹ · Jameleddine Hassine^{1,2} · Malak Baslyman¹

Received: 5 June 2023 / Accepted: 12 January 2024 / Published online: 4 March 2024
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2024

Abstract

Goal-oriented requirements engineering aims to describe both stakeholders and system goals and their relationships using goal models. Large goal models for complex systems are often constructed from sub-models describing various stakeholders' views and context-related aspects. These goal-oriented sub-models, also called views, may exhibit overlaps and present discrepancies. Hence, integrating such views is considered a significant barrier to the construction of a unified goal model. Current approaches to merging goal models require intensive human intervention. This paper proposes an approach and a prototype tool, called *GRLMerger*, to integrate two GRL (Goal-oriented Requirement Language) models into one consolidated model that is correct, complete, and free from any conflict that may arise during the merging process. *GRLMerger* considers both syntactic and semantic aspects of the GRL models and allows analysts to merge them either interactively or in a fully automated mode. *GRLMerger* employs natural language processing (NLP) techniques to match intentional elements based on their semantic similarity. The proposed *GRLMerger* approach and tool have been validated using 12 experimental tasks derived from two case studies, exhibiting very promising performance.

Keywords Goal-oriented modeling · Goal-oriented Requirement Language (GRL) · Integration · Matching · Merging · Semantics · Natural language processing

1 Introduction

Stakeholders' requirements are continuously growing which increases the systems' size and complexity [33]. Requirements models have emerged as a promising solution to cope with software complexity by modeling various software aspects at several levels of abstraction. Requirements models are often constructed incrementally by describing several system aspects, e.g., behavioral, functional/non-functional aspects, and merging partial viewpoints into a unified view,

while resolving any conflicts that may arise. In practice, several versions of a given requirement artifact are built [27] in order to cope with new or changing business needs, new technological advances, etc. In addition, partial models (also called sub-models) may be developed by different teams within an organization. A partial model reflects the viewpoints of the team who developed it [4, 5]. Moreover, these sub-models are built gradually to help grasp the problem domain [11]. Later, these partial models have to be merged to obtain a complete and comprehensive model. However, this task is faced with many challenges including choosing an appropriate level of granularity, conflicting stakeholders' intentions, the usage of different vocabularies, the presence of inconsistent viewpoints, and semantic flaws [4, 5, 40].

A common starting point in requirements engineering (RE) activities is the elicitation of stakeholders' high-level goals and intentions regarding the targeted system. Goal models are meant to describe both stakeholders and system goals and their relationships. Over the years, numerous goal modeling languages have been developed. Some of the famous and widely used ones are *i** [46], Keep All Objects Satisfied (KAOS) [41], the Non-Functional Requirements (NFR) Framework [7], Tropos [14], and the Goal-Oriented

✉ Jameleddine Hassine
jhassine@kfupm.edu.sa

Nadeen AlAmoudi
g201906430@kfupm.edu.sa

Malak Baslyman
malak.baslyman@kfupm.edu.sa

¹ Information and Computer Science Department, King Fahd University of Petroleum and Minerals, PO.Box 1621, 31261 Dhahran, Kingdom of Saudi Arabia

² Interdisciplinary Research Center for Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, PO.Box 1621, 31261 Dhahran, Kingdom of Saudi Arabia

Requirements Language (GRL) [19] part of the ITU-T's User Requirements Notation (URN) standard. Large goal models are often constructed from sub-models describing various stakeholders' views and context-related aspects. These goal-oriented sub-models, also called partial views, may exhibit overlaps and present discrepancies. Hence, integrating such views is considered a significant barrier to the construction of a unified goal model [40].

Merging goal models is essential when introducing new stakeholders, integrating new technologies in a business environment [2], maintaining existing business process models [11], updating current software products [3], and integrating quality aspects, into existing goal models [24, 25]. Although some approaches have been proposed to integrate goal models [2, 12, 13, 24, 39, 40], goal model integration still presents many serious challenges such as the usage of different vocabularies, the presence of stakeholders conflicts, the lack of traceability, the presence of inconsistent viewpoints, and the emergence of semantic flaws [5, 40]. Furthermore, most of the goal model integration techniques are manual and require heavy human intervention, which makes them error-prone and hence reduces their adoption.

Therefore, the ultimate goal of this research is to develop an approach to automate the integration of partial goal models, ensure the correctness and completeness of the integrated model, and mitigate any conflict that may arise among the merged partial views. In this research, the Goal-Oriented Requirements Language (GRL) [19] was used as our target goal-oriented language given its status as an international standard [19].

In this paper, we make the following contributions:

1. Propose the *GRLMerger* approach to merge two GRL models automatically. Merging goal models is necessary not only when building a unified goal model in the early elicitation of stakeholders' goals, but also when maintaining goal models. *GRLMerger* considers both syntactic and semantic aspects of the GRL models. *GRLMerger* employs NLP (Natural Language Processing) techniques to match the GRL constructs based on their semantic similarities.
2. A prototype tool, named *GRLMerger* after the proposed approach. The tool offers modelers/maintainers/analysts the possibility to merge GRL models either in a full-automated way or in an interactive way, giving the user more control over the integration decisions. The tool is published publicly as a Python package.¹
3. Validate the proposed *GRLMerger* approach using three experiments. A total of 24 TGRL models (derived from two case studies) were used in 12 integration cases, and

evaluated in terms of correctness, completeness, and freeness from errors.

The remainder of this paper is organized as follows. Section 2 provides the necessary background for this research with a brief introduction to the Goal-oriented Requirement Language (GRL) [19] and its textual version TGRL notation. Section 3 presents existing work related to merging goal models. Section 4 describes the proposed *GRLMerger* approach, while Sect. 5 presents the *GRLMerger* tool. In Sect. 6, we evaluate the proposed approach and tool using three experiments. An overview of the limitations of *GRLMerger*, some practical insights to help practitioners apply and adjust *GRLMerger* to their needs, and a discussion of the potential threats to validity are provided in Sect. 7. Finally, conclusions and future work are presented in Sect. 8.

2 Research background

This section introduces the basic background of this research. It is organized around three main themes: (1) model management in the context of model integration, (2) the Goal-oriented Requirements Language (GRL) and its textual representation, and (3) the techniques to compute text similarity.

2.1 Model management

Software systems are usually described using informal, semi-formal, or formal models [27]. Models are used to describe software systems at various levels of abstraction. They constitute software artifacts from the problem space (where they capture the requirements of the system under development) to the solution space (where they specify the design, development, and deployment of the final software product) [4]. Examples of models include, among others, control flow diagrams, ontologies, object diagrams, business process models, and form definitions [29]. In practice, these models are developed by different modelers and describe partial views of the system under development. Furthermore, multiple versions of a given model may be developed and maintained due to the changing nature of requirements and the inclusion of various stakeholders' viewpoints. Hence, consolidating such models is essential [29] to maintain a consistent view of the modeled system [27].

Model management aims to find the relationship between models in a systematic way [35]. This can be achieved by a set of algebraic operators [4, 29, 35]. These operators are generic as they can be applied to different problem domains [29]. However, their implementation is tailored to the model's type and applications [37]. Some of the major model management operators




¹ <https://pypi.org/project/GRLMerger/>.

are *Match*, *Merge*, *Diff*, *Extract*, *Transform*, and *Compose* [28–30].

In the context of model integration, given two or more input models, the *Match* operator aims to discover the relationships between the models' elements and creates a mapping between them. More specifically, it establishes a correspondence between similar/equivalent elements of the input models. Such elements are then fed to the *Merge* operator along with the input models, resulting in a merged model with no duplication of the matched elements. Other useful operators for model integration include: the *Compose* operator, which composes two or more element mappings, the *Diff* operator which takes as input two models and finds the differences between them (if any), the *Extract* operator returns a portion of a model that participates in a mapping, and the *Transform* operator copies the elements that have not been matched with the other elements into the merged model. Besides the major operators, there are other operators that can support them in solving model evolution and integration, such as *Split*, *Slice*, *Patch*, and *Propagate* [29].

2.2 Goal-oriented Requirements Language (GRL)

Goal-oriented Requirements Language (GRL) is a visual modeling language used for capturing and representing the goals and requirements of socio-technical systems [11, 26]. The primary goal of GRL is to help stakeholders of a system define and understand its high-level goals and requirements in a structured and systematic way [9, 26]. In addition, GRL enables stakeholders to identify and analyze the trade-offs between different requirements alternatives, which is particularly important when dealing with complex systems having competing objectives. The GRL language has been standardized by the International Telecommunication Union (ITU-T), as part of the User Requirements Notation (URN) [19].

In what follows, we introduce the main GRL constructs using a simple GRL model of a tiny online shopping business. This example will be used throughout the paper to illustrate various model integration configurations. The GRL model, built using the jUCMNav [20] tool and illustrated in Fig. 1, is composed of one GRL actor *Business Owner*. Actors (illustrated as ) represent the stakeholders who are involved in the interaction with the system. They can be human or non-human entities. The *Business Owner* encloses many intentional elements of different types. For instance, *Offer Online Shopping*, *Ensure Authentication*, and *Provide Identification* are GRL goals (illustrated as ) stating the desired outcomes of the actor, while *Increase Sales* and *Have System Security* are GRL softgoals (illustrated as ) representing non-functional requirements or quality attributes.

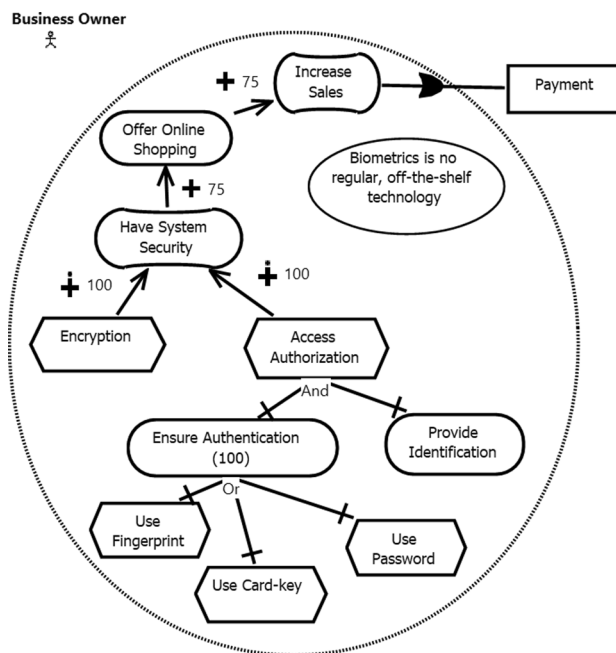






Fig. 1 GRL model: online shopping business

Intentional elements are linked to each other using decompositions (illustrated as ) dependencies (illustrated as ) and contribution links (illustrated as ). For instance, goal *Ensure Authentication* is decomposed, using an OR-decomposition, into three tasks (illustrated as ) describing the actions required to achieve a particular goal, softgoal or another task, namely, *Use Fingerprint*, *Use Card-key*, and *Use Password*. In addition, task *Access Authorization* is decomposed, using AND-decomposition, into two goals *Ensure Authentication* and *Provide Identification*. An AND-decomposition is used when all child-elements shall be satisfied in order to satisfy the parent goal. An OR-decomposition is used when at least one of the child-elements shall be satisfied in order to satisfy the parent goal, while an XOR-decomposition requires the fulfillment of only one of the alternatives.

Intentional elements can be connected using contribution links expressing the impact (positive or negative, at different levels of sufficiency) of the source element on a target element. The qualitative contribution types of a contribution link can fall into one of the following categories [19]:

- Make (\dagger): The contribution is positive and sufficient.
- Help (\ddagger): The contribution is positive but not sufficient.
- SomePositive (\ddagger): The contribution is positive, but the extent of the contribution is unknown.

- Unknown (?): There is some contribution, but the extent and degree (positive or negative) of the contribution is unknown.
- SomeNegative (−): The contribution is negative, but the extent of the contribution is unknown.
- Break (⊖): The contribution of the contributing element is negative and sufficient.
- Hurt (↔): The contribution is negative but not sufficient.

A contribution link may also have a quantitative weight (e.g., an integer value within $[-100, 100]$). For instance, task *Encryption* contributes positively (i.e., Make, +100) to softgoal *Have System Security*, while goal *Offer Online Shopping* contributes positively (i.e., SomePositive, +75) to softgoal *Increase Sales*.

Dependencies are used to describe that one element is dependent (dependor) on another element (dependee) to achieve its objectives. In Fig. 1, softgoal *Increase Sales* depends on resource *Payment*. Resources (illustrated as \square) represent physical or informational entities required to satisfy goals/softgoals. Beliefs are conditions or design rationales that can be attached to intentional elements, e.g., *Bionics is no regular, off-the-shelf technology*.

Actors and intentional elements may have a qualitative (e.g., High, Medium, Low, None) or quantitative (value between 0 and 100) importance. In Fig. 1, the goal *Ensure Authentication* has a *High* importance (shown as (100)).

In addition to its graphical notation, Abdelzad et al. [1] introduced a programming-like textual representation of GRL, called TGRL, in order to facilitate the analysis of GRL models and improve their usability and scalability [1]. TGRL is now part of the URN standard [19]. Figure 2 presents the TGRL specification that corresponds to the GRL model of Fig. 1. In this research, we implement our *GRLM-erger* prototype tool for TGRL. For a complete description of the GRL language, interested readers are referred to the URN standard [19].

2.3 Text similarity

Natural language processing (NLP) has emerged as a powerful tool for text similarity to solve many problems, such as information retrieval, answering questions, document classification, text clustering, and text summarization [10, 31]. Text similarity can be categorized into lexical-based similarity and semantic-based similarity [15, 31].

2.3.1 Lexical-based similarity

The lexical-based similarity operates on the order of characters to decide whether two words are similar or not regardless of their meanings [15, 31]. That is, it is concerned with the syntax only [31]. The lexical-based similarity can be categorized into character-based similarity and term-based similarity [15, 31, 42].

- *Character-based similarity* A character-based similarity measure calculates the similarity between two texts or documents based on the occurrence and frequency of shared characters or n-grams (sequences of n characters).
- *Term-based similarity* The algorithms under the term-based similarity category compare two texts or documents based on the occurrence and frequency of shared terms or words [42].

2.3.2 Semantic-based similarity

The semantic-based similarity identifies the similarity between two texts or documents based on their semantic meaning, rather than just the occurrence of shared words or characters [31]. For instance, the two words 'gift' and 'present' are semantically similar and this can be concluded by the semantic-based similarity measures. Since the lexical-based similarity measures calculate the similarity on a literal level [42], they will not identify 'gift' and 'present' as similar due to different spellings. The semantic-based similarity measures can be categorized into corpus-based and knowledge-based [15, 31].

- *Corpus-based similarity* Corpus-based similarity algorithms use a large set of written or spoken texts, called corpus, to determine the similarity between words [15].
- *Knowledge-based similarity* The knowledge-based similarity methods use a semantic network, such as WordNet, to find the semantic similarity between words [31]. WordNet is a large lexical database of English where it groups nouns, verbs, adjectives, and adverbs into sets of cognitive synonyms called synsets [32].

One way to compute semantic-based similarity is by using distributional semantic models, such as word embeddings or sentence embeddings [38]. These models represent words or sentences as dense, low-dimensional vectors in a continuous space, where similar words or sentences are located close to each other. To measure the semantic similarity between two texts, their word or sentence embeddings are first calculated, and then a similarity score is computed based on the distance or angle between the vectors. A vector is the numerical representation of a word or a set of words in a space such that the

```

gri OnlineShopping {
  actor businessOwner {
    name = "Business Owner";
    softGoal increaseSales {
      name = "Increase Sales";
    }
    softGoal systemSecurity {
      name = "Have System Security";
    }
  }
  goal onlineShopping {
    name = "Offer Online Shopping";
  }
  goal ensureAuthentication {
    name = "Ensure Authentication";
    decompositionType = or;
    importance = high;
  }
  goal provideIdentification {
    name = "Provide Identification";
  }
  goal accessAuthorization {
    name = "Access Authorization";
    decompositionType = and;
  }
  task encryption {
    name = "Encryption";
  }
  task fingerprint {
    name = "Use Fingerprint";
  }
  task password {
    name = "Use Password";
  }
  task cardkey {
    name = "Use Card-key";
  }
  belief biometric {
    name = "Biometrics belief";
    description = "Biometrics is no regular,
      off-the-shelf technology";
  }
  onlineShopping contributesTo increaseSales {somePositive};
  systemSecurity contributesTo onlineShopping {somePositive};
  accessAuthorization contributesTo systemSecurity {make};
  encryption contributesTo systemSecurity {make};
  accessAuthorization decomposedBy ensureAuthentication,
    provideIdentification;
  ensureAuthentication decomposedBy fingerprint, password,
    cardkey;
  increaseSales dependsOn payment;
  resource payment {
    name = "Payment";
  }
}

```

Fig. 2 TGRL specification: online shopping business

distribution of these numbers conserves the textual meaning [23]. To measure the similarity between vectors (i.e., embedding), the most common similarity metric is to find the cosine of the angle between two vectors [21]. It can be computed as follows:

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{|v_1| * |v_2|} \quad (1)$$

Word embedding models do not perform well in representing the meaning of a full sentence [8]. To overcome the issues of the word embedding models, several state-of-the-art sentence embedding models were proposed. Some of the latest state-of-the-art sentence embedding models are InferSent [8], Universal Sentence Encoder [6], and SBERT (Sentence Bidirectional Encoder Representations from Transformers) [38]. However, SBERT outperformed InferSent and Universal Sentence Encoder. SBERT is a modification to the pre-trained BERT network that creates the embedding of sentences with semantic meaning using Siamese and triplet

network architectures [38]. The sentence embeddings can be compared using cosine similarity as well. In this paper, we use SBERT as our Word embedding model.

3 Related work

Several approaches have been proposed to integrate goal models [2, 3, 13, 16, 26, 39]. Sabetzadeh and Easterbrook [39] proposed an algebraic framework to merge early i^* [46] views. Their approach was inspired by categorical algebra and treated early i^* views as structured objects, and the relationships between them as structural mappings. A three-way merge was used to build a consolidated view that contains their common parts. The authors presented general algorithms that use partially ordered sets (i.e., posets) to merge typed graphs. The authors claimed that the poset-annotated graphs, are capable of modeling incompleteness and inconsistency graph-based views, while ensuring full traceability with the partial views. A tiny i^* example was for illustration. Hence, the applicability of the approach to real i^* models remains untested. Furthermore, the approach is fully manual, making it error-prone and limiting its adoption.

Feng et al. [13] presented an approach for merging semantic similar goal models of networked software systems. The authors [13] used the Role-Goal-Process-Service (RGPS) framework [43] to represent functional goals, where each goal is described using (1) a verb to indicate the operation, (2) a noun that indicates the object with which the operation deals, and (3) the manner, a prefix or a suffix that indicates how operation manipulates/impacts the object. Their approach is triggered when there exists a merging point which is specified by having an overlap point (i.e., a pair of goals having semantic similarity) of the root elements of two models. In addition, the authors define three basic merging patterns, namely, AND-AND, OR-OR, and AND-OR patterns, to merge atomic goal models. The approach suffers from many limitations: (1) the merging is done manually and requires analyst judgment to ensure consistency and correctness, (2) no merging is performed in case of an overlap of non-root elements, (3) only AND/OR goal refinements were considered, other goal modeling constructs such as dependencies, and contributions links (present in goal-oriented notations like i^* [46] and GRL [19]) are not considered, and (3) to resolve AND-OR merging conflict, goals may be duplicated. Contrary to this approach, our proposed *GRLMerger* approach resolves conflicts in decomposition types without duplicating the child elements.

In the context of new technology integration in business environments, Baslyman and Amyot [2] proposed

an integration method to merge GRL models with different contexts. The method starts with experts identifying similarities and dissimilarities between the input models to be merged. The similarities are added to a similarity integrated goal model (SIGM), while the dissimilarities are added to a dissimilarity integrated goal model (DIGM). Traceability between SIGM, DIGM, and the input models are preserved using seven relationships, namely, S (Similar), TS (Transitive Similarity), DS (Dissimilar), C (Conflict), D (Different), N (New), and A (Approved). Finally, the integrated model is checked for consistency using OCL (Object Constraint Language) rules. In the last step, the analyst investigates the fully integrated goal model, and resolves any inconsistencies. Although the approach enables consistency analysis through OCL verification, it requires analyst intervention in all steps.

Contrary to the above-mentioned studies that focus on the integration of goal models of the same type, Beckers et al. [3] proposed a method trying to integrate SI* goal-based models with software problem frames. The former captures stakeholders' goals, roles, tasks, and resources, while the latter describes the software development problems. The mapping of the elements in the SI* models to elements of the problem frame notation requires the intervention of a human expert, since it is not a one-to-one mapping and there may exist many options in problem frame elements that are mapped to one element from the SI* model. The proposed method can be used to find relations between goal-based and problem-based models, but the integration between them is not meaningful.

Liu et al. [26] presented the idea of combining reviews with a goal model aiming to support the evolution of mobile applications. First, the relationships between goals and reviews are analyzed and the relevancy between them is computed by comparing their contents. For this phase, an extensible vocabulary for each goal is established considering different terminologies used in the reviews to help establish a strong relationship; the output is a goal model GM with reviews (R-GM). Then, the user sentiments from reviews are introduced to the goal model to help developers determine which goals to be updated. This study emphasized the importance of considering different terminologies used for describing a goal.

The recent work by Hablutzet et al. [16] is the most closely related work to our proposed *GRLMerger* approach. The authors [16] proposed a semi-automated approach for merging Tropos [14] goal models. They defined two operators, *gullibility* and *consensus*. The former moves actors and intentions, if they exist in one model only, to the merged model, while the latter combines similar actors or intentions in both models into one element in the merged model. Their approach merges the actors with their relationships first and resolves any contradiction;

then, the intentions with their relationships are merged and then resolve any contradictions. Some of the contradiction cases such as different types of merged actors are resolved automatically and others are left to the user to make a suitable decision. Our *GRLMerger* approach merges the links (i.e., relations) after completing the merging of actors and intentional elements. This is because child elements connected through the links of the merged parent elements could or could not be merged, which affects the process of link merging. Furthermore, their approach [16] is semi-automatic and considers only the structural aspect of the GRL models, while *GRLMerger* is fully automated (an interactive option is also offered) and considers both the structure and the semantics of the GRL models.

Table 1 summarizes the comparison with related work. The comparison is based on the following criteria:

- **Operator:** denotes the operator used for the integration.
- **Input:** specifies the input to the used process.
- **Output:** specified the output generated from the operator.
- **Automation:** denotes whether the operator is fully automated, semi-automated, or manual.
- **Notation/language:** denotes the GORE model type.
- **Language specific:** specifies whether the approach can be applied to other types of goal models.
- **Validation:** presents the empirical method(s) conducted to validate the approach.
- **Syntactic/semantic:** denotes whether the approach is based on models syntactic similarities, semantic similarities, or both.

The main ascertainment is the existence of one single study that proposed the integration of GRL models [2]. The majority of the reviewed studies in Table 1 are manual [2, 3, 13, 39], while only one is semi-automated [16] and two are fully automated [26, 39]. Moreover, none of the reported studies have described all processes required to integrate two goal models starting from matching until the automatic generation of a complete integrated model that is free from errors. Although Hablutzet et al. [16] automated the merging process, the authors [16] did not consider the semantic matching between the intentional elements of Tropos goal models. The proposed *GRLMerger* approach covers all required operators: Match, Merge, Transform, and Refine.

It is worth noting that some of the proposed manual matching operators (i.e., map) are based on the goal model semantics [3, 24, 39]. However, these studies did not consider the syntactical differences between the matched constructs. Ignoring these syntactical differences might cause conflicts in the resulting integrated model.

Furthermore, Feng et al. [13] defined three specific rules for handling the syntactical differences based on the goal

model semantics. The proposed *GRLMerger* approach deals with the syntactical differences via two modes: fully automatically or semi-automatically, without restricting the user to specific constructs. Therefore, these two modes can be adapted to different languages (with minimal modifications).

4 GRLMerger: the approach

In this section, we describe our proposed *GRLMerger* approach to merge GRL models, whose overview is given as a workflow diagram in Fig. 3.

GRL models can be considered as a graph with rich semantics, where nodes and edges are of different types. Nodes represent GRL intentional elements, e.g., goals, softgoals, tasks, etc., and edges represent GRL links, e.g., contributions, decompositions, dependencies, etc. To tackle the complexity arising from the richness and heterogeneity of elements and links, the *GRLMerger* approach divides the problem into three separate steps, where each step deals with the processing of one type of constructs, e.g., actors, intentional elements, and links. Indeed, as shown in Fig. 3, the proposed *GRLMerger* approach starts first by integrating the actor's containers, followed by the integration of the intentional elements that are bound to the integrated actor's containers, and finally proceeds with the integration of links. Each of these steps involves two major operators, *match* and *merge*. The match operator employs semantic similarity to match actor containers and intentional elements. The merge operator decides about the final output result of the matched elements. During the merge process of two GRL models, conflicts may arise, e.g., different intentional element types, different link types, and different attributes (e.g., importance values, descriptions, etc.). To automate the resolution of such configuration conflicts, one of the two input GRL models is used as a *base* model. Therefore, a higher priority is given to the configuration of the selected base model, which is used as a reference model. The base model shall be specified by the user at the beginning and used throughout the whole merging process. Section 4.2 and 4.3 provides examples of resolving conflicts using the base model.

The proposed *GRLMerger* prototype tool (presented in Sect. 5) can be run in two modes: (1) fully automated mode and (2) interactive (semi-automated) mode. The fully automated mode uses a base model to resolve conflicts, while in the interactive approach the user has to select an option from a computed set of alternatives.

The following subsections describe the processes of the *GRLMerger* in detail. To demonstrate the proposed approach, we will use two GRL models *model_a* (Fig. 4a) and *model_b* (Fig. 4b) that are constructed from the online shopping GRL model of Fig. 1.

The proposed *GRLMerger* approach is structured using the pipes and filters architectural style and illustrated in Fig. 3. In what follows, we detail each of its processes.

4.1 Integration of GRL actor containers

The *GRLMerger* approach takes as input two syntactically correct GRL models, as no checks for correctness are performed. Before presenting the matching and merging procedures, we start by formalizing the notion of GRL actors.

Definition 1 (*GRL Actor*) A GRL actor is defined as a tuple $Actor = (ActorID, ActorName, ActorDescription, ActorImportance, ActorMetadata)$, where:

- *ActorID* is the Id of the actor.
- *ActorName* denotes the name of the actor (i.e., its enclosed text).
- *ActorDescription* denotes the description of the actor. This attribute is optional.
- *ActorImportance* denotes the importance of the actor that can be qualitative (i.e., low, medium, or high) or quantitative (value between 0 and 100). This default value is zero (i.e., not important).
- *IMetadata* denotes the metadata, i.e., attribute-value pairs, associated with the intentional element. This attribute is optional.

4.1.1 Matching actor containers

GRLMerger uses the names of actors (i.e., *ActorName* attribute) to compute the semantic similarity between actor containers. Given two input GRL models *Model_a* and *Model_b*, semantic similarity is calculated for each actor in *Model_a* with all actors of *Model_b*. If the semantic similarity between a pair of actors from *Model_a* and *Model_b* equals one, they are considered as matching actors. Otherwise, the user has to decide whether the rest of the actors from *Model_a* and *Model_b* are matching or not. Each actor is matched with one actor at most. The user input is essential in matching actors because incorrectly matching or mismatching actors would impact the merging results. In the example of Fig. 4, each model has one single actor. The computed similarity value between *Business Owner* and *Proprietor* is 0.7 (less than 1). Since matching actors is a crucial step toward obtaining a consolidated GRL model, the user is asked (in both modes automatic and interactive) to confirm that the actors are matching.

4.1.2 Merging actor containers

Matched actor containers are merged either automatically or interactively. A new *ActorID* is generated for the resulting

Table 1 Summary of goal-oriented models integration approaches

| References | Operator | Input | Output | Automation | Notation/ language | Lan- guage specific | Validation | Syntactic/ semantic |
|---------------------------------|--------------|-----------------------------------------------------------------|----------------------------------------------------------------------------|---------------------------|-----------------------|---------------------------|------------------------------|---------------------------|
| Sabetzadeh and Easterbrook [39] | Map | Two i* views | Interconnected sets | Manual | i* views | Yes | Example and Tool support | Semantic |
| | Merge | Interconnected sets | Merged i* view | Fully Auto- mated | | | | |
| Feng et al. [13] | Map | Two models | Overlap points Merge points Conflict points | Manual | Generic | No | Case study | Syntactic and Semantic |
| | Merge | Two models Overlap points Merge points Conflict points | Merged model | Manual | | | | |
| Beckers et al. [3] | Match | Goal-based models Problem-based models | Mapping between goal-based model elements and problem-based model elements | Manual | SI* Problem frames | Yes | Application scenario | Semantic |
| Li et al. [24] | Match | Goal model Non-functional patterns | Candidates of the non-functional patterns | Manual | Generic | No | Case study and Tool support | Semantic |
| | Merge | Goal model Candidates of the non-functional patterns | Merged goal model | Manual | | | | |
| Baslyman and Amyot [2] | Match | Goal models | Similarity model | Manual | GRL | Yes | Case study | Syntactic and Semantic |
| | Diff | Goal models | Dissimilarity model | Manual | | | | |
| | Merge | Similarity and dissimilarity models | Integrated model | Manual | | | | |
| Liu et al. [26] | Match | Goal models User reviews | Relationship between them | Fully Auto- mated | Generic | No | Experiment and Survey | Semantic |
| Hablutzet et al. [16] | Merge | Two goal models | Merged goal model | Semi Auto- mated | Tropos | Yes | Tool support and Experiment | Syntactic |
| GRLMerger | Match | Two goal models | Matched actors Matched elements Matched links | Fully Auto- mated | GRL | Yes | Tool support and Experiments | Syntactic and Semantic |
| | Merge | Matched actors Matched elements Matched links | Merged actors Merged elements Merged links | Fully / Semi Automated | | | | |
| | Transforming | Unmatched actors Unmatched elements Unmatched links | Moved to the merged model | Fully Auto- mated | | | | |
| | Refinement | Merged model | Refined merged model | Full / Semi Automated | | | | |

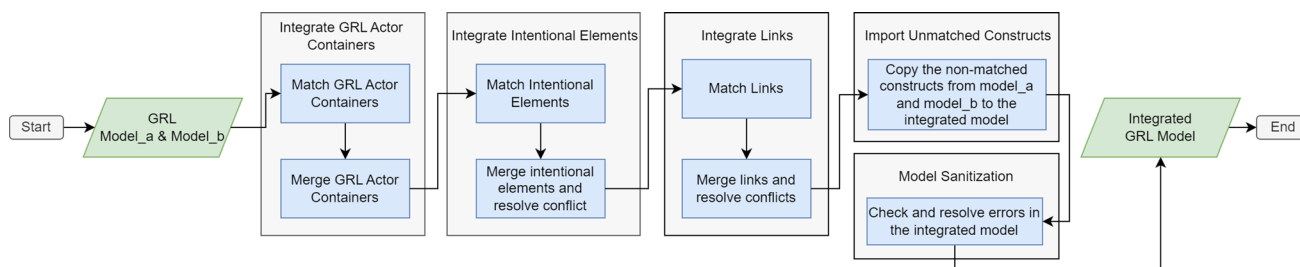


Fig. 3 *GRLMerger* approach

actor and is added to the integrated GRL model. However, the merge may lead to a conflict situation in case one or many of the actor attributes are different. We formalize this situation as follows:

Definition 2 (*Conflicting actor containers*) Let $A1$ and $A2$ be two actor containers: $A1 = (AID1, AName1, ADescription1, AImportance1, AMetadata1)$ and $A2 = (AID2, AName2, ADescription2, AImportance2, AMetadata2)$, such that $Similarity(AName1, AName2) = 1$, or user has confirmed that $AName1$ and $AName2$ are similar. A conflict occurs if at least one of the following conditions is satisfied:

- $ADescription1 \neq ADescription2$.
- $AImportance1 \neq AImportance2$.
- $AMetadata1 \neq AMetadata2$.

In case of automatic merging, the resulting actor container will have the attributes of the actor container that belongs to the selected base model. In the interactive mode, the user is prompted to select the attributes of the resulting actor container. In the example of Fig. 4, model_a (Fig. 4a) is used as a base model in the automatic mode. Hence, the resulting actor name will be *Business Owner* and all attributes (except the ActorID) of model_a will be copied to the resulting GRL model.

4.2 Integration of GRL intentional elements

Merged actor containers could have common (i.e., semantically similar) intentional elements that need to be matched and merged. GRL intentional elements can be defined as formalized as follows:

Definition 3 (*GRL Intentional Element*) A GRL intentional element is defined as a tuple:

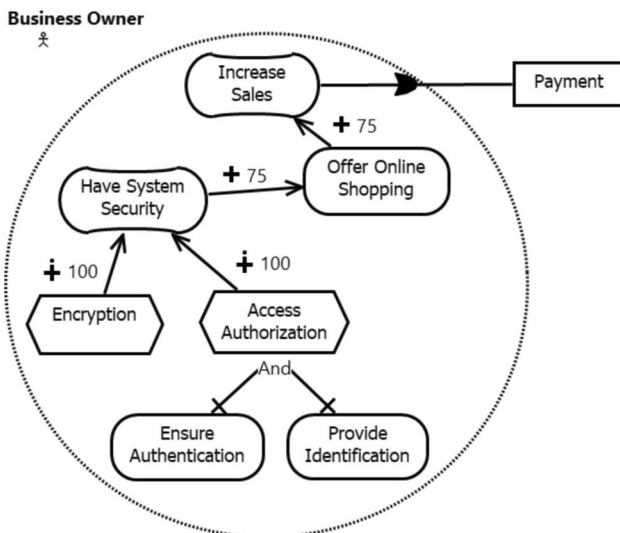
$IE = (IEID, IEActorID, IEType, IENAME, IEDescription, IEImportance, IEMetadata, IEdecompositionType)$, where:

- *IEID* is the Id of the intentional element.

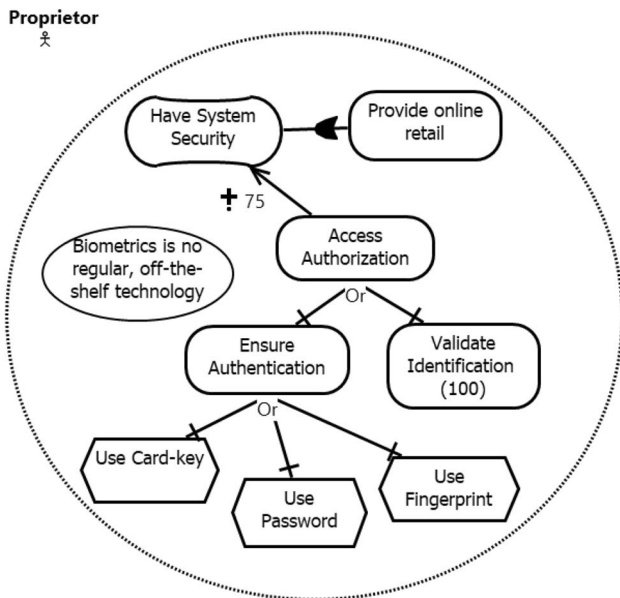
- *IEActorID* is the Id of the actor the intentional element is bound to. *IEActorID* may be empty (in case of an unbound intentional element).
- *IEType* denotes the type of the intentional element, e.g., Goal, SoftGoal, Task, etc.
- *IENAME* denotes the name of the intentional element (i.e., its enclosed text).
- *IEDescription* denotes the description of the intentional element. This attribute is optional.
- *IEImportance* denotes the importance of the intentional element that can be qualitative (i.e., low, medium, or high) or quantitative (value between 0 and 100). This default value is zero (i.e., not important).
- *IEMetadata* denotes the metadata, i.e., attribute-value pairs, associated with the actor. This attribute is optional.
- *IEdecompositionType* refers to the decomposition type, e.g., AND, OR, XOR. It may be empty (in case an intentional element is not decomposed into other intentional elements).

4.2.1 Matching GRL intentional elements

Similar to the actor containers matching, intentional elements are matched based on their names as well. Given two actor containers $A1$ and $A2$ that got matched as the result of the previous step (Sect. 4.1.1), *GRLMerger* calculates the semantic similarity between each intentional element of $A1$ and the intentional elements of $A2$. Thereafter, the semantic similarity values for all pairs are sorted in a descendant order and presented to the user. The user is then asked (in both modes automatic and interactive) to choose an appropriate threshold value, based on which *GRLMerger* decides whether two intentional elements are matching or not. It compares the semantic similarity value of each pair of intentional elements with the similarity threshold value. If the semantic similarity value is greater or equal to the similarity threshold value, the *GRLMerger* considers them as matched intentional elements. Each intentional element from $A1$ will be matched with one intentional element from $A2$ at most. It is worth noting that the selected similarity threshold value will be used throughout the whole merging process.



(a) Online Shopping GRL Model (Model.a)



(b) Online Shopping GRL Model (Model.b)

Fig. 4 GRL model_a and model_b

Definition 4 (Matched GRL intentional Elements) Let IE1 and IE2 be two GRL intentional elements: $IE1 = (IE1ID, IEActorID1, IEType1, IEName1, IEDescription1, IEImportance1, IEMetadata1, IEDecompositionType1)$ and $IE2 = (IEID2, IEActorID2, IEType2, IEName2, IEDescription2, IEImportance2, IEMetadata2, IEDecompositionType2)$.

IE1 matches IE2 (written as: $matches(IEID1, IEID2)$) iff $Similarity(IEName1, IEName2) \geq threshold$.

Table 2 shows some of the computed similarity values between intentional elements of model_a and model_b. With a threshold of 0.7, *Offer Online Shopping* is matched

with *Provide online retail*, and *Provide Identification* matches *Validate identification*.

4.2.2 Merging GRL intentional elements

In this step, the two matched intentional elements will be merged into one consolidated intentional element. This merged intentional element is then copied to the integrated GRL model and bound to the merged actor container.

Matched intentional elements are merged either automatically or interactively. A new ID is generated for the resulting intentional element. However, the merge may lead to a conflict situation in case one or many of the attributes of the matched elements are different. We formalize this situation as follows:

Definition 5 (Conflicting intentional elements) Let IE1 and IE2 be two intentional elements part of the same matched actor of two different GRL models:

$IE1 = (IEActorID1, IEType1, IEName1, IEDescription1, IEImportance1, IEMetadata1, IEDecompositionType1)$ and $IE2 = (IEActorID2, IEType2, IEName2, IEDescription2, IEImportance2, IEMetadata2, IEDecompositionType2)$, such that $matches(IEID1, IEID2)$. A conflict occurs if at least one of the following conditions is true:

- $IEType1 \neq IEType2$
- $IEDescription1 \neq IEDescription2$
- $IEImportance1 \neq IEImportance2$
- $IEMetadata1 \neq IEMetadata2$

Conflicts can be resolved either automatically (using the selected base model as a reference model) or interactively where the user selects the attributes of the resulting intentional element. In the example of Fig. 4, *Access Authorization* is described as a task (i.e., $IEType = Task$) in model_a (Fig. 4a), while it is described as a goal (i.e., $IEType = Goal$) in model_b (Fig. 4b). Since model_a was chosen as a base model, the resulting type of *Access Authorization* will be a task. The integrated model is shown in Fig. 1.

4.3 Integration of GRL links

In GRL, each intentional element could be linked to one intentional elements via one link at most. Each link has a parent intentional element and a child intentional element, as illustrated in Fig. 5. GRL intentional elements can be defined as formalized as follows:

Definition 6 (GRL link) A GRL link is defined as a tuple $Link = (LinkID, LinkType, ParentID, ChildID, LinkDecompositionType, LinkContributionValue)$, where:

Table 2 Computed similarity values between the intentional elements of the actors *Business Owner* and *Proprietor element*

| Business owner element | Proprietor element | Similarity |
|------------------------|-------------------------|------------|
| Have System Security | Have System Security | 1 |
| Ensure Authentication | Ensure Authentication | 1 |
| Access Authorization | Access Authorization | 1 |
| Offer Online Shopping | Provide online retail | 0.72 |
| Provide Identification | Validate identification | 0.7 |

- *LinkID* is the Id of the link.
- *LinkType* denotes the type of the link, e.g., Dependency, Decomposition, Contribution, etc.
- *ParentID* represents the ID of the parent element of the link.
- *ChildID* represents the ID of the child element of the link.
- *LinkDecompositionType* refers to the decomposition type, e.g., AND, OR, XOR. It may be empty (in the case of contributions and dependencies).
- *LinkContributionValue* denotes the contribution value, e.g., +75.

4.3.1 Matching GRL links

The matching of actors and intentional elements is made based on the semantic similarity of their names. In contrast, the links are matched based on their syntactical similarity, i.e., parent and child intentional elements. Each link is matched with one link at most. We define matched links as follows:

Definition 7 (*Matched GRL links*) Let $L1$ and $L2$ be two GRL links, defined as follows:

$$L1 = (LID1, LType1, LParentID1, LChildID1, LDecompositionType1, LContributionValue1),$$

$$L2 = (LID2, LType2, LParentID2, LChildID2, LDecompositionType2, LContributionValue2).$$

$L1$ matches $L2$ (written as $matches(LID1, LID2)$), iff

$matches(LChildID1, LChildID2)$ and $matches(LParentID1, LParentID2)$.

The link type is not considered as a condition to match links because GRL does not allow more than one link between two intentional elements, even when these links are of different types. Figure 6 illustrates the matched links between *model_a* and *model_b*, each color represents one matching. For instance, the contribution link (+100) between task *Access Authorization* and softgoal *Have System*

Security (in blue color) matches the contribution link (+75) between goal *Access Authorization* and softgoal *Have System Security* (in blue color). Both links share the same ParentID and ChildID.

4.3.2 Merging GRL links

Merging the matched links may lead to many conflicting situations:

1. **Different link types.** For example, in *model_a* (Fig. 6a), the link connecting softgoal *Have System Security* and goal *Offer Online Shopping* is a contribution, while in *model_b* (Fig. 6b) it is a dependency link.
2. **Different decomposition types.** For example, in *model_a* (Fig. 6a), task *Access Authorization* is decomposed, via a AND-decomposition, into goal *Ensure Authentication* and goal *Provide Identification* (orange and purple color), while in *model_b* (Fig. 6b) the goal *Access Authorization* is decomposed via an OR-decomposition into similar intentional elements.
3. **Different contribution values.** For example, in *model_a* (Fig. 6a), task *Access Authorization* contributes to softgoal *Have System Security* with +100 contribution value, while in *model_b* (Fig. 6b) it is +75.

These four situations are formalized as follows:

Definition 8 (*Conflicting links*) Let $L1$ and $L2$ be two GRL matching links:

$$L1 = (LID1, LType1, LParentID1, LChildID1, LDecompositionType1, LContributionValue1),$$

$$L2 = (LID2, LType2, LParentID2, LChildID2, LDecompositionType2, LContributionValue2).$$

A conflict occurs iff at least one of the following conditions is satisfied:

1. $LType1 \neq LType2$
2. $LDecompositionType1 \neq LDecompositionType2$
3. $LContributionValue1 \neq LContributionValue2$

Each pair of matched links is merged separately from the other matched links and copied to the integrated model. Conflicts can be resolved either automatically (using the selected base model as a reference model) or interactively where the user selects the attributes of the resulting link. Algorithm 1 depicts the automatic merging algorithm. The inputs to the algorithm are the matched links, i.e., *link_a* and *link_b*, and the selected base model, i.e., *model_a*.

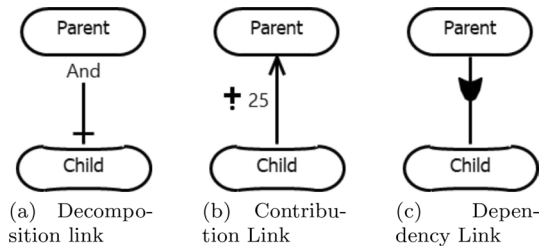


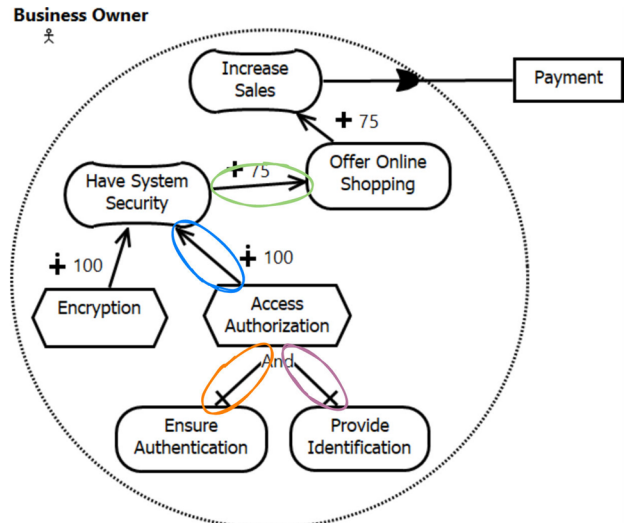
Fig. 5 Link parent/child

Algorithm 1 Automatic merging of links

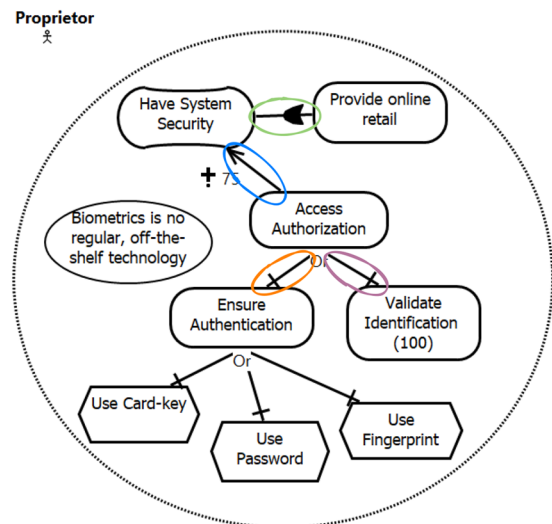
```

Input : model_a           ▷ base model
         link_a             ▷ link from model_a
         link_b             ▷ link from model_b
Output: link_ab          ▷ resulting merged link
link_ab.LinkType = link_a.LinkType;
if link_a.LinkType ≠ link_b.LinkType then
  reportConflict;
  if link_ab.LinkType = contribution then
    link_ab.LinkContributionValue :=
      link_a.LinkContributionValue;
    if link_a.LinkContributionValue ≠
      link_b.LinkContributionValue then
      | reportConflict;
    end
  else
    if link_ab.LinkType = decomposition then
      link_ab.LinkDecompositionType :=
        link_a.LinkDecompositionType;
      if link_a.LinkDecompositionType ≠
        link_b.LinkDecompositionType then
      | reportConflict;
      end
    end
  end
else
  if link_a.LinkContributionValue ≠
    link_b.LinkContributionValue then
  | reportConflict;
  end
  if link_a.LinkDecompositionType ≠
    link_b.LinkDecompositionType then
  | reportConflict;
  end
  link_ab.LinkContributionValue :=
    link_a.LinkContributionValue;
  link_ab.LinkDecompositionType :=
    link_a.LinkDecompositionType;
end

```



(a) Online Shopping (Model_a) showing the matched Links



(b) Online Shopping (Model_b) showing the matched links

Fig. 6 Matched links between Model_a and Model_b

If the matched links have different types, the algorithm reports the specific type of the conflict and resolves it by selecting the attributes of the link of the base model, i.e., model_a. If the matched links have the same type, the algorithm checks their attributes (*LinkContributionValue* for

contributions and *LinkDecompositionType* for decompositions) for conflicts. Conflicts are resolved by selecting the attributes of the link of the base model, i.e., model_a. In the case of a reciprocal parent–child situation, the resulting link (i.e., link_ab) will have the same attributes as link_a, of the base model, such that the parent and child ids as direct mappings of the child and parent ids of the link of the base model.

In the example of Fig. 4, the contribution (+75) between softgoal *Have System Security* and goal *Offer Online Shopping* in Fig. 6a is merged with the dependency between goal *Provide online retail* and softgoal *Have System Security* in Fig. 6b, resulting in an integrated contribution (+75) link (see the integrated model in Fig. 1).

4.4 Import unmatched GRL constructs

Given two models model_a and model_b to be merged, the unique constructs are the constructs from model_a that are not matching any constructs from model_b and vice versa. These constructs have to be copied to the integrated model to ensure its completeness. The unmatched actors, intentional elements, and links from model_a and model_b are imported (i.e., copied) in the integrated model.

Figure 7 highlights the constructs in model_a (Fig. 7a) and model_b (Fig. 7b) that will be copied to the integrated model. The syntactical structure of the copied constructs remains the same in the merged model. That is, the softgoal will remain a softgoal, and the contribution link will remain a contribution link, etc. Also, the intentional elements linked to an intentional element that got merged will be linked to the same merged intentional element in the merged model. For example, softgoal *Increase Sales* is copied along with its link to the merged goal *Offer Online Shopping* via the same contribution link and the same contribution value, i.e., +75.

4.5 Model sanitization

Each type of GRL construct is integrated separately from the other types. Furthermore, matched constructs are merged without checking the other constructs connected to it directly or indirectly. Furthermore, unmatched constructs are copied to the resulting model. Subsequently, the integrated GRL model may be subject to three syntactic errors: (1) parent intentional element with more than one decomposition type, (2) reciprocal parent–child relationships, and (3) cycles (although the initial input models are cycle-free). The aim of model sanitization phase is to resolve such errors. The following subsections will detail each error and how the *GRLMerger* approach deals with it.

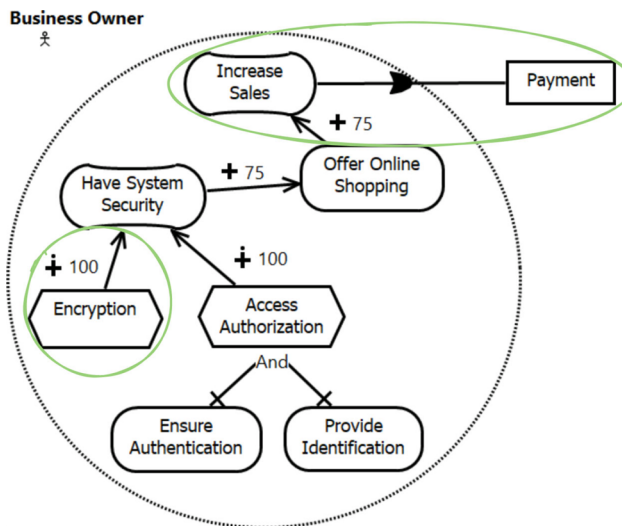
4.5.1 Sanitizing parent intentional element having multiple decomposition types

Each parent intentional element can have one decomposition type at most. However, if two parent intentional elements with different decomposition types were integrated, and they are decomposed by unique intentional elements (i.e., there is no matching between them), the integrated model will include a parent intentional element connecting its children via two different decomposition types. For instance, in Fig. 8, the goal *Ensure Authentication* with AND-decomposition (Fig. 8a) got integrated with the goal *Ensure Authentication* that has XOR-decomposition (Fig. 8b). In addition, their children's tasks *Use Fingerprint* and *Use Password* got integrated. Moreover, the links connecting the integrated goal *Ensure Authentication* and the integrated tasks *User Fingerprint* and *Use Password* were integrated. The conflict of different decomposition types was resolved during the merging process. However, during the import of task *Use Card-Key* and its XOR-decomposition link from model_f, the parent intentional element *Ensure Authentication* would have two different decomposition types as illustrated in the imaginary picture of Fig. 8c (*GRL* language and the *jUCM-Nav* tool do not allow such configuration).

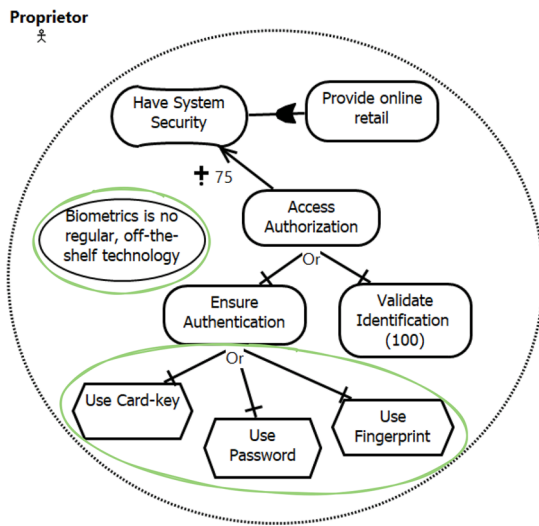
We propose two strategies to resolve this error:

1. **Strategy #1—Unification of the decomposition types:** It consists of changing one of the decomposition types to the other. Hence, all siblings will be connected via the same decomposition type. In the automatic merging mode, the non-base decomposition type is converted to the base decomposition type, if there is at least one merged intentional element among the siblings that are from the base model.

The existence of common siblings means that there is a common concept between them. Hence, the decomposition type can be changed. Assume that Model_e shown in Fig. 8a is the base model, while Model_f presented in Fig. 8b is the non-base model. *GRLMerger* would unify different decomposition types of the goal *Ensure Authentication* (Fig. 8c), to the AND-decomposition type (Fig. 8d). This is because the original siblings of the task *Use Card-Key* in Model_f (that has a different decomposition type) were integrated with the tasks in the base Model_e (i.e., tasks *Use Fingerprint* and *Use Password* in Model_f, Fig. 8b) that were integrated with tasks *Use Fingerprint* and *Use Password* in the base Model_e (Fig. 8a), respectively.
2. **Strategy #2—Add a temporary intentional element:** This strategy is selected if the parent elements do not share any matched children elements (i.e., Strategy #1



(a) Constructs to be copied from Model_a



(b) Constructs to be copied from Model_b

Fig. 7 Constructs to be imported in the merged model

is not applicable). Therefore, this strategy consists of adding a temporary intentional element (i.e., temporary parent) connected to the original parent intentional element, that has two decomposition types. The decomposition type from the base model will be retained for the parent with the multiple decomposition types. The temporary intentional element is connected to that parent via the retained decomposition type. The intentional elements of the different decomposition type (i.e., non-

base decomposition type) will no longer be connected to that parent directly (i.e., links shall be removed), yet they will be connected to the temporary intentional element via their non-base decomposition type. The added temporary element has the same type of siblings and it is connected to the parent intentional element via the same decomposition type of its siblings. Figure 9 depicts this configuration. Goal *Ensure Authentication* in Model_g (Fig. 9a) got integrated with the goal *Ensure Authentication* that is part of Model_h (Fig. 9b). Their children’s intentional elements with their links were copied to the integrated model (Fig. 9c) since there is no matching between them. Since there are no common siblings between the children of *Ensure Authentication* in Model_g and Model_h, and assuming that Model_g is the base model, the AND-decomposition will be retained for the parent intentional element (i.e., the goal *Ensure Authentication*) and the XOR-decomposition type from the non-base model will be used to connect the siblings of the Model_h. The user is then asked to name this temporary intentional element, i.e., *TEMP*.

4.5.2 Sanitizing the reciprocal parent–child relationships

Since each pair of matched links is merged separately from the other matched links, the resulting model may contain two intentional elements, where the parent of the first link is matching the child of the second link and vice versa (see Definition 9). Such configuration is not allowed in GRL and is not supported in jUCMNav.

Definition 9 (*Reciprocal parent–child relationship*) Let L1 and L2 be two GRL matching links:

$L1 = (LID1, LType1, LParentID1, LChildID1, LDecompositionType1, LContributionValue1),$

$L2 = (LID2, LType2, LParentID2, LChildID2, LDecompositionType2, LContributionValue2).$

A reciprocal parent–child relationship occurs iff: $matched(link_a.LChildID, link_b.LParentID)$ and $matched(link_b.LChildID, link_a.LParentID)$

Similar to the automatic processes of merging actors/intentional elements/links, the resolution of this error condition consists of the removal of the link originating from the non-base model. In the interactive mode, the user is asked to select the link to be dropped.

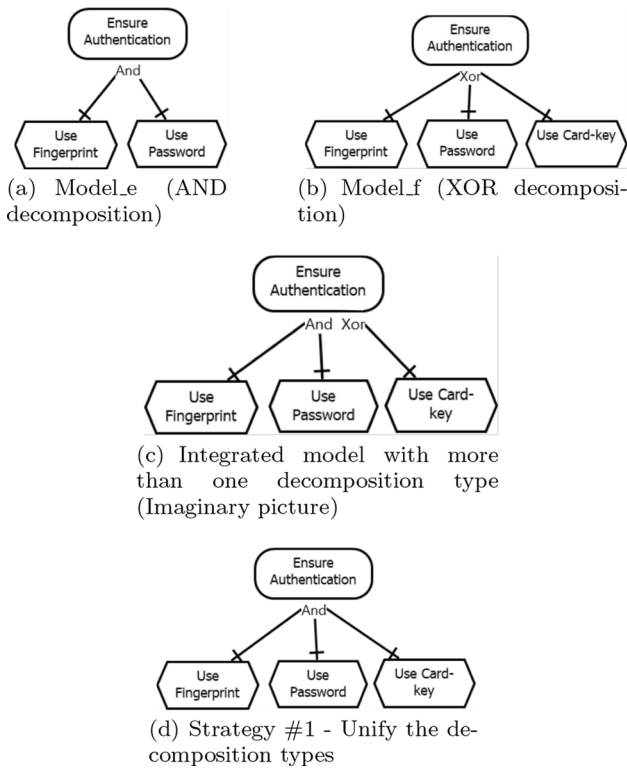


Fig. 8 Strategy #1: resolution of a parent intentional element with two decomposition types

4.5.3 Removing GRL cycles

Cycles are bad smells in GRL models that need to be identified and resolved [34]. When performing satisfaction analysis (one of the most important goal model analysis techniques [19]) on a GRL model, the presence of cycles would prevent computed satisfaction values from propagating through all GRL model elements; hence hindering the satisfaction analysis.

A cycle is defined as a set of consecutively linked intentional elements, where each one is a parent in one link and a child in another link. We call such an intentional element

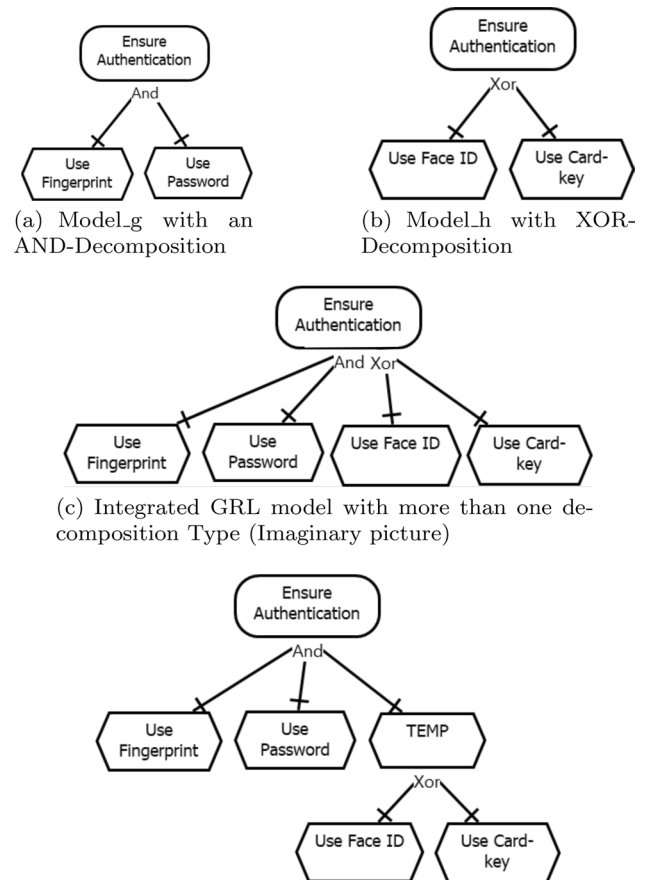


Fig. 9 Strategy #2: resolution of a parent intentional element with two decomposition types

an internal intentional element. Figure 10a and b illustrates two sub-models (Model_c and Model_d) resulting in a cycle once integrated (Fig. 10c). In Fig. 10c, the three links (link 1, link 2, and link 3) form a cycle between the internal intentional elements *Access Authorization*, *Strong Password*, and *Ensure Authentication*.

Algorithm 2 shows the cycle detection procedure. It uses two sub-procedures, *getAllParents* and *getAllChildren*, to get the list of all parents and children of a given intentional element, respectively. Assuming that *X* is an internal intentional element, we start by getting all parent intentional elements of the links connected to *X* until reaching the root, and all children intentional elements of the links connected to *X* until reaching a leaf. Thereafter, we check if there is an intentional element in the set of *X*'s parents that exists in the set of *X*'s children. This means that there is a cycle between *X* and the found intersection.

Algorithm 2 Identifying cycles

```

Input : internal_intentional_elements ▷ intentional
         elements that are parents in link and
         children in another link
         links_i ▷ links from the integrated model
Output: cycle_links ▷ links forming the cycle
for i in internal_intentional_elements do
  parents = getAllParents(i)
  children = getAllChildren(i)
  cycle_intentional_elements = parents ∩ children
  if cycle_intentional_elements not empty then
    for x ← count(cycle_intentional_elements)
      do
        for y ←
          count(cycle_intentional_elements)-1 do
          cycle_links ← links from links_i
          where (parent.links_i ==
            cycle_intentional_elements[x] and
            child.links_i ==
            cycle_intentional_elements[y+1]) or
            (child.links_i ==
            cycle_intentional_elements[x] and
            parent.links_i ==
            cycle_intentional_elements[y+1])
          end
        end
      return cycle_links
    end
  end
end

```

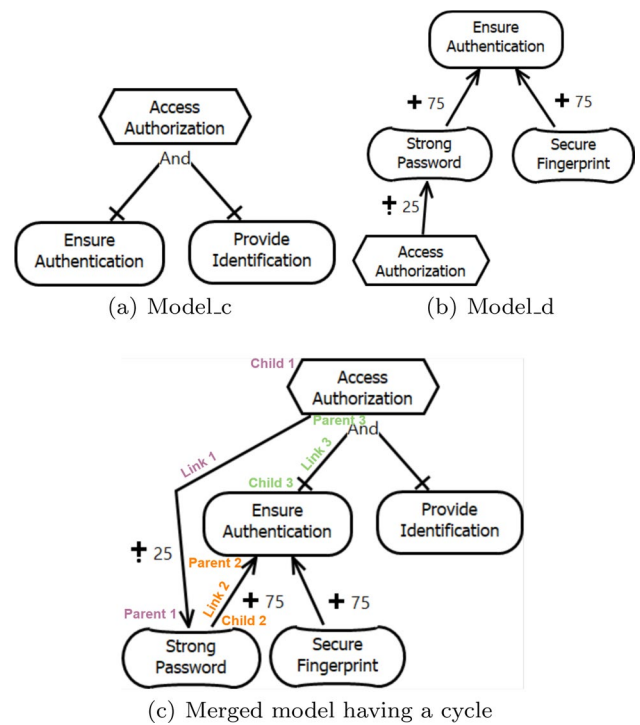


Fig. 10 GRL cycle

For example, in Fig. 10, the task *Access Authorization* is an internal intentional element that is a child in link 1 and a parent in link 3. The parents of the link that is connected to the internal task *Access Authorization* are *Strong Password* and *Ensure Authentication*, and the children are *Ensure Authentication*, *Provide Identification*, *Strong Password*, and *Secure Fingerprint*. The intentional elements *Ensure Authentication* and *Strong Password* exist in both parent's and children's sets. Therefore, there is a cycle between *Access Authorization*, *Ensure Authentication*, and *Strong Password*.

Algorithm 3 getAllParents

```

Input : internal_intentional_element ▷ an
         intentional element that is parent in a link
         and a child in another link
         links_i ▷ links from the integrated model
Output: parents ▷ intentional elements that are
         parents in the links connected to the
         internal_intentional_element
for link in links_i do
  if child.link == internal_intentional_element
  then
    parents.append(parent.link)
    getAllParents(parent.link)
  end
end
end
return parents

```

Algorithm 4 getAllChildren

```

Input : internal_intentional_element ▷ an
         intentional element that is parent in a link
         and a child in another link
         links_i ▷ links from the integrated model
Output: children ▷ intentional elements that are
         children in the links connected to the
         internal_intentional_element
for link in links_i do
  if parent.link == internal_intentional_element
  then
    children.append(child.link)
    getAllChildren(child.link)
  end
end
end
return children

```

The automatic breaking of a cycle consists of removing a non-base model link from the cycle. If the links composing the cycle are from the base model, or they are integrated links, the user is asked to choose a link to be removed. In the interactive mode, the user is asked to select the link to be dropped. Hence, he can select a link from the base model.

It is worth noting that depending on the targeted analysis technique, the presence of cycles would not cause an issue. To this end, in addition to the final cycle-free model, we keep a version of the integrated model that includes cycles. It is the responsibility of the analyst to use it in its current form or break the cycles.

5 GRLMerger prototype tool

The *GRLMerger* approach is implemented as a Python application. It is publicly available on PyPi.² The user can install and run the package as described in the README.md file. An overview of the *GRLMerger* prototype tool processes is

shown in Fig. 11. The following sections detail the implementation of each phase.

5.1 Input models

The *GRLMerger* tool works with the textual representation of GRL (TGRL) [1]. The tool takes two TGRL files (with the extension *.xgrl*) as input, e.g., *startGRLMerger('model_a.xgrl', 'model_b.xgrl')*) and converts them into dataframes [36]. Next, it will check whether each construct has a unique ID. Lastly, it will pre-process the names of actors and intentional elements for automatic semantic matching.

5.1.1 Converting TGRL to dataframes

Although the syntax of TGRL is well-structured, simple, and consistent, it is more convenient to convert the input TGRL models to dataframes [36] in order to facilitate its processing.

Definition 10 (*Dataframe data model*) A dataframe *D* is a tuple (*A*, *R*, *C*, *T*), where *A* is an $m \times n$ array of data entries that represents the dataframe content, *R* is an array of *m* row labels, *C* is an array of *n* column labels, and *T* is an array of types for each column.

The dataframe structure was selected because it is a logical data structure that organizes data in rows and columns [36], which facilitates accessing, retrieving, and modifying the data entries. Furthermore, dataframes can be embedded in Python, which offers many NLP libraries to support the automatic semantic matching of data entries. Before populating the dataframes, some pre-processing steps are conducted to clean the TGRL files, including the removal of spaces at the beginning and at the end of each line, the removal of empty lines, and ensuring that each construct definition or attribute is in one line.

For each TGRL model, we define three dataframes, one for actors, one for the intentional elements, and one for the links. The three dataframes have the same structure as the definitions 1, 3, and 6, respectively. After pre-processing the TGRL specification, each actor, intentional element, and link are parsed and inserted into the corresponding dataframe as a new row. However, some intentional elements (e.g., resources) usually do not belong to an actor, especially if they are part of a dependency link. The *GRLMerger* tool adds a dummy actor definition to the actors dataframe with the ID 'X#Y' and name 'X#YDUMMYACTOR' (see Table 3) and assigns to it the intentional elements that do not belong to any actor. TGRL syntax does not allow the use of the hash sign

² <https://pypi.org/project/GRLMerger/>.

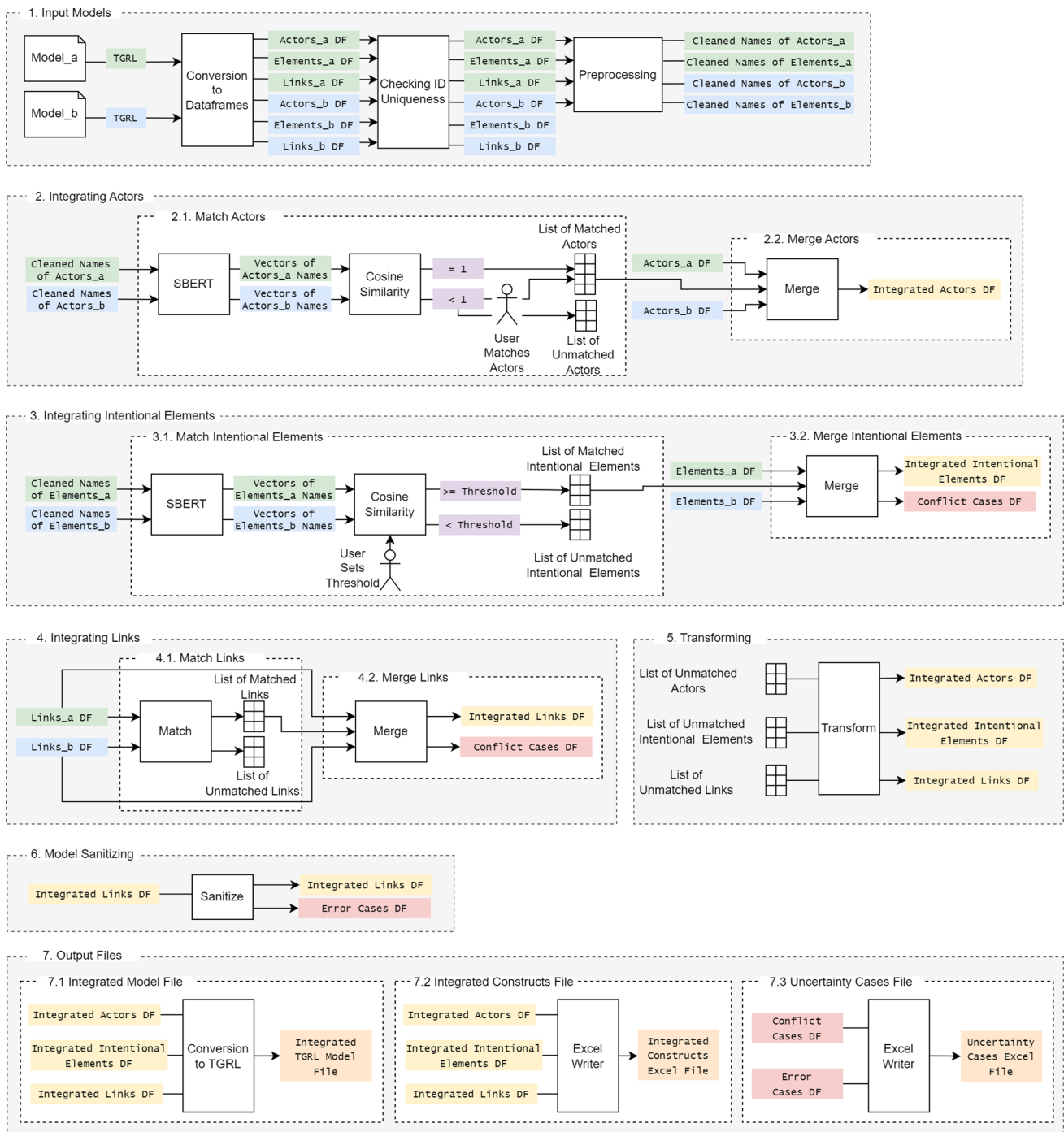


Fig. 11 GRLMerger tool

so *GRLMerger* uses it to differentiate this dummy actor and its intentional elements and links. It is worth noting that TGRRL does not specify an ID for links, however, the *GRLMerger* tool assigns an ID for each link to facilitate their processing (see Definition 6).

5.1.2 Checking the uniqueness of IDs

TGRRL uses IDs to differentiate between the model constructs. If two constructs have the same ID and they are linked to other constructs via links, the links will not show correctly when generating the corresponding graphical GRL model in the jUCMNav tool. Therefore, *GRLMerger*

Table 3 Actors dataframe for the online shopping Model_a

| ActorID | ActorName | ActorDescription | ActorImportance | ActorMetadata |
|----------------------|---------------------------------|------------------|-----------------|---------------|
| businessOwner X#Y | Business Owner X#YDUMMYACTOR | | | |

checks that each ID in both input models is unique. If an ID was used for two constructs, a random number between 0 and 99 is generated and attached to the ID of one of the similar constructs' IDs. This would allow the tool to distinguish them in case they were not merged.

5.1.3 Pre-processing

Actors and intentional element names, i.e., *ActorName* and *IEName*, are the attributes used for semantic matching. To achieve better results, names are pre-processed by lower casing them, removing the special characters, if any, expanding contractions (“Can’t” is changed to “Cannot”), and lemmatizing them (reducing the names to their base forms). The pre-processing is performed automatically. However, if the TGRL model contains context-dependent abbreviations, the user is asked to provide the full text. For example, the abbreviation “SMS” could refer to “Short Message Service,” while in the experimental model used in Sect. 6, it refers to “Seminars Managing System”. These pre-processing steps aim to convert the names of constructs to simple English sentences to be used for measuring the similarity values [18].

5.1.4 Name embedding

The proposed *GLRMerger* approach matches actors and intentional elements based on the semantic similarity between their names. Semantic-based similarity measures convert the input text into vectors (i.e., embedding) that capture their semantic information, where similar sentences are close in vector space [38]. There are several word embedding models trained on large corpora. However, word embedding models do not perform well in representing the meaning of a full sentence [8]. Since the names of GRL constructs could be one or two words long (as for actors), or a full sentence (as for intentional elements), the *GRLMerger* tool uses one of the latest state-of-the-art sentence embedding model to convert constructs' names into vectors. It uses the Sentence-BERT (SBERT) embedding model [38]. The SBERT model was selected because it outperformed the other state-of-the-art sentence

embedding models (InferSent [8] and Universal Sentence Encoder [6]) in the semantic textual similarity task [38]. The *GRLMerger* tool uses the cosine similarity (Eq. 1) to calculate the semantic similarity between sentences as suggested by Reimers and Gurevych [38].

5.2 Supporting dataframes

The *GRLMerger* tool uses two additional dataframes:

1. **Conflicts dataframe:** used to store the resolved conflicts. Table 4 presents the structure of the conflict cases dataframe. The column *m_ID* refers to the merged construct ID where this conflict occurred, while the *m_selected_value* is the selected value to resolve the conflict either automatically or interactively. The *conflict_type* could be one of the conflict cases, which are *Element Type*, *Link Type*, *Decomposition Type*, or *Contribution Value*. If the interactive mode is chosen, any decision taken by the user is stored in the conflict cases dataframe. Therefore, the *conflict_type* column could include *Actor Name*, *Actor Description*, *Element Name*, *Element Importance*, etc.
2. **Errors dataframe:** used to store the error cases. Table 4 presents the structure of the error cases dataframe. The *error_type* could be *Reciprocal parent-child relationship*, *Cycle*, or *Multiple Decomposition Types*. The *actor_name* refers to the actor where this issue occurred. Sometimes cycles are spread between two or more actors. All actors involved in the cycle are listed in the *actor_name* column. If the error was a cycle, the solution will be to drop a link, hence, *m_ID* refers to the dropped link ID. The description includes details about the dropped link. In case the error was multiple decomposition types, the solution could be either to unify the decomposition type or to add a temporary intentional element. The *m_ID* in this case reports the ID of the intentional element that got its decomposition type changed to the selected decomposition type, and the description depicts the conducted sanitization strategy. The description column includes free-text because it is easier to describe the solution instead of separating it into multiple structured columns.

5.3 Integration of actor containers

The actors dataframes of `model_a` and `model_b` are used by *GRLMerger* to match and merge the actors. *GRLMerger* calculates the semantic similarity between each actor from `model_a` and each actor from `model_b` using the cosine similarity measure (Eq. 1). Identical actors (i.e., cosine similarity = 1) got merged directly. If the cosine similarity was less than one, the *GRLMerger* tool asks the user to match the rest of the unmatched actors. For the online shopping example, the similarity between “Business Owner” and “Proprietor” is 0.7. In the automatic mode, the user is asked whether these two actors match or not (see Fig. 12a). The integrated actor would have the attributes of the actor from the selected base model. However, in the interactive mode, the user is asked to select the attributes’ values of the integrated actor if they are not identical. For example, Fig. 12b illustrates the question asked to the user to select the name of the merged actor. The *GRLMerger* tool creates a new ID for the integrated actors by combining the IDs of the matched actors using the plus sign (i.e., $actor_a_ID + actor_b_ID$). This format helps *GRLMerger* to refer to the original actors if needed. The integrated actors are stored in a new dataframe that has the same structure as the actors’ dataframe.

5.4 Integration of the intentional elements

The intentional elements integration process uses the intentional elements dataframes of the input models. *GRLMerger* uses the embedded names of the intentional elements (as described in Sect. 5.1.4) to calculate the semantic similarity between each intentional element from `model_a` that is part of the integrated actor_{ab} with all intentional elements from `model_b` that belong to the integrated actor_{ab}. The *GRLMerger* tool displays the similarity values sorted in descending order to the user (see Fig. 13), so the user can select the optimal similarity threshold value to ensure the correct matching of the intentional elements. The similarity threshold value must be a real value between 0 (not matching at all) and 1 (identical).

In the automatic mode and in case of a matching conflict, e.g., different intentional element types, the type from the base model is retained (along with all its attributes). The conflict and how it was resolved are stored in the conflicts’ dataframe. In the interactive mode, the user is asked to specify the integrated intentional element’s attributes if they were not identical (Fig. 14). All decisions made by the user are stored in the conflict dataframe.

The integrated intentional elements are stored in a new dataframe (having the same structure as the other intentional elements dataframes) created to store the integrated intentional elements. All attributes of the integrated intentional elements are either from the base model (automatic

Table 4 Supporting dataframes

| Conflicts dataframe | | Errors dataframe | |
|---------------------|---------------|------------------|-------------|
| m_ID | conflict_type | actor_name | error_type |
| | | model_a_name | actor_name |
| | | model_a_value | solution |
| | | model_b_name | description |
| | | model_b_value | |
| | | m_selected_value | |

```

Does the actor ( business owner ) match the actor ( proprietor )?
1- Yes
2- No

1 | 2: 

```

(a) Automatic mode: *GRLMerger* asking the user to match the actors

```

Select the name of the merged actor for the matched actors [ Proprietor ] with [ Business Owner ]
1- Proprietor
2- Business Owner
3- Proprietor/Business Owner

1 | 2 | 3 : 

```

(b) Interactive mode: *GRLMerger* asking the user to select the merged actor's Name**Fig. 12** *GRLMerger* tool: merging actors' containers

mode) or entered by the user (interactive mode), except the ID which is generated using the format *intentional_element_a_ID+intentional_element_b_ID*. Furthermore, the links dataframe of *model_a* that connects the matched *intentional_element_a* and the links dataframe of *model_b* that connects the matched *intentional_element_b* are updated to be connected to the integrated intentional element. That is, if the intentional elements that got integrated are the parent intentional element or the child intentional element in the links dataframes of *model_a* and *model_b*, the *GRLMerger* tool updates the IDs with the new integrated intentional element ID.

5.5 Integration of links

The *GRLMerger* tool uses the links dataframes of *model_a* and *model_b* to integrate their links. A new ID for each integrated link is generated by combining the two IDs of the matched links (i.e., the integrated link format is *link_a_ID+link_b_ID*). The new ID of the integrated links will be used to know whether the link that is part of a cycle is an integrated link or an unmatched imported link. The integrated links are inserted in a new dataframe (having the same structure as the input links dataframe) for the integrated links. All resolved link conflicts, either automatically or interactively, are stored in the conflict dataframe.

5.6 Importing unmatched elements

After completing the integration of all matched constructs (actors, intentional elements, and links), the *GRLMerger* tool copies the unmatched/unmerged constructs to the integrated

model dataframes. The IDs of the imported constructs from *model_a* and *model_b* have not been changed.

5.7 Model sanitization

The *GRLMerger* tool uses the integrated links dataframe (that includes the transformed links as well) to sanitize the resulting model. For example, Fig. 15 illustrates a scenario where the user is asked to resolve the multiple decomposition types issue.

5.8 Output files

The *GRLMerger* tool generates the following files:

1. **The integrated model file:** the generated integrated model is the combination of the three dataframes: (1) the integrated actors dataframes, (2) the intentional elements dataframes, and (3) the links dataframe. The *GRLMerger* tool converts these dataframes to TGRL syntax. The *GRLMerger* cleans the IDs of the integrated constructs by removing the added “plus sign,” since the TGRL syntax does not allow it. The *GRLMerger* tool saves the integrated model TGRL file with the extension *.xgrl*. The file name is *integratedModel_model_a_model_b.xgrl*, where *model_a* and *model_b* are the names of the input models.
2. **the integrated model before cycle removal:** *GRLMerger* produces a copy of the resulting *xgrl* file before the sanitization from cycles.
3. **the integrated constructs file:** To assist modelers when analyzing or reviewing the integration results, the *GRLMerger* tool stores the integrated constructs in an excel file enclosing three sheets (i.e., a sheet for the

| | Actor Name | OnlineShopping_model_a | OnlineShopping_model_b | Similarity Value |
|----|----------------|------------------------|-----------------------------------------------------|------------------|
| 0 | Business Owner | Have System Security | Have System Security | 1 |
| 1 | Business Owner | Ensure Authentication | Ensure Authentication | 1 |
| 2 | Business Owner | Access Authorization | Access Authorization | 1 |
| 3 | Business Owner | Offer Online Shopping | Provide online retail | 0.72 |
| 4 | Business Owner | Provide Identification | Validate identification | 0.7 |
| 5 | Business Owner | Access Authorization | Ensure Authentication | 0.57 |
| 6 | Business Owner | Ensure Authentication | Access Authorization | 0.57 |
| 7 | Business Owner | Ensure Authentication | Use Password | 0.51 |
| 8 | Business Owner | Access Authorization | Use Password | 0.5 |
| 9 | Business Owner | Have System Security | Ensure Authentication | 0.45 |
| 10 | Business Owner | Ensure Authentication | Have System Security | 0.45 |
| 11 | Business Owner | Provide Identification | Use Fingerprint | 0.45 |
| 12 | Business Owner | Ensure Authentication | Validate identification | 0.44 |
| 13 | Business Owner | Encryption | Have System Security | 0.44 |
| 14 | Business Owner | Access Authorization | Have System Security | 0.38 |
| 15 | Business Owner | Have System Security | Access Authorization | 0.38 |
| 16 | Business Owner | Have System Security | Use Password | 0.37 |
| 17 | Business Owner | Encryption | Use Password | 0.37 |
| 18 | Business Owner | Encryption | Ensure Authentication | 0.33 |
| 19 | Business Owner | Increase Sales | Provide online retail | 0.33 |
| 20 | Business Owner | Ensure Authentication | Use Fingerprint | 0.32 |
| 21 | Business Owner | Provide Identification | Ensure Authentication | 0.31 |
| 22 | Business Owner | Have System Security | Use Fingerprint | 0.29 |
| 23 | Business Owner | Provide Identification | Biometrics is no regular , off-the-shelf technology | 0.27 |
| 24 | Business Owner | Encryption | Use Fingerprint | 0.25 |

Specify the threshold value
0.7

Fig. 13 GRLMerger tool: online shopping example: computed similarity values

Fig. 14 GRLMerger tool: user is asked to select the merged intentional element's type

```
Conflict in the type of the matched elements: [ Access Authorization ] with [ Access Authorization ]
1- The element Access Authorization has the type goal
2- The element Access Authorization has the type task
Select the type of the matched elements: [ Access Authorization ] with [ Access Authorization ]
1- goal
2- task
1 | 2 : 
```

```
The intentional element ( Access Authorization ) has children with multiple decomposition types
1- The elements ['Ensure Authentication'] linked via a decomposition of type or
2- The elements ['Provide Identification'] linked via a decomposition of type and
Would you like to unify the decomposition type or add a temporary element?
1- Unify the decomposition types
2- Add temporary element
1 | 2 : 
```

Fig. 15 GRLMerger tool-Interactive mode: ask the user to select s resolution strategy for the multiple decomposition types issue

integrated actors (Table 5), a sheet for the integrated intentional elements (Table 2), and a sheet for the integrated links (Table 6)). The GRLMerger tool uses the integrated actors, the integrated intentional elements, and the integrated links dataframes (without including the imported constructs). The file name is *integrated-Constructs_model_a_model_b.xlsx*. The difference between this file and the integrated model file is that this file only includes the integrated constructs in a table

format, while the integrated model file is the complete integrated TGRL model.

- File containing the resolved conflict and error cases:** The GRLMerger tool generates a file that reports on all resolved conflict and error cases, automatically or inter-

Table 5 Integrated actors sheet

| actor_1 | actor_2 | similarity |
|----------------|------------|------------|
| Business Owner | Proprietor | 0.7 |

Table 6 Integrated links sheet

| parent_element_1 | link_1 | child_element_1 | parent_element_2 | link_2 | child_element_2 |
|-----------------------|---------------|------------------------|-----------------------|---------------|------------------------|
| Offer Online Shopping | contribution | Have System Security | Offer Online Shopping | dependency | Have System Security |
| Have System Security | contribution | Access Authorization | Have System Security | contribution | Access Authorization |
| Access Authorization | decomposition | Ensure Authentication | Access Authorization | decomposition | Ensure Authentication |
| Access Authorization | decomposition | Provide Identification | Access Authorization | decomposition | Provide Identification |

actively, during the integration. The *GRLMerger* tool uses the conflict cases and the error cases dataframes to generate the file. The file name is *conflict_error_cases_model_a_model_b.xlsx*. It includes two sheets, one for the conflict cases (Table 7) and the other for the error cases.

6 Evaluation of the GRLMerger approach and prototype tool

In this section, we evaluate empirically our proposed *GRLMerger* approach and its prototype tool. We follow the template and recommendations presented by Wohlin et al. [44].

Figure 16 provides an overview of the experimental plan. We have used two TGRL models as our main subjects (see Sect. 6.1) in order to generate 24 TGRL sub-models (see Appendices 8 and 8).³ Then, we have designed three experiments (see Sect. 6.2) to evaluate the basic merging cases, conflict cases and semantic merging cases. The effectiveness of the *GRLMerger* approach and tool was measured in terms of correctness, completeness, and freeness from errors. The steps of the experimental plan are explained in the following sections.

6.1 Subjects

In addition to the running example, we have applied our *GRLMerger* approach and tool to two TGRL models:

- Example 1: University Alumni (Fig. 17):** A modified version of a GRL model, introduced by Hassine and Amyot [17], describing how to foster a university–alumni relationship. The model has 4 actors: University, Alumni department, Alumnus, and Professor. In order to cover all types of GRL constructs, we have extended the model by adding a softgoal, a belief, a resource, an indicator, and an OR-decomposition link. In the rest of the paper, we refer to this model as MF.
- Example 2: Seminars Managing System (Fig. 18):** It models the objectives of a future seminar management

system along with its stakeholders goals. The model has 5 actors: seminars managing system, admin, speaker, organizer, and attendee. In the rest of the paper, we refer to this model as MS.

6.2 Experiment procedure

We have designed and conducted three experiments:

6.2.1 Experiment 1: basic merging

The University–Alumni (MF) and the Seminars Managing System (MS) TGRL models were used as ground truth to evaluate the *GRLMerger* approach.

From the university–alumni (MF) model, we derive:

- Case #1: Two sub-models *MF1* (Fig. 21) and *MF2* (Fig. 22), as follows:
 - $MF1 = MF \setminus X1$, where $X1 \subset MF$
 - $MF2 = MF \setminus X2$, where $X2 \subset MF$
 - $X1 \cap X2 = \phi$
 - $MF1 \cup MF2 = MF$
- Case #2: Two sub-models *MF3* (Fig. 23) and *MF4* (Fig. 24), as follows:
 - $MF3 = MF \setminus X3$, where $X3 \subset MF$
 - $MF4 = MF \setminus X4$, where $X4 \subset MF$
 - $X3 \cap X4 = \phi$
 - $MF3 \cup MF4 = MF$

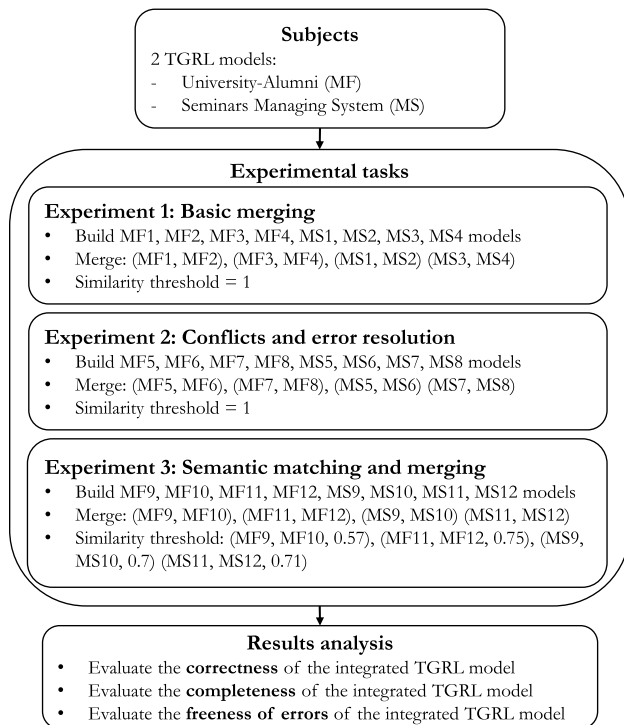
Similarly, from the Seminars Managing System (MS) model, we derive:

- Case #3: Two sub-models *MS1* (Fig. 33) and *MS2* (Fig. 34), as follows:
 - $MS1 = MS \setminus Y1$, where $Y1 \subset MS$
 - $MS2 = MS \setminus Y2$, where $Y2 \subset MS$
 - $Y1 \cap Y2 = \phi$
 - $MS1 \cup MS2 = MS$

³ The data used in the evaluation are openly available in github at <https://github.com/ndn94/GRLMerger-Paper>.

Table 7 Conflict cases sheet

| m_ID | conflict_type | actor_name | base_name | base_value | new_name | new_value | m_selected_value |
|------------------------------------------|--------------------|---------------------------------------|---------------------------------------------------------------|--------------|---------------------------------------------------------------|------------|------------------|
| accessAuthorizationaccessAuthorization50 | Element Type | Business Owner | Access Authorization | Task | Access Authorization | Goal | Task |
| R183R078 | Link Type | Business Owner - Business Owner | (Have System Security) linked to (Offer Online Shopping) | Contribution | (Have System Security) linked to (Offer Online Shopping) | Dependency | Contribution |
| R283R178 | Contribution Value | Business Owner - Business Owner | (Access Authorization) contributes to (Have System Security) | 100 | (Access Authorization) contributes to (Have System Security) | 75 | 100 |
| R483R278 | Decomposition Type | Business Owner - Business Owner | (Access Authorization) decomposed by (Ensure Authentication) | and | (Access Authorization) decomposed by (Ensure Authentication) | or | and |
| R583R378 | Decomposition Type | Business Owner - Business Owner | (Access Authorization) decomposed by (Provide Identification) | and | (Access Authorization) decomposed by (Provide Identification) | or | and |

**Fig. 16** Experimental design

- Case #4: Two sub-models $MS3$ (Fig. 35) and $MS4$ (Fig. 36), as follows:

- $MS3 = MS \setminus Y3$, where $Y3 \subset MS$
- $MS4 = MS \setminus Y4$, where $Y4 \subset MS$
- $Y3 \cap Y4 = \phi$
- $MS3 \cup MS4 = MS$

All pairs of sub-models (e.g., $MF1$ and $MF2$) have identically matching constructs syntactically and lexically. In addition, in the automatic merging mode, sub-models $MF1$, $MF3$, $MS1$, and $MS3$ were set as the base models. Similarity threshold value was set to 1 since there are no lexical differences between the names of the matched intentional elements. Table 8 presents the number and types of constructs in each input model.

6.3 Experiment 2: conflict and error cases

This aim of Experiment 2 is to evaluate the proposed *GRLMerger* approach and tool in terms of resolving the conflicts automatically and generating a merged model that is error-free. To this end, we have built pairs of sub-models, as in Experiment 1, but we have introduced

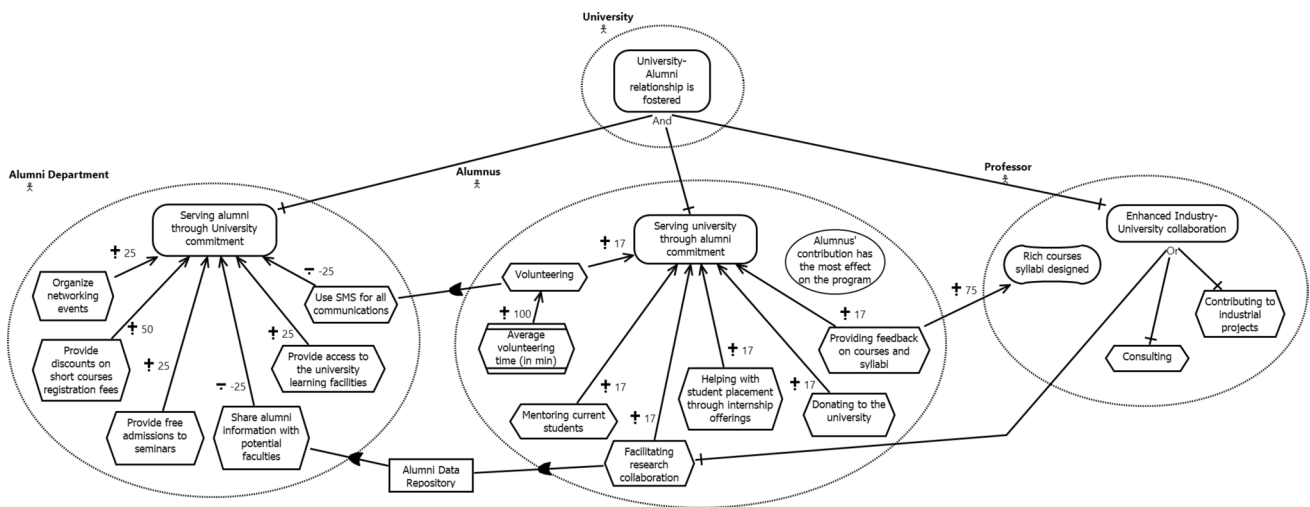


Fig. 17 Fostering university–alumni relationship GRL model

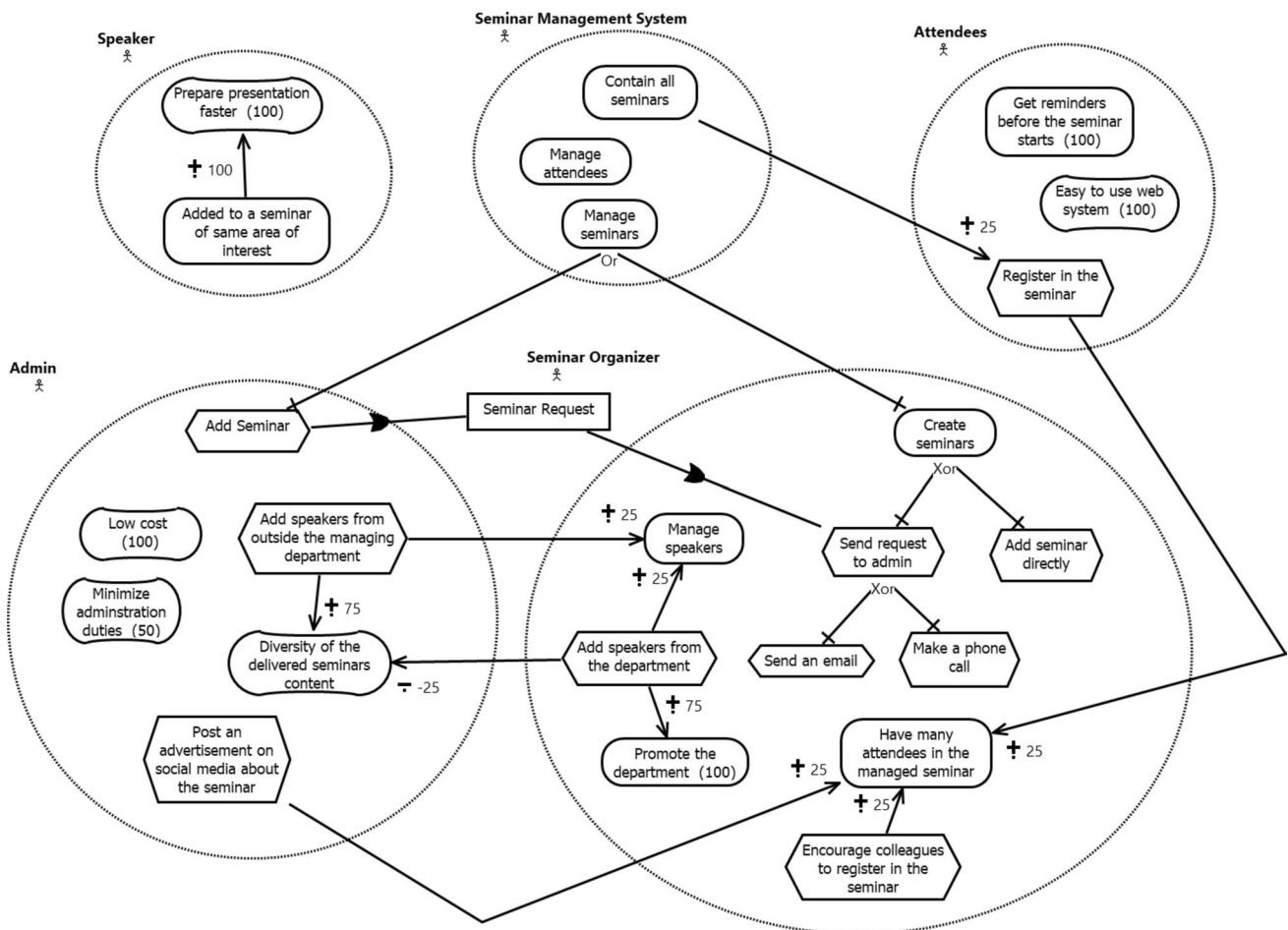


Fig. 18 Seminars managing system GRL model

Table 8 Experiments input models

| Case # | | Actor | Goal | Softgoal | Task | Belief | Resource | Indicator | Decomposition link | Contribution link | Dependency link | Total |
|---------------------|------|-------|------|----------|------|--------|----------|-----------|--------------------|-------------------|-----------------|-------|
| <i>Experiment 1</i> | | | | | | | | | | | | |
| Case 1 | MF1 | 3 | 3 | 0 | 8 | 1 | 1 | 0 | 5 | 6 | 1 | 28 |
| | MF2 | 4 | 2 | 1 | 11 | 0 | 1 | 1 | 1 | 8 | 2 | 31 |
| Case 2 | MF3 | 3 | 3 | 0 | 12 | 1 | 0 | 1 | 3 | 11 | 1 | 35 |
| | MF4 | 4 | 4 | 1 | 9 | 0 | 1 | 0 | 5 | 8 | 2 | 34 |
| Case 3 | MS1 | 3 | 6 | 3 | 7 | 0 | 1 | 0 | 4 | 5 | 2 | 31 |
| | MS2 | 5 | 7 | 5 | 9 | 0 | 0 | 0 | 4 | 7 | 0 | 37 |
| Case 4 | MS3 | 4 | 4 | 5 | 9 | 0 | 1 | 0 | 2 | 3 | 2 | 30 |
| | MS4 | 5 | 5 | 3 | 7 | 0 | 1 | 0 | 4 | 6 | 2 | 33 |
| <i>Experiment 2</i> | | | | | | | | | | | | |
| Case 1 | MF5 | 4 | 4 | 1 | 11 | 0 | 1 | 1 | 5 | 12 | 2 | 41 |
| | MF6 | 4 | 3 | 2 | 12 | 1 | 1 | 1 | 5 | 12 | 2 | 43 |
| Case 2 | MF7 | 3 | 3 | 1 | 13 | 1 | 0 | 1 | 3 | 12 | 1 | 38 |
| | MF8 | 4 | 4 | 2 | 9 | 0 | 1 | 0 | 5 | 8 | 3 | 36 |
| Case 3 | MS5 | 4 | 7 | 2 | 7 | 0 | 1 | 0 | 4 | 6 | 2 | 33 |
| | MS6 | 5 | 8 | 5 | 9 | 0 | 0 | 0 | 4 | 9 | 0 | 40 |
| Case 4 | MS7 | 5 | 6 | 3 | 9 | 0 | 1 | 0 | 5 | 7 | 2 | 38 |
| | MS8 | 5 | 9 | 5 | 10 | 0 | 1 | 0 | 6 | 9 | 3 | 48 |
| <i>Experiment 3</i> | | | | | | | | | | | | |
| Case 1 | MF9 | 4 | 4 | 0 | 13 | 1 | 0 | 1 | 6 | 11 | 1 | 41 |
| | MF10 | 4 | 4 | 2 | 12 | 0 | 1 | 0 | 4 | 13 | 3 | 43 |
| Case 2 | MF11 | 4 | 4 | 1 | 12 | 0 | 0 | 1 | 6 | 12 | 1 | 41 |
| | MF12 | 4 | 3 | 2 | 13 | 1 | 1 | 1 | 3 | 18 | 3 | 49 |
| Case 3 | MS9 | 5 | 8 | 5 | 9 | 0 | 1 | 0 | 6 | 7 | 2 | 43 |
| | MS10 | 5 | 8 | 5 | 11 | 0 | 0 | 0 | 6 | 9 | 0 | 44 |
| Case 4 | MS11 | 5 | 9 | 3 | 8 | 0 | 1 | 0 | 4 | 8 | 2 | 40 |
| | MS12 | 5 | 8 | 5 | 12 | 0 | 1 | 0 | 7 | 9 | 1 | 48 |

syntactic differences in the non-base models in order to generate conflict cases and errors in the integrated model. All pairs of sub-models (e.g., MF5 and MF6) have lexically identical elements (semantically matching). Hence, the similarity threshold value was set also to 1.

Examples of syntactic changes to the non-base sub-models include:

- Change the intentional element type, e.g., from goal to softgoal.
- Change the link type, from contribution to a dependency.
- Change the contribution value, e.g., from +100 to +50.
- Change the decomposition type, e.g., from OR to XOR.
- Switch the direction of a link to create an error state or a cycle.

From the university–alumni (MF) model, we derive:

- Case #1: Two sub-models MF5 (Fig. 25) and MF6 (Fig. 26), as follows:

- MF5 = MF \ Z1, where Z1 \subset MF
- MF6 is built from MF by removing some constructs and making syntactic changes to some others.
- MF5 \cup MF6 \neq MF, as MF6 contains syntactic changes.

- Case #2: Two sub-models MF7 (Fig. 27) and MF8 (Fig. 28), as follows:

- MF7 = MF \ Z2, where Z2 \subset MF
- MF8 is built from MF by removing some constructs and making syntactic changes to some others.
- MF7 \cup MF8 \neq MF, as MF8 contains syntactic changes.

Similarly, from the Seminars Managing System (MS) model, we derive:

- Case #3: Two sub-models $MS5$ (Fig. 37) and $MS6$ (Fig. 38), as follows:
 - $MS5 = MS \setminus Z3$, where $Z3 \subset MS$
 - $MS6$ is built from MS by removing some constructs and making syntactic changes to some others.
 - $MS5 \cup MS6 \neq MS$, as $MS6$ contains syntactic changes.
- Case #4: Two sub-models $MS7$ (Fig. 39) and $MS8$ (Fig. 40), as follows:
 - $MS7 = MS \setminus Z4$, where $Z4 \subset MS$
 - $MS8$ is built from MS by removing some constructs and making syntactic changes to some others.
 - $MS7 \cup MS8 \neq MS$, as $MS8$ contains syntactic changes.

The union of sub-models pairs $MF5$ and $MF6$, $MF7$ and $MF8$, $MS5$ and $MS6$, and $MS7$ with $MS8$ will not result in the regeneration of the original models MF or MS . Table 8 presents the number of constructs in each input model of experiment 2.

6.4 Experiment 3: semantic matching

The aim of Experiment 3 is to evaluate the ability of *GRLMerger* to automatically match the constructs based on their semantics (using sentence embedding and cosine semantic similarity measures) and merge the matched ones.

From the university–alumni (MF) model, we derive:

- Case #1: Two sub-models $MF9$ (Fig. 29) and $MF10$ (Fig. 30), as follows:
 - $MF9$ is built from MF by removing some constructs and making semantic changes to some of the remaining ones.
 - $MF10$ (non-base model) is built from MF by removing some constructs and making syntactic and semantic changes to some of the remaining ones.
- Case #2: Two sub-models $MF11$ (Fig. 31) and $MF12$ (Fig. 32), as follows:
 - $MF11$ is built from MF by removing some constructs and making semantic changes to some of the remaining ones.
 - $MF12$ (non-base model) is built from MF by removing some constructs and making syntactic and semantic changes to some of the remaining ones.

Similarly, from the Seminars Managing System (MS) model, we derive:

- Case #3: Two sub-models $MS9$ (Fig. 41) and $MS10$ (Fig. 42), as follows:
 - $MS9$ is built from MS by removing some constructs and making semantic changes to some of the remaining ones.
 - $MS10$ (non-base model) is built from MS by removing some constructs and making syntactic and semantic changes to some of the remaining ones.
- Case #4: Two sub-models $MS11$ (Fig. 43) and $MS12$ (Fig. 44), as follows:
 - $MS11$ is built from MS by removing some constructs and making semantic changes to some of the remaining ones.
 - $MS12$ (non-base model) is built from MS by removing some constructs and making syntactic and semantic changes to some of the remaining ones.

The QuillBot tool⁴ was used to paraphrase and generate semantically similar text to the original model constructs' names. The union of sub-models pairs $MF9$ and $MF10$, $MF11$ and $MF12$, $MS9$ and $MS10$, and $MS11$ with $MS12$ will not result in the regeneration of the original models MF or MS . Table 8 presents the number of constructs in each input model of Experiment3.

The threshold value for each case was set based on the given input and the generated semantic similarity values: (1) **Case 1**: it was set to 0.57, (2) **Case 2**: it was set to 0.75, (3) **Case 3**: it was set to 0.7, and (4) **Case 4**: it was set to 0.71.

6.5 Effectiveness measurement

The effectiveness of the *GRLMerger* approach is evaluated in terms of: (1) correctness, (2) completeness, and (3) freedom from errors.

6.5.1 Correctness

In our context, correctness can be defined as follows:

Definition 11 (Correctness) The *GRLMerger* approach is able to match and merge the similar GRL constructs of two GRL models and produce one conflict-free integrated model.

In order to measure the correctness, we define:

⁴ <https://quillbot.com/>.

- **True positives (TP)**: the number of correctly matched and merged constructs by *GRLMerger*.
- **True negatives (TN)**: Number of constructs that were correctly not matched/merged by *GRLMerger*.
- **False positives (FP)**: Number of constructs that were not matched/merged by *GRLMerger*, while they should.
- **False negatives (FN)**: Number of constructs that were incorrectly matched and merged by *GRLMerger*.

Figure 19 shows the confusion matrix that was used to evaluate the correctness of *GRLMerger*. Correctness can be measured using accuracy (Eq. 2), precision (Eq. 3), and recall (Eq. 4):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4)$$

6.5.2 Completeness

In our context, completeness can be defined as follows:

Definition 12 (Completeness) The *GRLMerger* approach is able to produce an integrated model that includes all the matched/merged and unmatched/unmerged constructs from the input models (no construct from the input models is missing).

The completeness can be measured by a simple count of all non-matched constructs from the input models that are not present in the integrated model.

6.5.3 Freeness from errors

Definition 13 (Freeness from errors) *GRLMerger* is able to produce an integrated model that is free from cycles, reciprocal parent–child, and intentional elements with more than one decomposition type.

The freeness from errors of the integrated model generated by *GRLMerger* can be measured by a simple count of the number of syntactical errors (i.e., cycles, reciprocal parent–child, intentional element with more than one decomposition type) that are present in the integrated model.

| | | Actual Matching and Merging | |
|-----------|----------------------|---------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| | | Matching and Merging | Not Matching |
| GRLMerger | Matching and Merging | True Positive (TP) : Number of correctly matched and merged constructs by <i>GRLMerger</i> | False Negative (FN) : Number of constructs that were incorrectly matched and merged by <i>GRLMerger</i> |
| | Not Matching | False Positive (FP) : Number of constructs that were not matched/merged by <i>GRLMerger</i> , while they should | True Negative (TN) : Number of constructs that were correctly not matched/merged by <i>GRLMerger</i> |

Fig. 19 Correctness confusion matrix

6.6 Experiment results

This section presents the results of the conducted experiments.

6.6.1 Results of experiment 1

The *GRLMerger* tool was able to match and merge all the actually matching constructs as depicted in Table 9. Therefore, the correctness accuracy, precision, and recall were equal to one. Furthermore, the *GRLMerger* tool copied all non-matched constructs from the input models to the integrated model, which resulted in regenerating the original models, MF and MS. The generated integrated models were free from errors.

6.6.2 Results of experiment 2

Experiment 2 focused more on evaluating the performance of the *GRLMerger* approach and tool in the presence of conflicts and error cases. The results of Experiment 2 in Table 9 show that all the actually matching constructs were matched and merged. Therefore, the accuracy, precision, and recall of correctness were equal to one. Moreover, all non-matched constructs were copied to the integrated model. Therefore, the original models, MF and MS, were reconstructed with no errors. In addition, all conflicts were resolved as the base constructs' types were retained in the integrated model.

Table 9 Evaluation results: correctness confusion matrices

| Case # | GRLMerger | Actual Matching and Merging | |
|---------------------|----------------------|-----------------------------|--------------|
| | | Matching and Merging | Not Matching |
| <i>Experiment 1</i> | | | |
| Case 1 MF1–MF2 | Matching and merging | 10 | 0 |
| | Not Matching | 0 | 39 |
| Case 2 MF3–MF4 | Matching and merging | 20 | 0 |
| | Not Matching | 0 | 29 |
| Case 3 MS1–MS2 | Matching and merging | 20 | 0 |
| | Not Matching | 0 | 28 |
| Case 4 MS3–MS4 | Matching and merging | 18 | 0 |
| | Not Matching | 0 | 30 |
| <i>Experiment 2</i> | | | |
| Case 1 MF5–MF6 | Matching and merging | 33 | 0 |
| | Not Matching | 0 | 16 |
| Case 2 MF7–MF8 | Matching and merging | 23 | 0 |
| | Not Matching | 0 | 26 |
| Case 3 MS5–MS6 | Matching and merging | 25 | 0 |
| | Not Matching | 0 | 23 |
| Case 4 MS7–MS8 | Matching and merging | 32 | 0 |
| | Not Matching | 0 | 16 |
| <i>Experiment 3</i> | | | |
| Case 1 MF9–MF10 | Matching and merging | 27 | 0 |
| | Not Matching | 7 | 15 |
| Case 2 MF11–MF12 | Matching and merging | 40 | 0 |
| | Not Matching | 0 | 9 |
| Case 3 MS9–MS10 | Matching and merging | 30 | 0 |
| | Not Matching | 7 | 11 |
| Case 4 MS11–MS12 | Matching and merging | 31 | 0 |
| | Not Matching | 3 | 14 |

6.6.3 Results of experiment 3

The correctness results are presented in Table 9. Out of the four cases implemented in Experiment 3, the *GRLMerger* matched and merged the actually matching constructs in case 2 only. In the other cases, the *GRLMerger* mismatched some of the actually matching intentional elements which further resulted in mismatching the connecting links. The accuracy and recall of cases 1, 3, and 4 were less than one

Table 10 Experiment 3: correctness accuracy, precision, and recall

| Case # | Accuracy | Precision | Recall |
|-------------------|----------|-----------|--------|
| Case 1: MF9–MF10 | 0.86 | 1 | 0.79 |
| Case 2: MF11–MF12 | 1 | 1 | 1 |
| Case 3: MS9–MS10 | 0.85 | 1 | 0.81 |
| Case 4: MS11–MS12 | 0.94 | 1 | 0.91 |

as shown in Table 10. The correctness precision was equal to one because all matched and merged constructs are actual matching constructs. Although *GRLMerger* failed to match some of the constructs, these constructs and non-matched constructs were copied to the integrated model. Moreover, the integrated models were free from errors.

6.7 Results interpretation

The matching of actors and intentional elements is based on their names (i.e., semantic matching), while links are matched syntactically (i.e., based on the connected parent and child intentional elements). In experiments 1 and 2, the differences between the input models were purely syntactical. For example, some of the intentional elements exist in one input model and they are not in the other input model, or different types of the matched intentional elements (e.g., a goal is matched with a task). However, the matched actors and intentional elements had identical names (i.e., no synonyms or paraphrasing were applied). Therefore, the *GRLMerger* tool was able to match all actually matching constructs (using a similarity threshold of 1). Furthermore, the non-matched constructs were copied to the integrated model resulting in a model that is complete and free from errors.

GRLMerger decides if two intentional elements are matching based on the specified similarity threshold value. This threshold is set manually by the user as an input to the approach. Therefore, the results of the matching are impacted by the specified threshold. If a high threshold value is specified, the *GRLMerger* will not match all matching constructs (lower accuracy and recall), and hence, the generated merged model would include more constructs. However, based on our experiment 3, we couldn't confirm this correlation. Indeed, we notice that an increase in the threshold value (0.57 in case #1 to 0.75 in case #2) would increase the accuracy (from 0.86 in case #1 to 1 in case #2) and the recall (from 0.79 in case #1 to 1 in case #2). However, an increase in the threshold value (0.57 in case #1 to 0.7 in case #3) would decrease the accuracy (from 0.86 in case #1 to 0.85 in case #2).

In the input models of the third experiment, synonyms and paraphrases were used and results showed that the *GRLMerger* tool was not able to match all intentional elements (i.e., synonyms generated by Quillbot) that were supposed to be matched. This is because the names of these intentional

Table 11 Special experiments: correctness

| Case # | GRLMerger | Actual matching and merging | |
|--------------------------|----------------------|-----------------------------|--------------|
| | | Matching and merging | Not matching |
| Similarity threshold = 1 | Matching and merging | 15 | 0 |
| | Not matching | 19 | 15 |
| Similarity threshold = 0 | Matching and merging | 34 | 2 |
| | Not matching | 0 | 13 |

elements have semantic similarity values less than the specified similarity threshold value. Therefore, the *GRLMerger* did not recognize them as matching intentional elements.

Since the *GRLMerger* match/merge process starts with actor containers followed by intentional elements, and finally processes the links, a mismatch of the intentional elements would propagate to the links. Indeed, not all the actually matching links were matched and merged since their parent/child intentional elements are different. Moreover, this mismatch resulted in having some semantically similar intentional elements within the same actor. However, changing the threshold value would lead to a different resulting model and different accuracy values.

6.8 Impact of threshold selection

To provide more insight into the importance of the selection of an appropriate threshold value, two more experiments were conducted. These experiments were conducted to evaluate the performance of *GRLMerger* with the highest and the lowest possible similarity values which are 1 and 0, respectively. In these two extra experiments, we have used the same input models *MF9* and *MF10* (syntactically and lexically different but semantically similar) that were used in case 1 of the third experiment (Sect. 6.4). The first experiment was conducted with a threshold value equals to one, while the second experiment with a threshold value equals to zero"

- *Threshold = 1*: A threshold of 1 forces *GRLMerger* to do a lexical matching (matching only identical terms). *GRLMerger* was not able to match the intentional elements that have semantically similar but lexically different names. Table 11 shows 19 mismatched constructs. That is, it mismatched eight intentional elements which resulted in mismatching the links that are linking the integrated intentional elements. Furthermore, *GRLMerger* copied all the non-matched and mismatched constructs to the integrated model. Accuracy, precision, and recall are presented in Table 12.
- *Threshold = 0*: *GRLMerger* would match each intentional element from model_a with at most one intentional element from model_b. Therefore, all the matched

and unmatched intentional elements will be integrated. Table 11 shows the presence of incorrect matching between two constructs, the task “*Provide discounts on short courses registration fees*” was integrated with the task “*Organize networking events,*” hence the links connecting them were incorrectly integrated.

Based on the results of these two extreme cases and the results of Experiment 3, no conclusions can be drawn on the optimal ranges of the thresholds.

7 Discussion

In this section, we present the limitations of the proposed *GRLMerger* approach, then we compare it with existing GRL integration techniques, and finally we present its main threats to validity.

7.1 Limitations of the *GRLMerger* approach

Despite the perceived benefits of the proposed *GRLMerger* approach and tool, some limitations could impact the quality of the integrated GRL model:

- *Selection of the base model*: Conflicts that may arise during the integration of two GRL models are resolved automatically by giving a higher priority to the selected base model, i.e., used as a reference model. However, the selection of the other model as a base model would result in a different integrated model. Such discrepancies between the results of the merging process can be avoided by using the *GRLMerger* interactive mode.
- *Systematic merging*: *GRLMerger* merges the input models systematically without considering the context. For example, if the two input models come from different

Table 12 Special experiments results

| Case # | Accuracy | Precision | Recall |
|--------------------------|----------|-----------|--------|
| Similarity threshold = 1 | 0.61 | 1 | 0.44 |
| Similarity threshold = 0 | 0.96 | 0.94 | 1 |

contexts and domains, *GRLMerger* proceeds blindly with the automatic merging and does not detect that the models are from different domains. This issue should be detected by the analyst.

- *No discovery of new dependencies*: In case the input models have no common actors, the resulting model will be the combination of all actors within the input models. However, actors from the first input model won't be connected to actors of the second model, as no new dependencies are created. It is the role of the analyst to add such dependencies.
- *No redistribution of contribution weights*: The merging of contribution links may lead to a situation where the aggregation of the contribution weights exceeds 100. For instance, Fig. 20 illustrates the resulting model when integrating two models: Model_a (composed of goal G and tasks A and B) and Model_b (composed of goal G and tasks C, D, and E). The aggregation of contribution weights is 200. For an optimal model, we need to re-balance the contribution weights (to have a total of 100). This task is left to the analyst.
- *No backtracking*: In the interactive mode, the user is presented with a set of options that he has to choose from. Once the choice is made, the next step is processed and there is no possibility to go back and change previous choices.
- *Textual interaction*: In the interactive mode, the user is given a range of options (in the form of text) to choose from (e.g., dialog boxes in Figs. 12, 14). A graphical representation of the choices would be more effective.

7.2 Practical considerations

In this section, we provide some practical insights to help practitioners apply and adjust the proposed approach to their needs. From a practical perspective, *GRLMerger* may be used in the following situations:

1. **Early stages of the RE process**: Various techniques, such as interviews, surveys, and workshops, can be employed to collect information from stakeholders. The aim is to capture a comprehensive understanding of the stakeholders' needs, concerns, and objectives. The gathered information, collected by different teams, is then used to construct partial GRL models. A partial GRL model reflects the viewpoints of the team that developed it. The *GRLMerger* tool would assist in integrating these partial GRL models (belonging to the same context). While the fully automated mode of the tool may be used, the interactive mode is more suitable at this stage. As this marks the initial development of a comprehensive GRL model, the interactive merging mode affords users greater control over integration decisions, enabling the identification and resolution of potential conflicts. Deploying *GRLMerger* in this phase eliminates subjectivity among

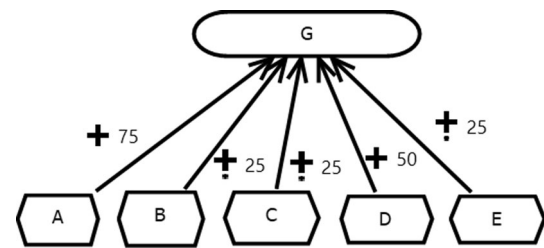


Fig. 20 Merge requiring weight redistribution

different teams when deciding on the semantic similarity of integrated goal model elements. It is worth noting that practitioners may also experiment with different semantic similarity thresholds and different base models.

2. **Requirements evolution**: In this scenario, we assume the existence of a well-established GRL model that captures the current goals and requirements agreed upon by stakeholders, serving as the baseline. Any introduction of new goals or requirements, prompted by factors such as emerging technology, evolving business landscapes, or the inclusion of novel quality aspects, necessitates adjustments to the baseline GRL model. To accommodate these changes, a partial GRL model is developed to specifically address the new technological or business perspectives. Utilizing *GRLMerger* in this scenario involves integrating the newly developed model with the baseline, treating the new model as the base during conflict resolution. Given the shared requirements context of both models, the automated mode of *GRLMerger* proves more efficient, reducing effort and time compared to manual or interactive merging. Practitioners may also experiment with different semantic similarity thresholds in this scenario. If the resulting consolidated model is unsatisfactory, they have the option to switch to the interactive mode for further refinement.

It is important to highlight that partial GRL models are not simply discarded post-integration; rather, they are retained to document the merging rationale.

7.3 Threats to validity

The proposed *GRLMerger* approach and the experimental validation are subject to several threats to validity that are categorized according to three important types identified by Wright et al. [45].

- **Construct Validity**: There is a potential threat concerning the need to specify a base model to be able to resolve the emerged conflicts automatically. In addition, choosing a different base model would lead to a different integrated model. To mitigate this risk, we have provided an

interactive mode, where the user decides how conflicts are resolved.

Another possible risk is related to the selection of a similarity threshold, based on which intentional elements are matched and merged. Having different threshold values would lead to different integrated models. To mitigate this risk, the *GRLMerger* tool computes the semantic similarity between all pairs of elements and presents them to the user. This would help the user to select an appropriate similarity threshold.

There is a potential criticism concerning the lack of validation of the integrated models by experts. To mitigate this risk, we have used three metrics (correctness, completeness, and freedom from errors) as performance indicators to evaluate the proposed *GRLMerger* approach. However, validating the results by experts could increase its validity.

Furthermore, a potential concern that may affect the performance of the matching process in the *GRLMerger* tool is the domain of the input models. To mitigate this threat, SBERT is chosen, having been trained on the Stanford Natural Language Inference (SNLI) corpus, a diverse collection of human-authored sentences. Consequently, the *GRLMerger* tool is not constrained by a specific domain.

- **Internal Validity:** The first threat is related to the performance of the used sentence embedding model, which may impact the effectiveness of *GRLMerger* matching process and hence affect the overall results. This threat is mitigated by selecting SBERT sentence embedding model, which outperforms other state-of-the-art embedding models proposed in the literature.

A second possible risk is related to the size of the used models in the experiments. Indeed, although the size of the used examples is comparable to those published in GORE papers, they are considered of medium size. The use of bigger models would increase the validity of the proposed *GRLMerger* approach and tool.

A third potential risk is that the *GRLMerger* approach may introduce cycles, reciprocal child-parent, and intentional elements with multiple decomposition types in the integrated model. To mitigate this risk, we have introduced the model sanitization phase to resolve and document such issues. A related risk is that the sanitization step involves the removal of some links to break a cycle or to resolve the reciprocal child-parent issue (having the base model as a reference). To mitigate this risk, the user may run choose the interactive mode and choose which link to break.

- **External Validity:** As for external validity, a possible threat is that our proposed *GRLMerger* approach is tailored to the GRL language [19] and more specifically the TGRL notation [1]. However, the *GRLMerger* tool

can be easily adjusted to cover the textual URN syntax proposed by Kumar and Mussbacher [22]. In addition, the four main steps of the integration process (match, merge, transform, sanitize) can likely be adjusted and applied to other goal-oriented languages, like iStar, that support actors, intentional elements, and their relationships. However, it is imperative to conduct a proof of concept and additional experiments to assess the effectiveness of *GRLMerger* in GORE languages that share similar constructs.

Another possible threat is related to the use of two models (University–Alumni and Seminars Management System) to derive the 24 models used in the 12 experiments. Having additional models from other domains would support the generalization of our results.

8 Conclusions and future work

The manual integration of GRL models requires intensive human effort and time. It is an error-prone process that could be biased by the conflicts between stakeholders' intentions, the usage of different vocabularies and the subjectivity of the requirements engineer/analyst performing the integration. In this paper, we have proposed the *GRLMerger* approach to integrate automatically (and interactively) two TGRL models into one consolidated model that is correct, complete, and free from any conflict that may arise during the merging process. *GRLMerger* considers both syntactic and semantic aspects (based on semantic similarity) when merging sub-models. *GRLMerger* has four main sequential steps: matching, merging, transforming, and sanitizing. A prototype tool was developed to implement the *GRLMerger* approach. In addition to the integrated model, the tool documents all matched and merged constructs, the resolved conflicts and error cases, and provides a copy of the model before cycle resolution. The *GRLMerger* approach and tool have been validated using three experiments showing very promising performance.

As future work, we plan to extend the proposed *GRLMerger* approach to be able to integrate more than two GRL models at a time. Furthermore, in order to increase its adoption, we plan to integrate the developed *GRLMerger* tool within *jUCMNav* [20].

Appendix A: fostering university–alumni relationships GRL sub-models

See Figs. 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 and 32.

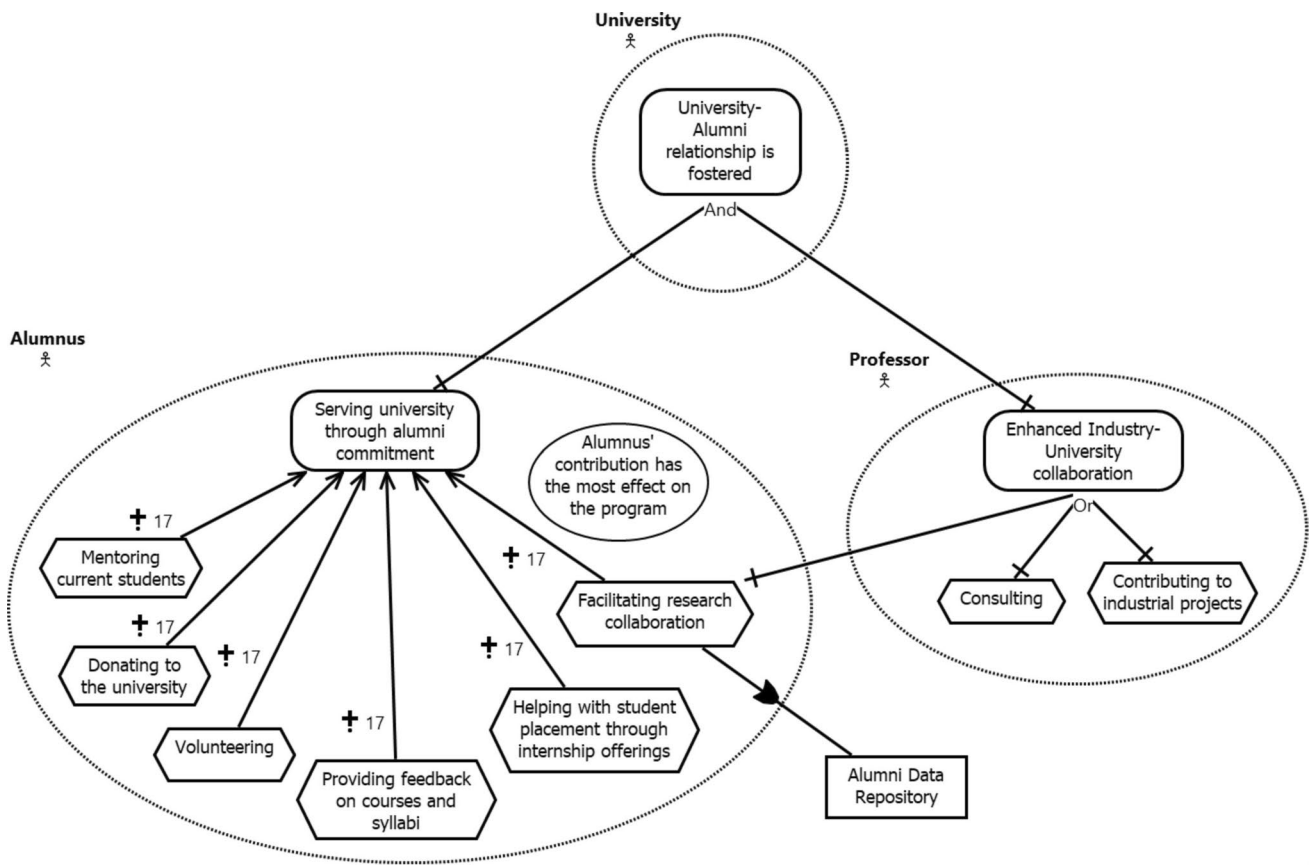


Fig. 21 Input model MF1

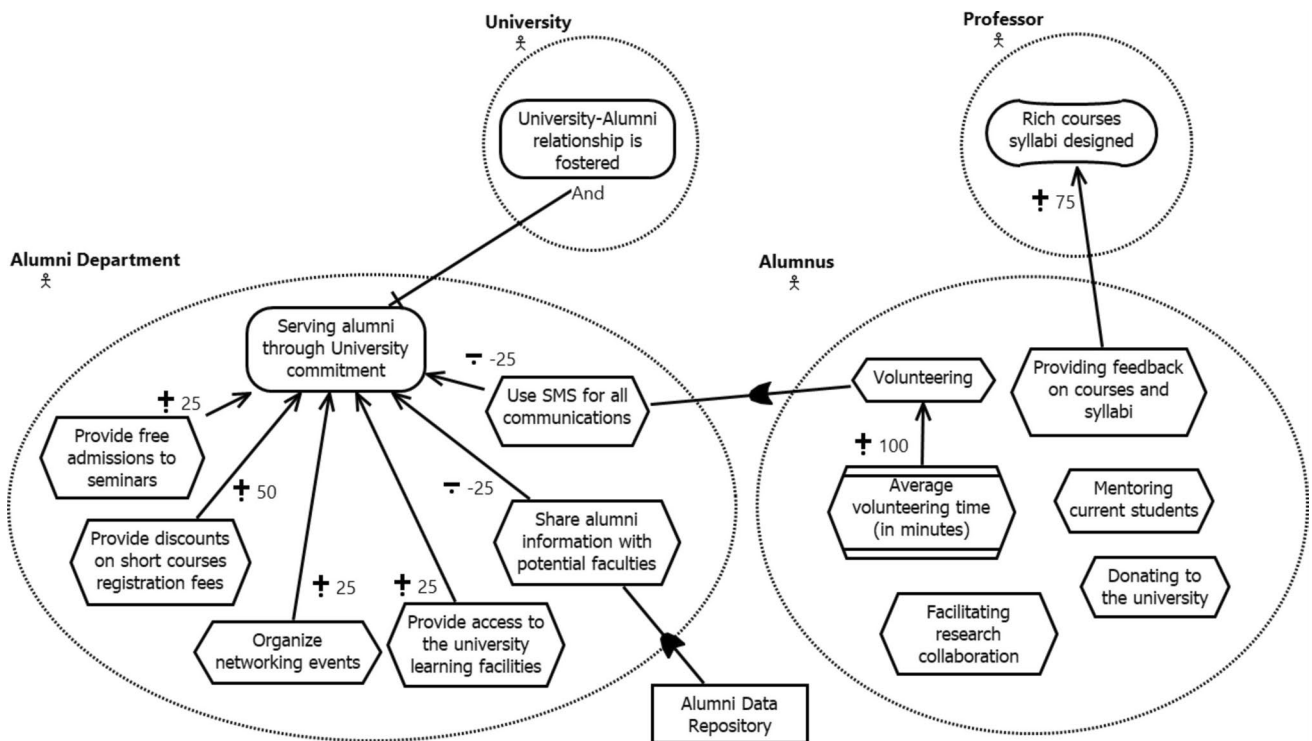


Fig. 22 Input model MF2

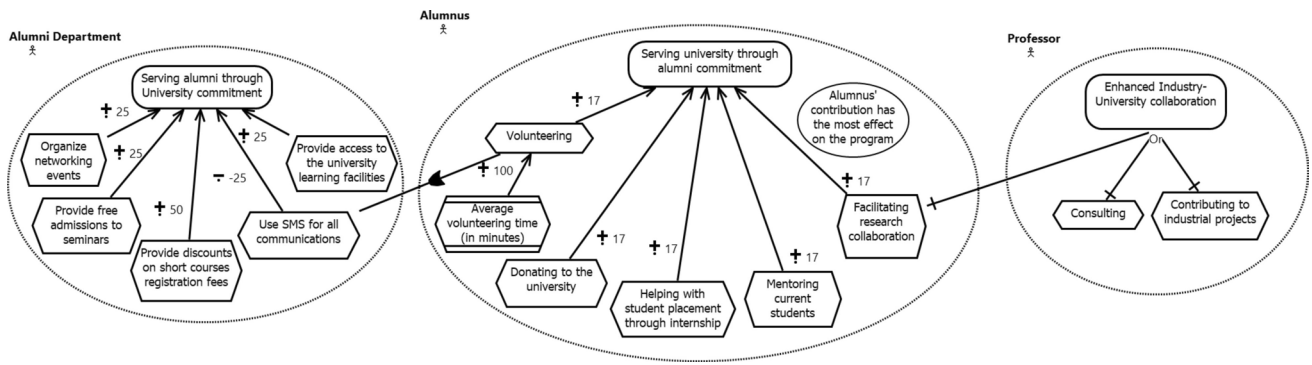


Fig. 23 Input model MF3

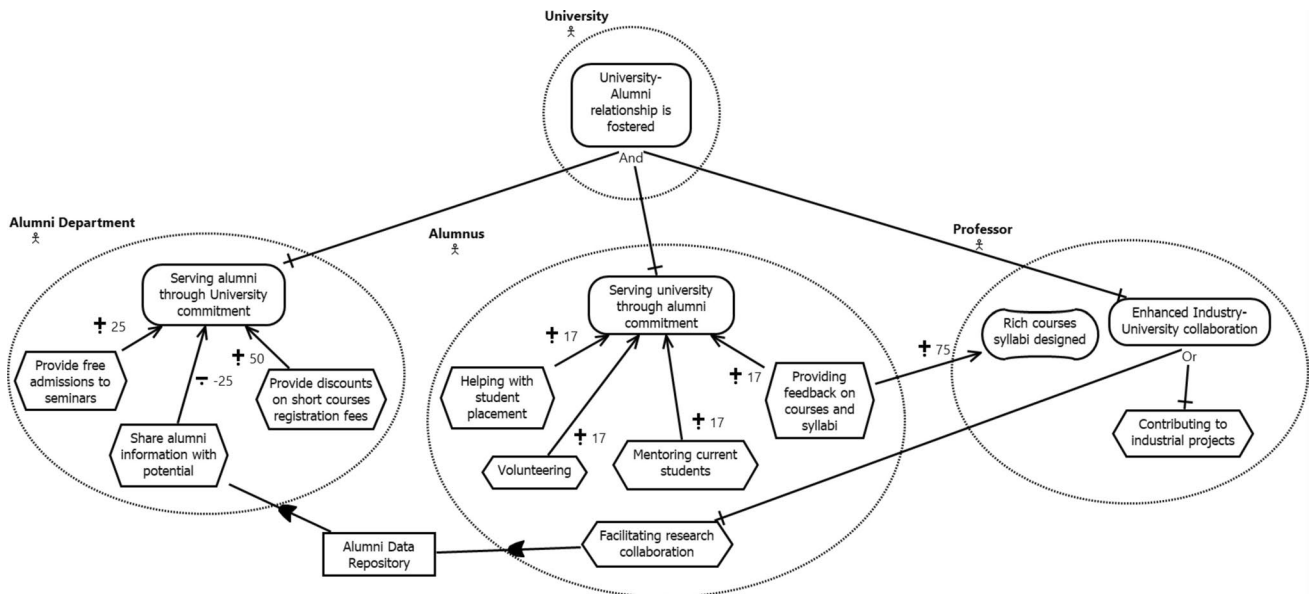


Fig. 24 Input model MF4

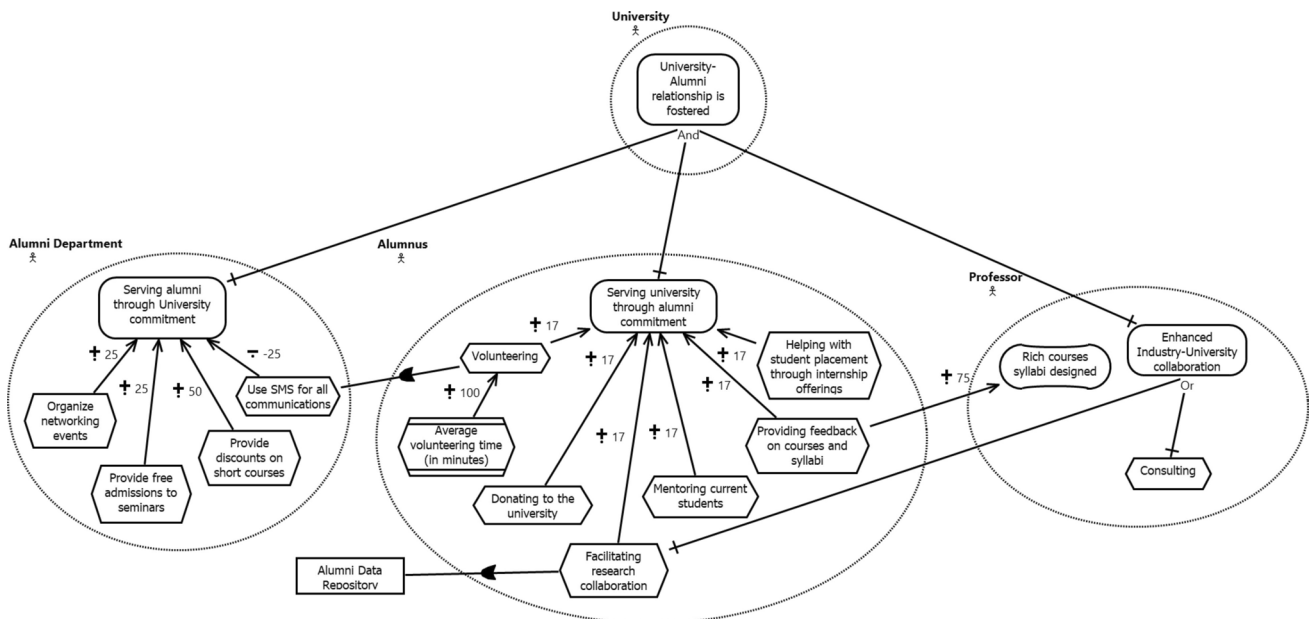


Fig. 25 Input model MF5

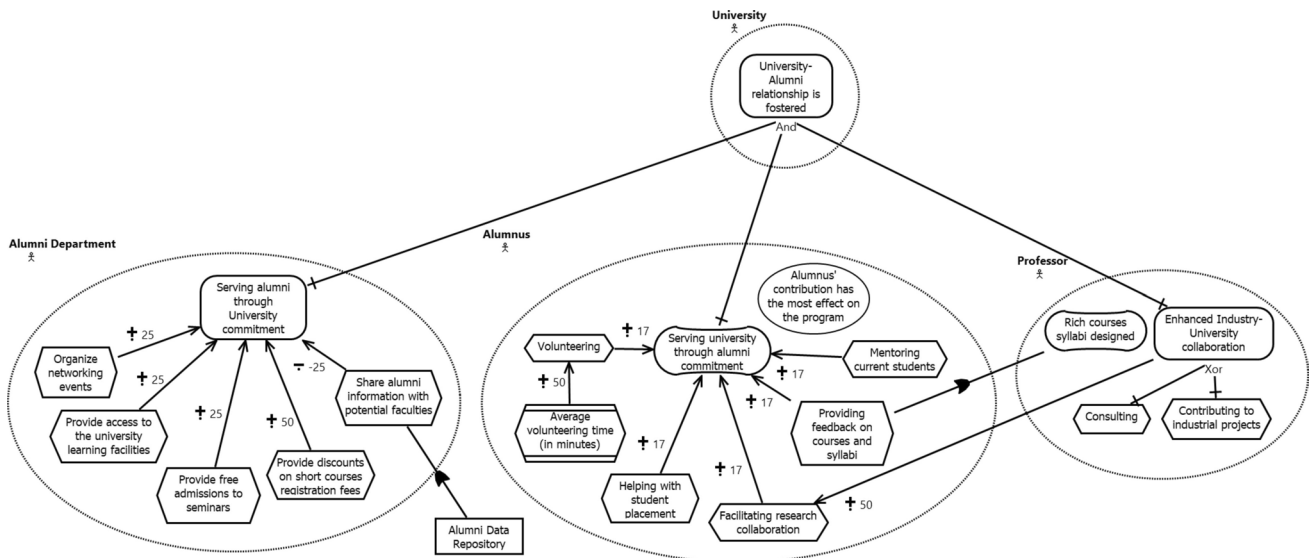


Fig. 26 Input model MF6

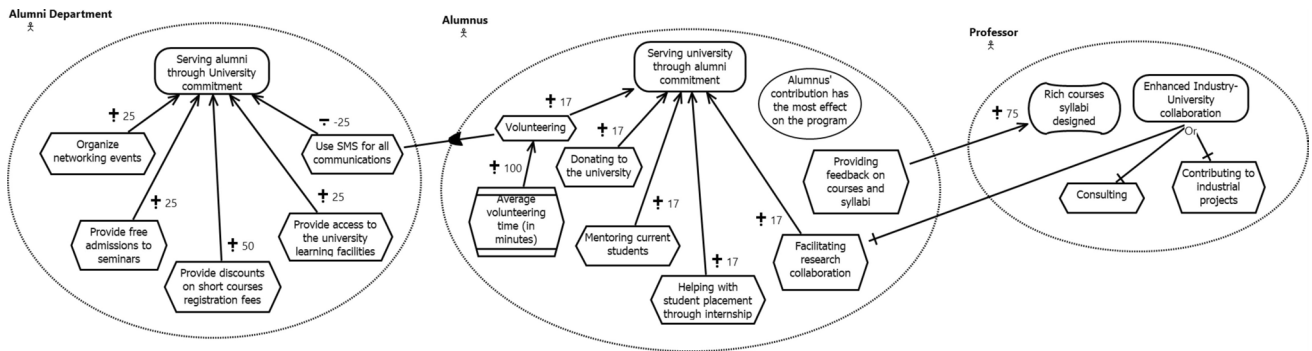


Fig. 27 Input model MF7

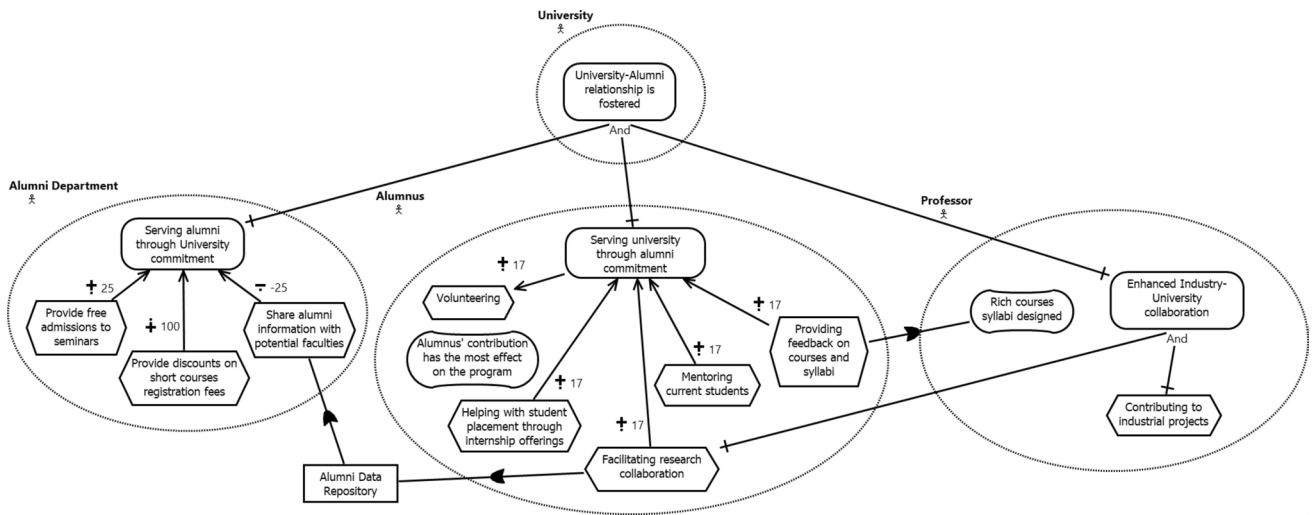


Fig. 28 Input model MF8

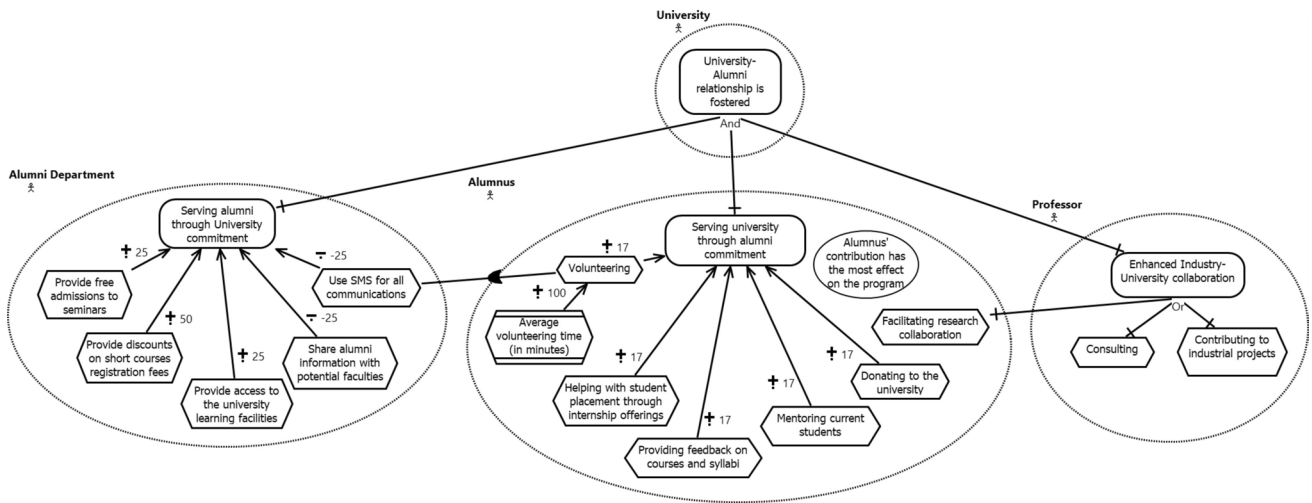


Fig. 29 Input model MF9

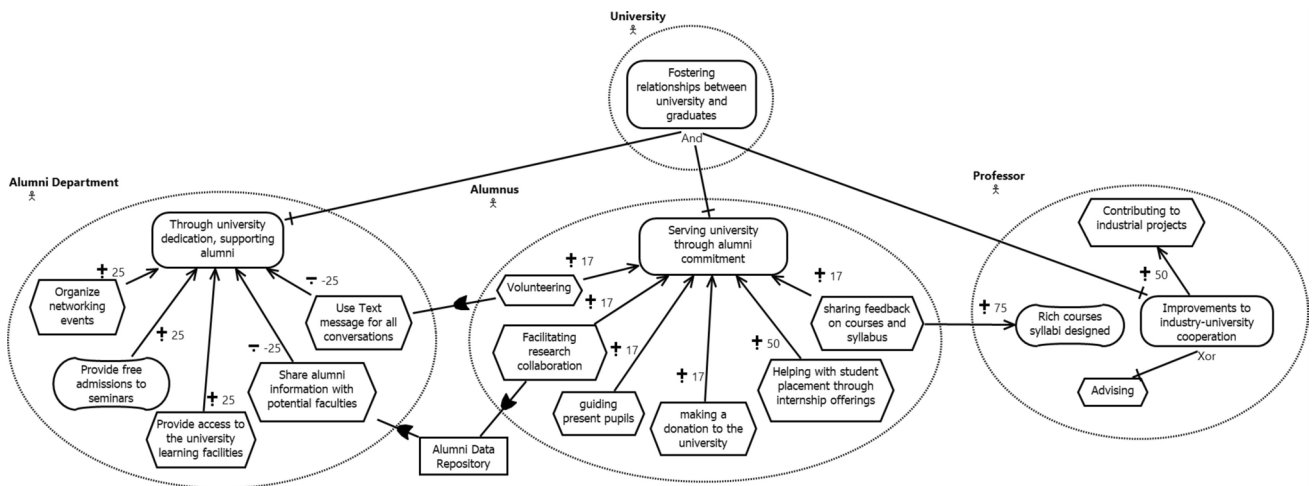


Fig. 30 Input model: MF10

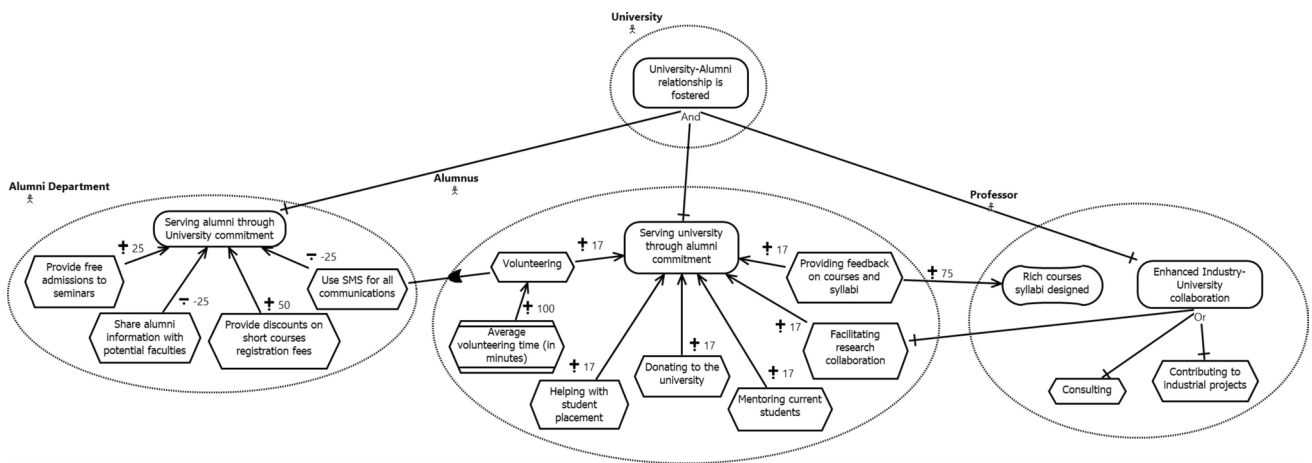


Fig. 31 Input model MF11

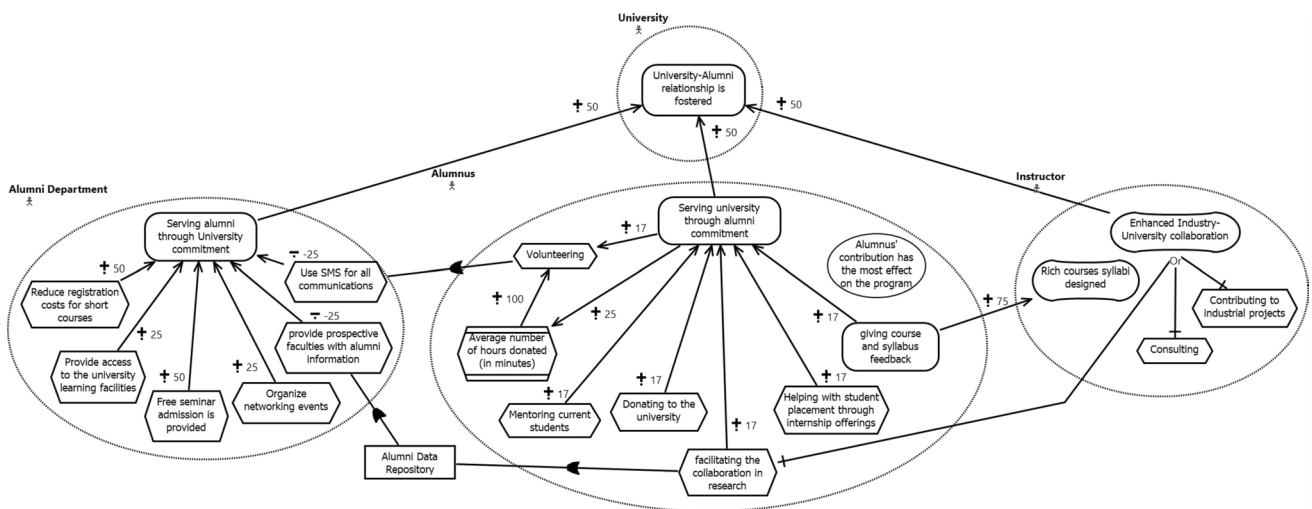


Fig. 32 Input model MF12

Appendix B: seminars managing system GRL sub-models

See Figs. 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43 and 44.

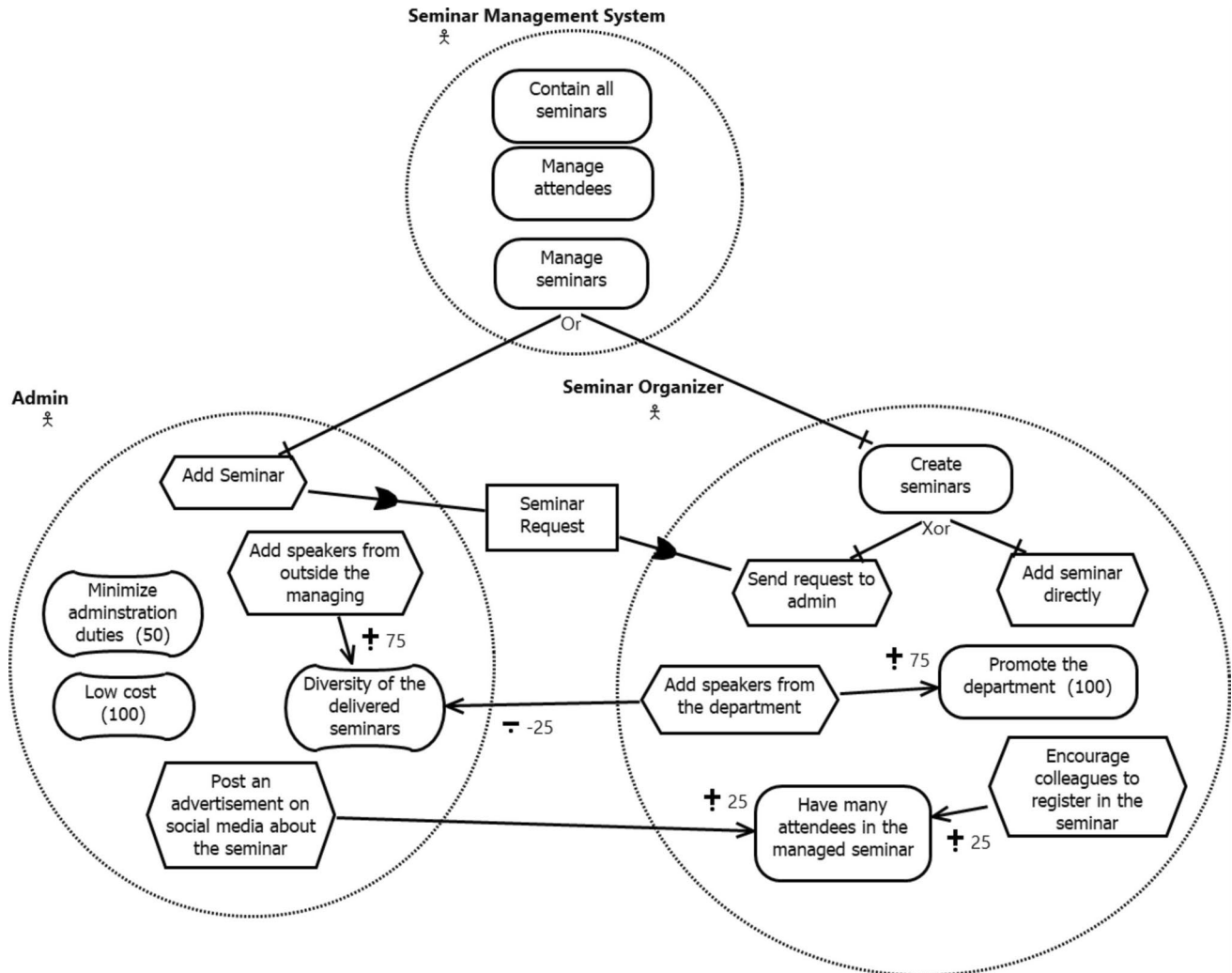


Fig. 33 Input model MS1

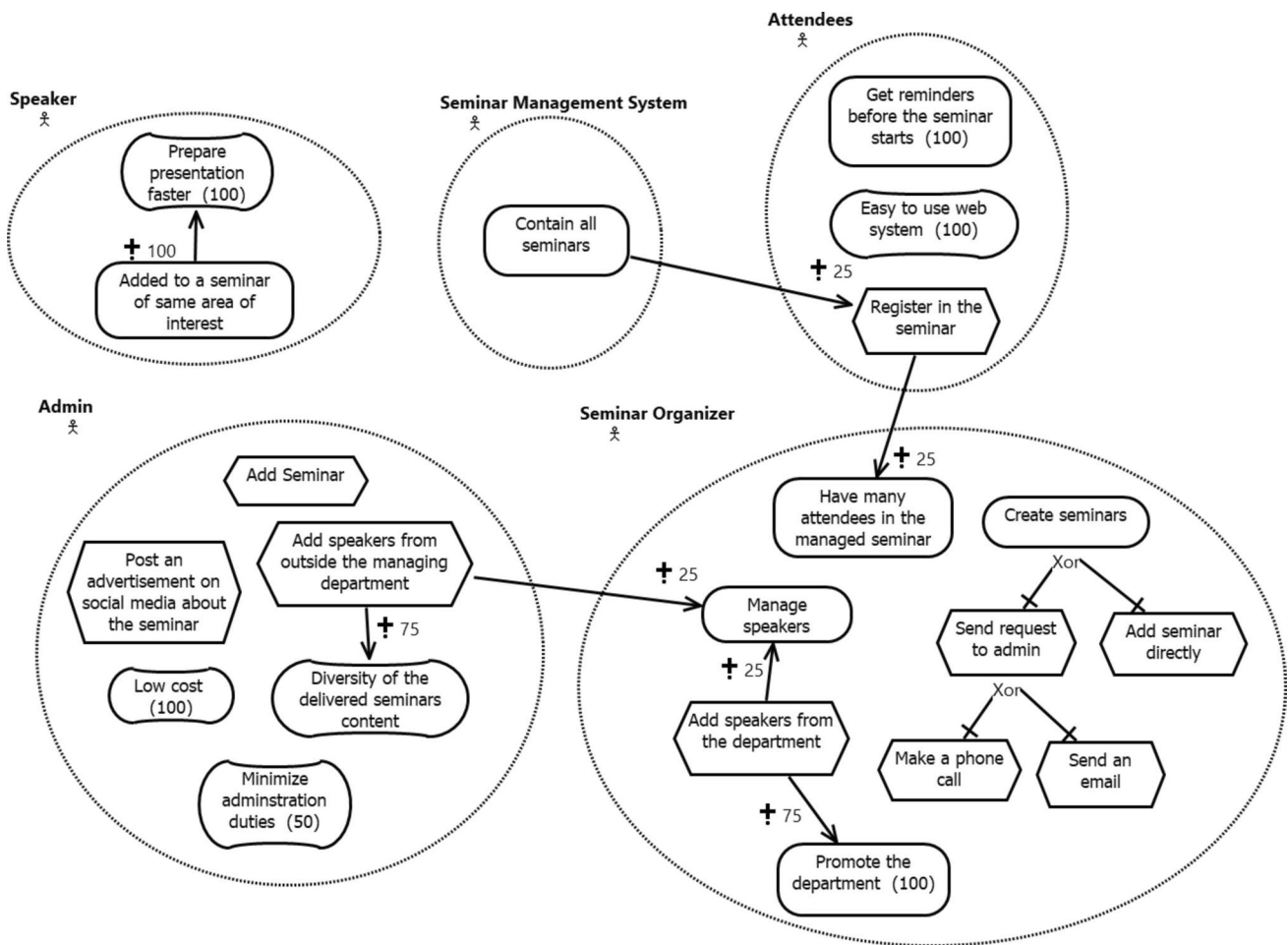


Fig. 34 Input model MS2

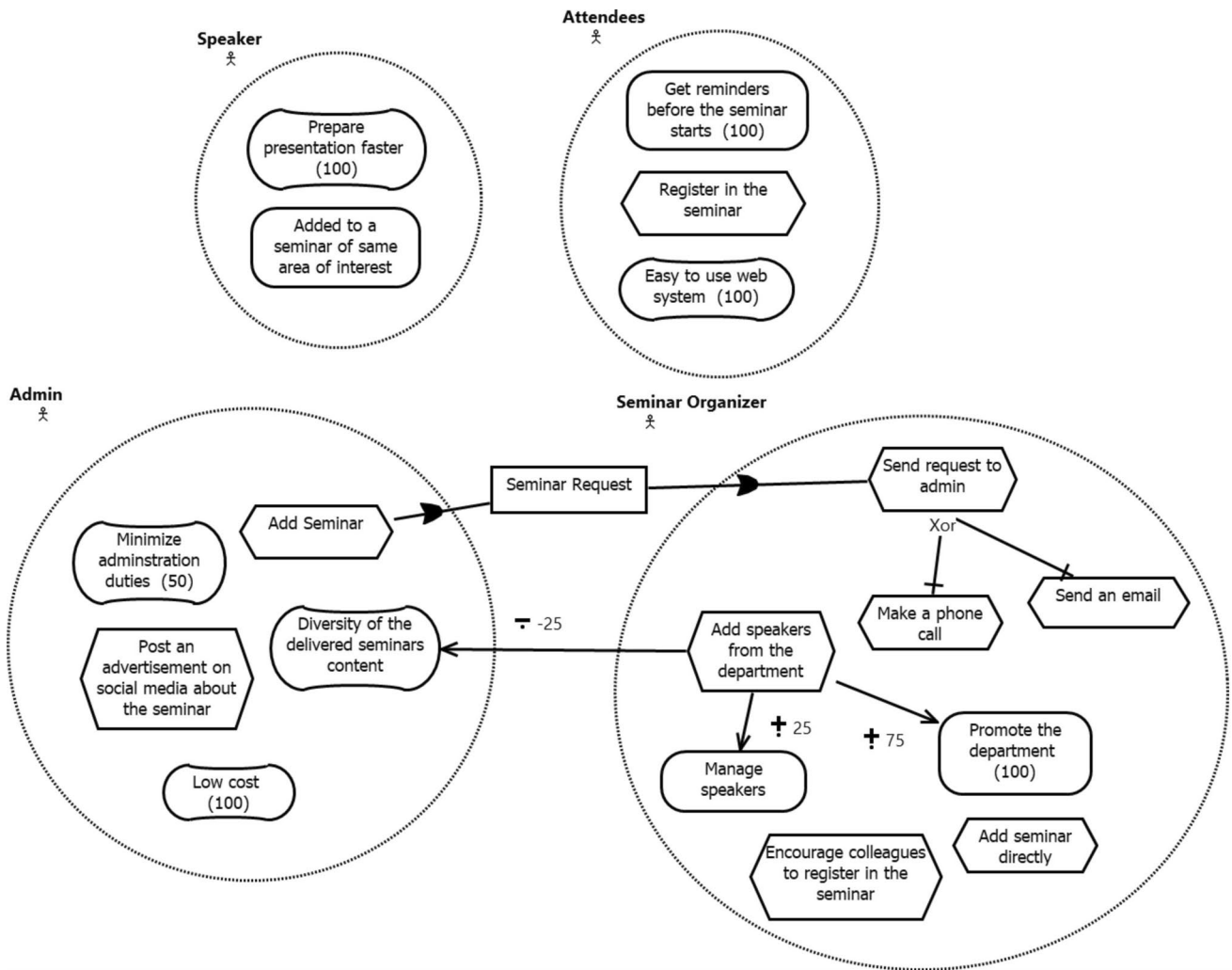


Fig. 35 Input model MS3

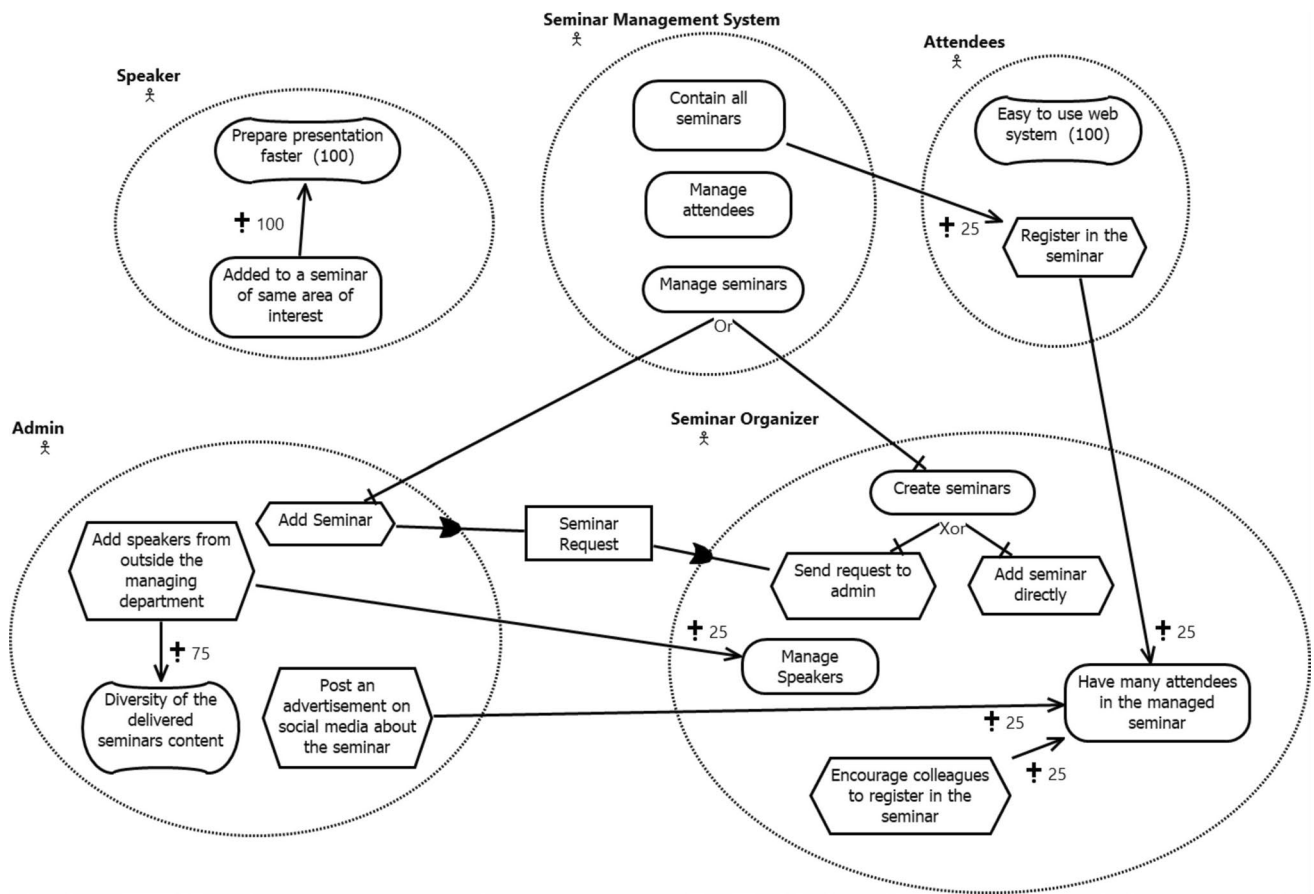


Fig. 36 Input model MS4

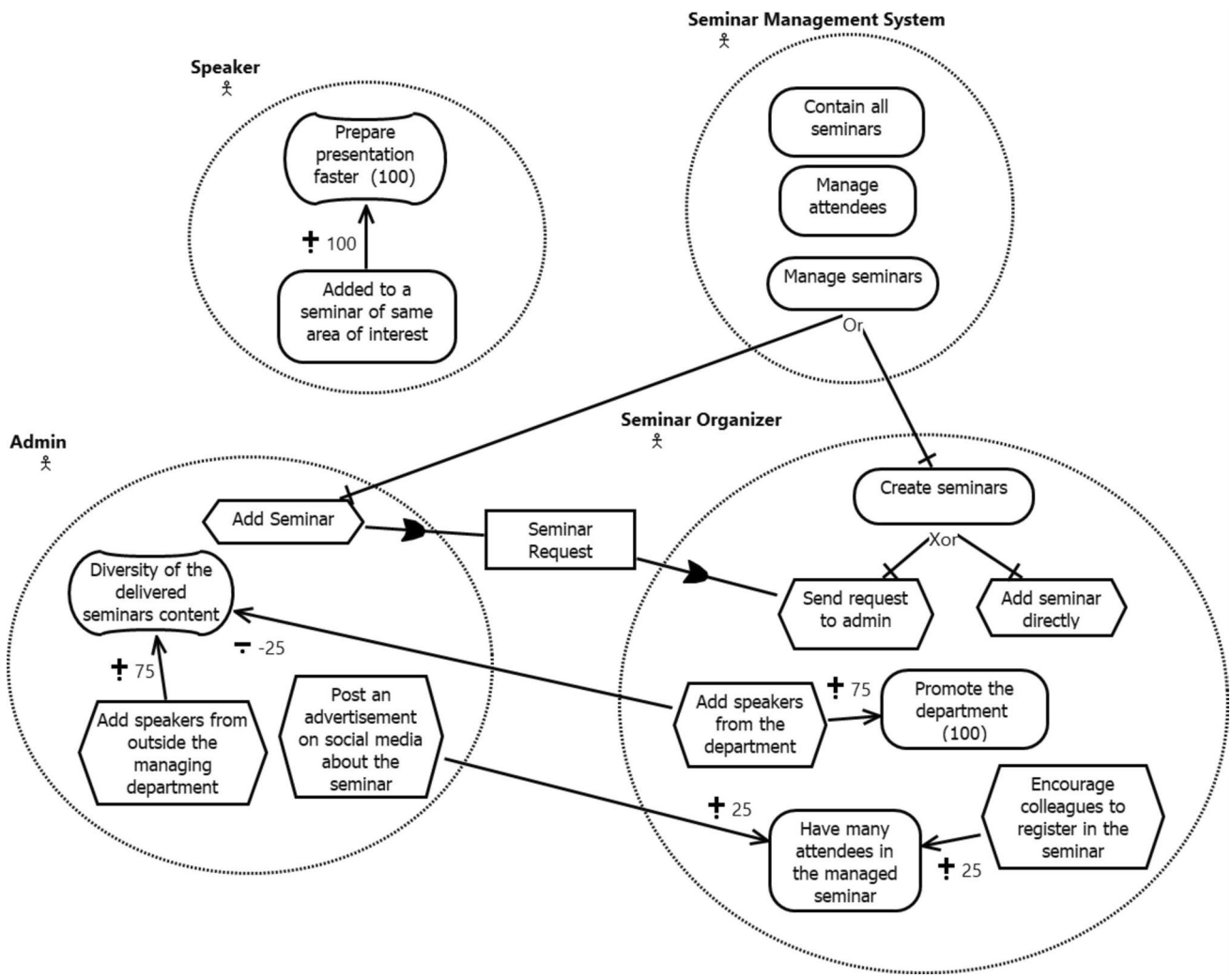


Fig. 37 Input model MSS

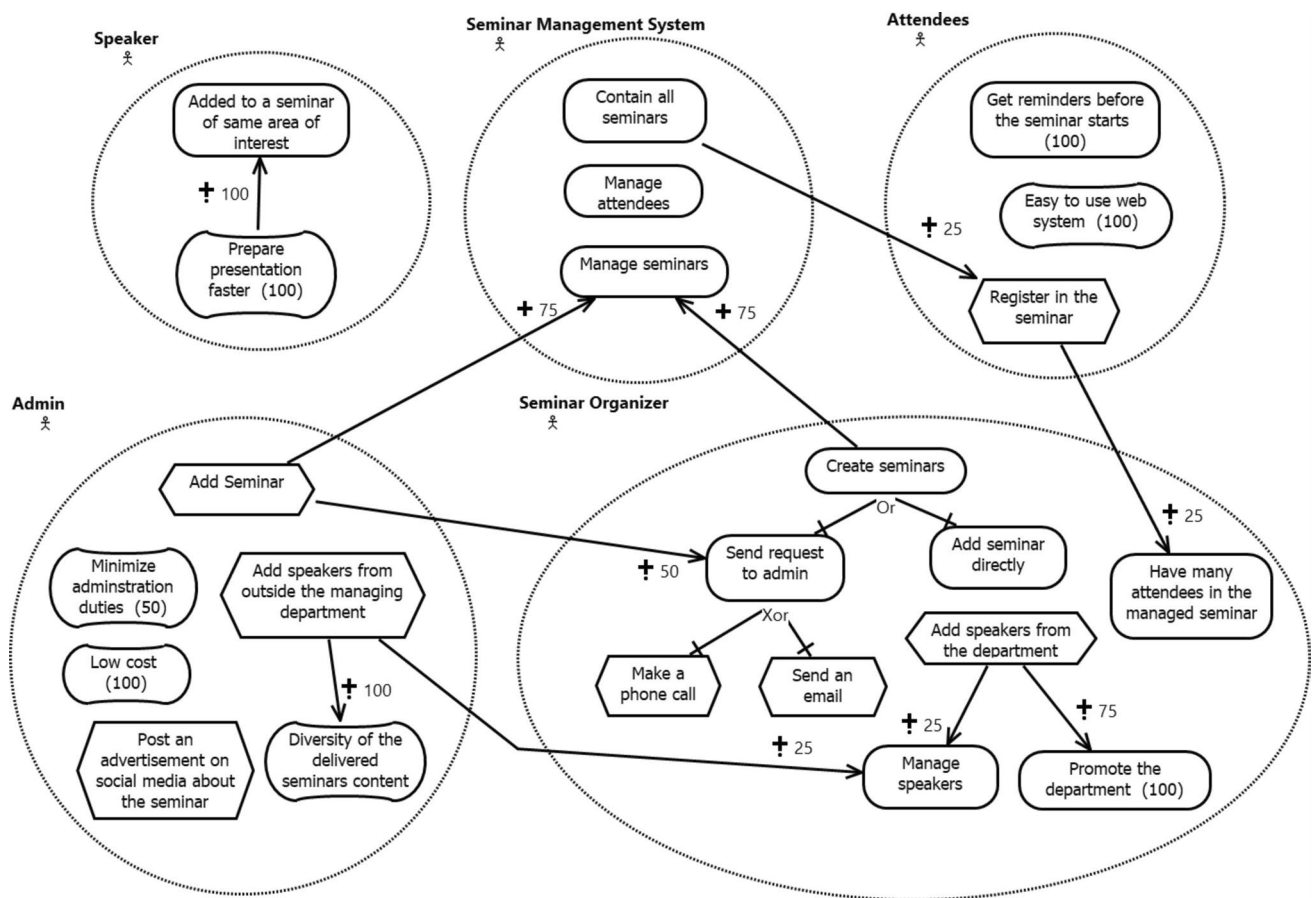


Fig. 38 Input model MS6

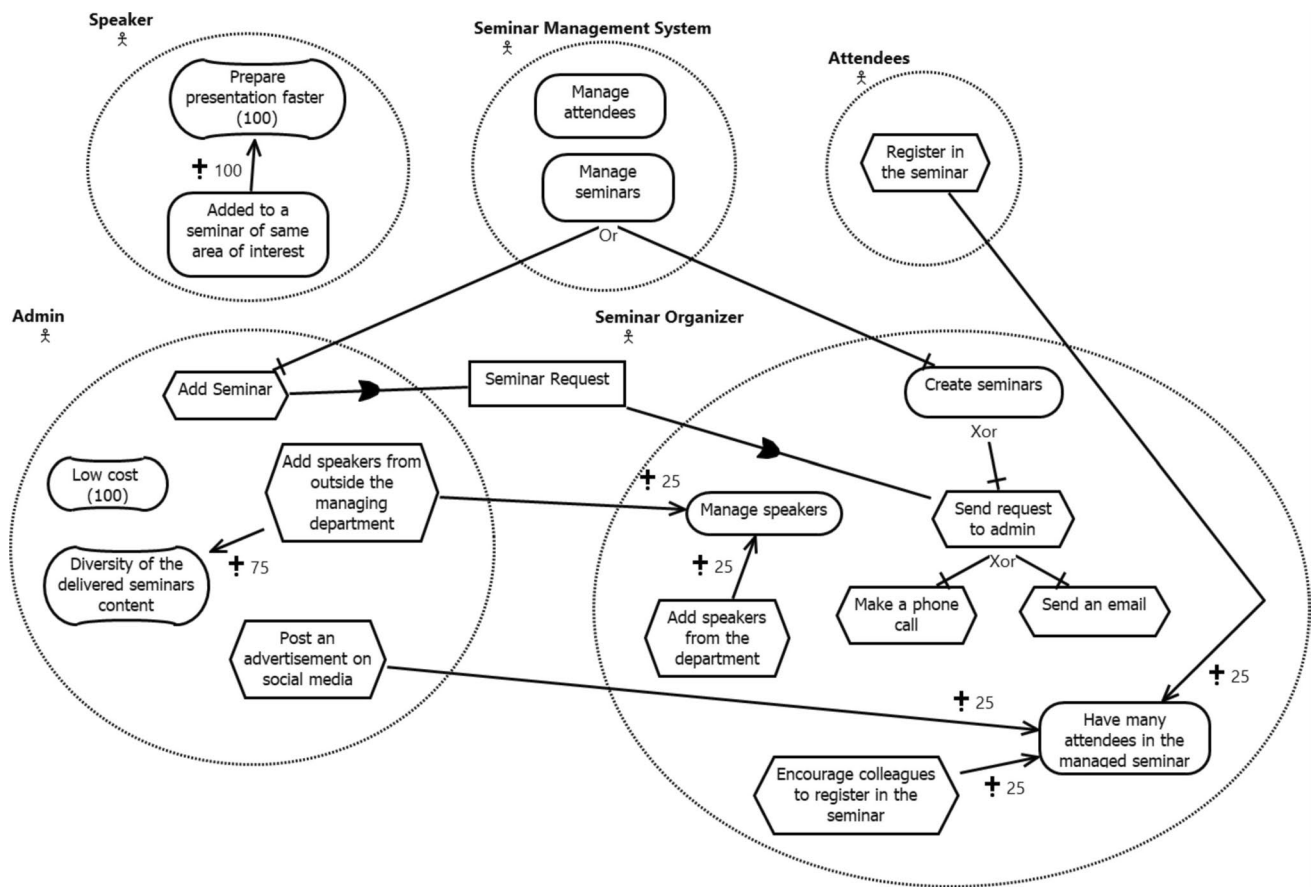


Fig. 39 Input model MS7

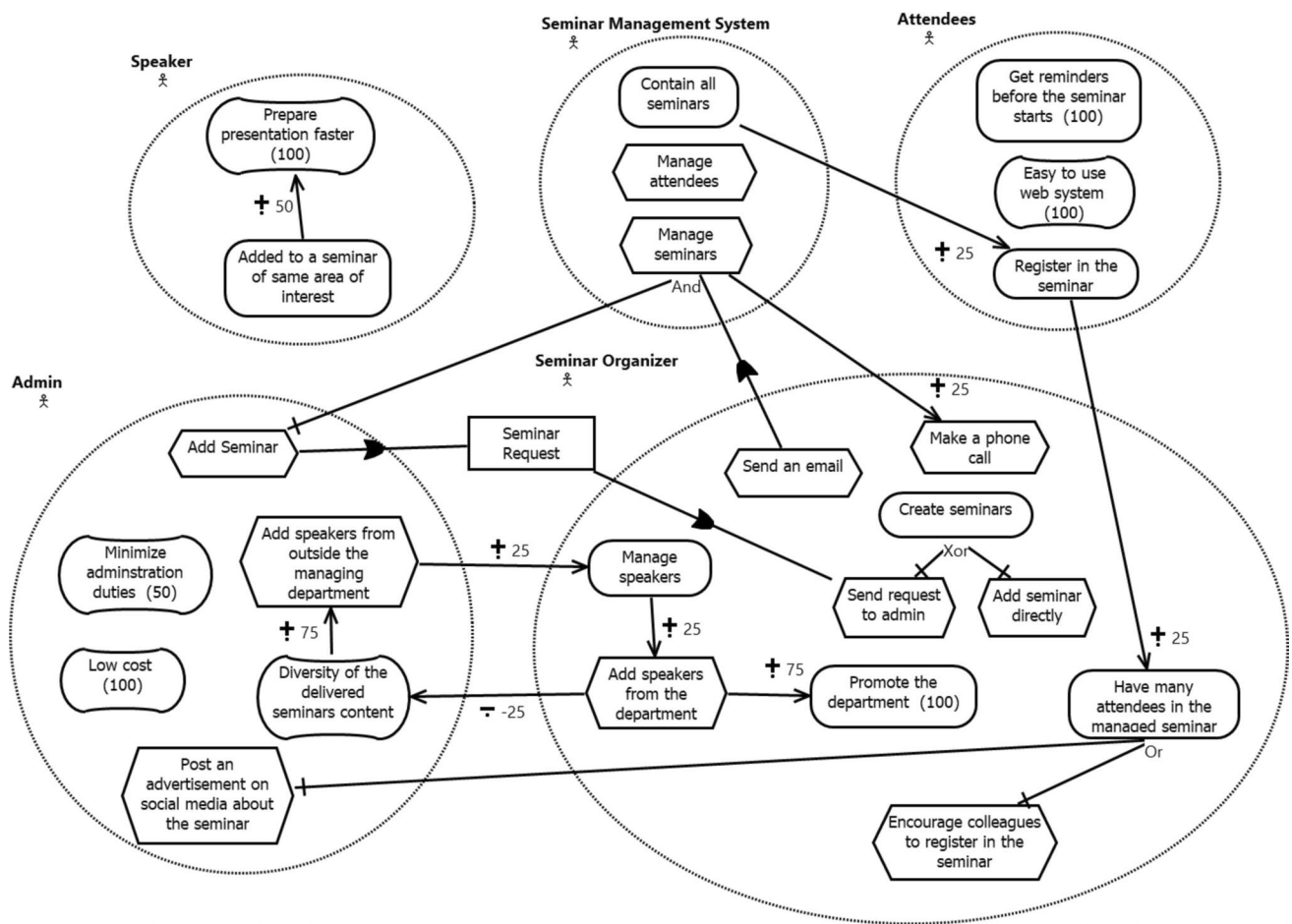


Fig. 40 Input model MS8

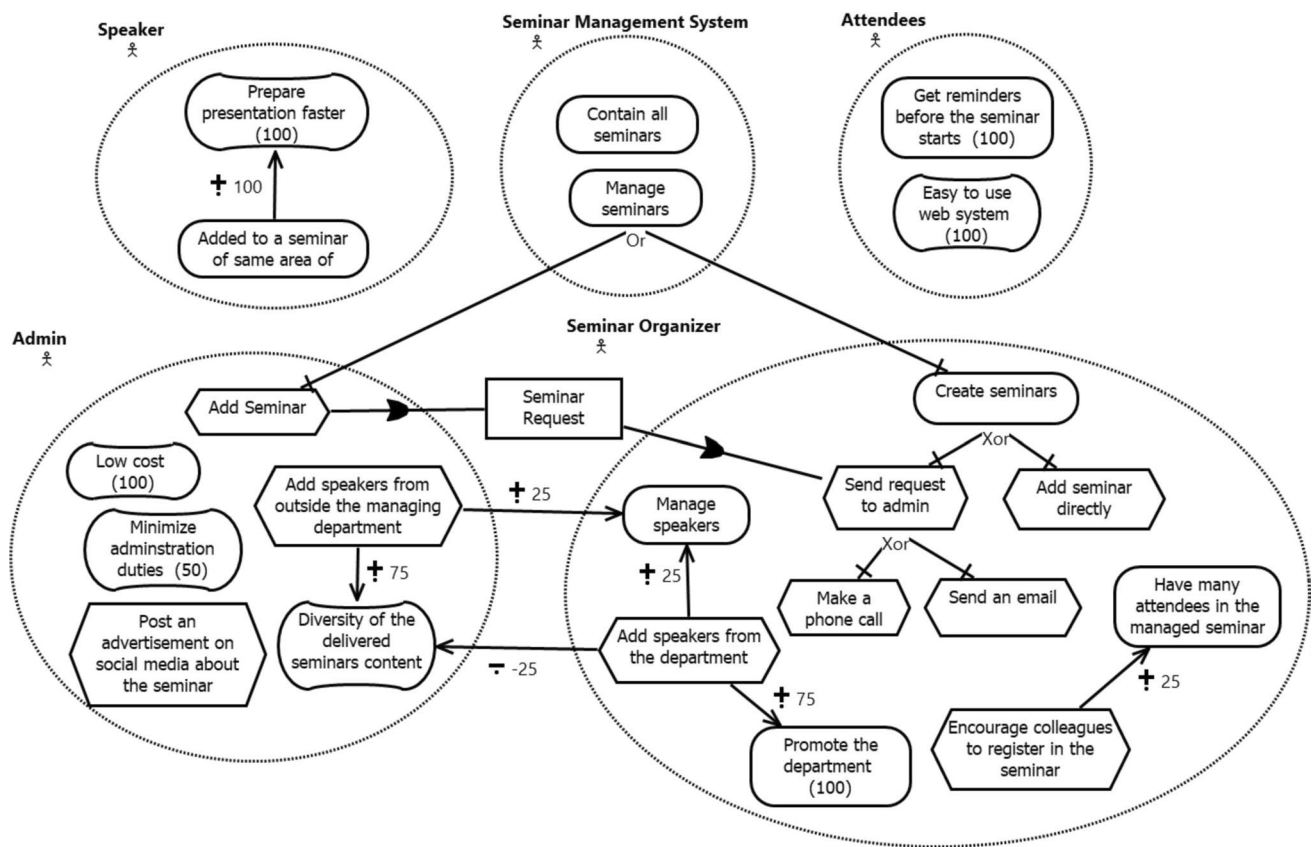


Fig. 41 Input model MS9

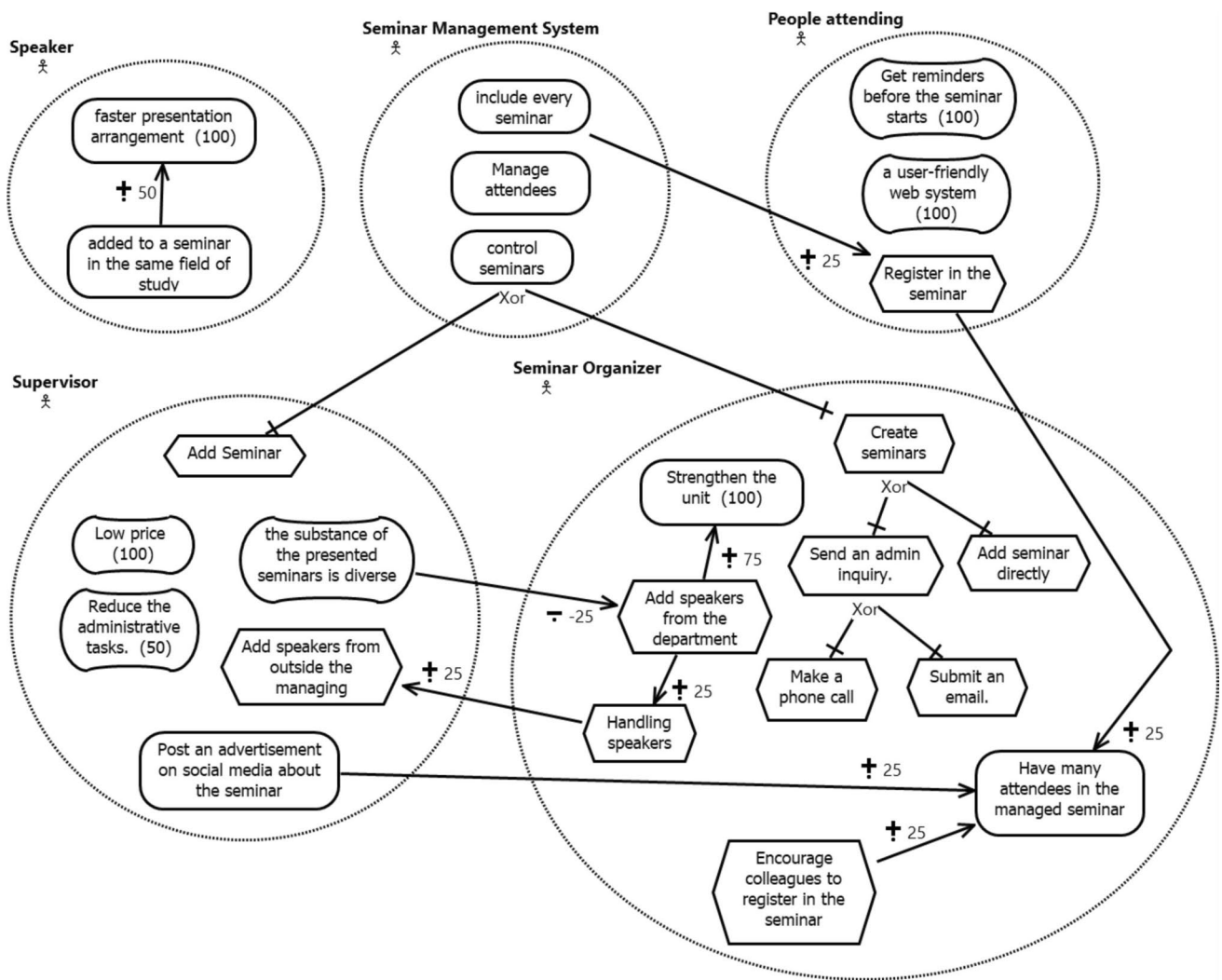


Fig. 42 Input model MS10

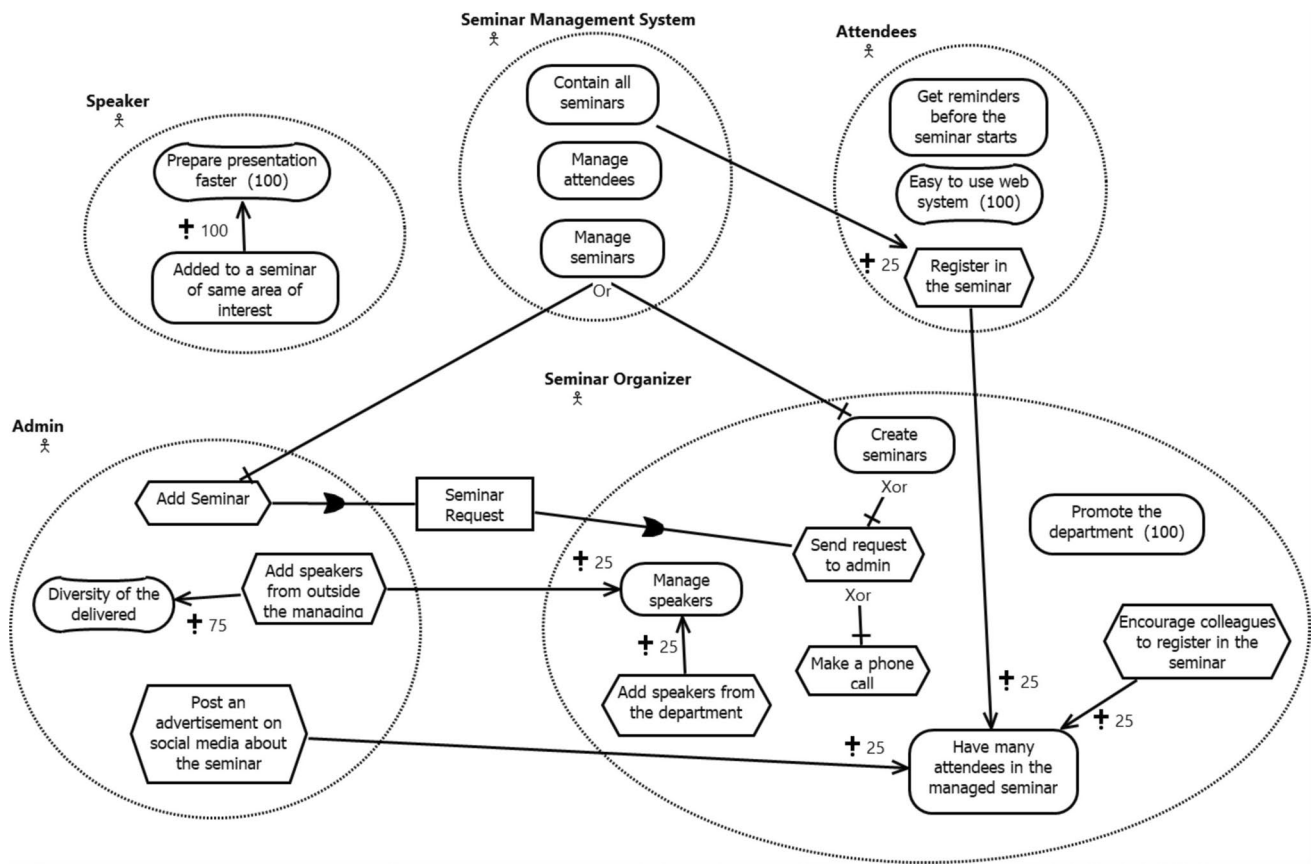


Fig. 43 Input model MS11

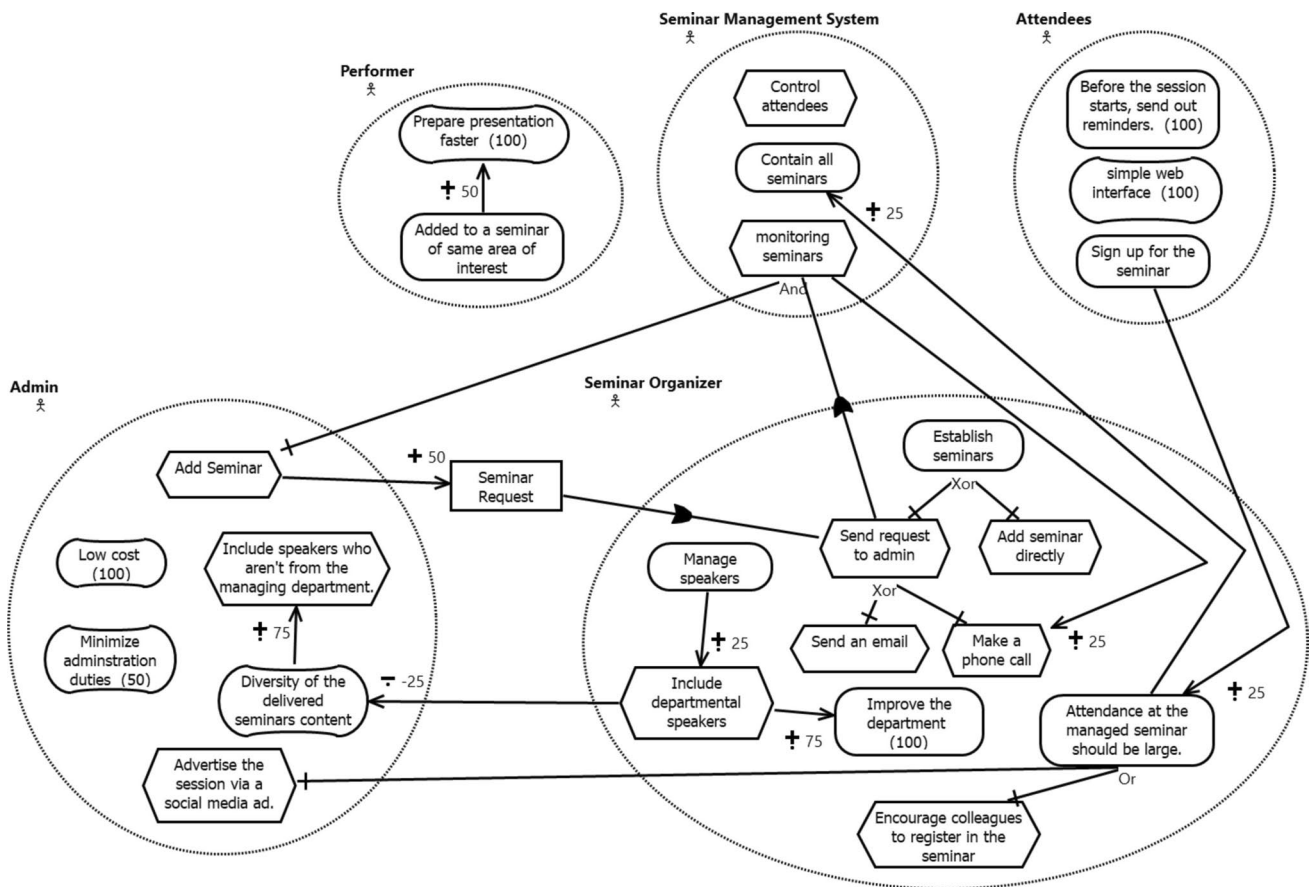


Fig. 44 Input model MS12

Funding Funding was provided by King Fahd University of Petroleum and Minerals (Grant No. SB201022).

Declarations

Conflict of interest All authors declare that they have no conflicts (financial and non-financial) of interest.

References

- Abdelzad V, Amyot D, Alwidian SA, Lethbridge T (2015) A textual syntax with tool support for the goal-oriented requirement language. In: *iStar*, pp 61–66
- Baslyman M, Amyot D (2019) Goal model integration: advanced relationships and rationales documentation. In: *International conference on system analysis and modeling*. Springer, pp 183–199
- Beckers K, Faßbender S, Heisel M, Paci F (2013) Combining goal-oriented and problem-oriented requirements engineering methods. In: *International conference on availability, reliability, and security*. Springer, pp 178–194
- Boronat A, Carsí JA, Ramos I, Letelier P (2007) Formal model merging applied to class diagram integration. *Electron Notes Theor Comput Sci* 166:5–26
- Brunet G, Chechik M, Easterbrook S, Nejati S, Niu N, Sabetzadeh M (2006) A manifesto for model merging. In: *Proceedings of the 2006 international workshop on global integrated model management*, pp 5–12
- Cer D, Yang Y, Kong S, Hua N, Limtiaco N, John RS, Constant N, Guajardo-Cespedes M, Yuan S, Tar C, Strophe B, Kurzweil R (2018) Universal sentence encoder for English. In: Blanco E, Lu W (eds) *Proceedings of the 2018 conference on empirical methods in natural language processing, EMNLP 2018: system demonstrations*, Brussels, Belgium, October 31–November 4, 2018. Association for Computational Linguistics, pp 169–174. <https://doi.org/10.18653/V1/D18-2029>
- Chung L, Nixon BA, Yu E, Mylopoulos J (2012) *Non-functional requirements in software engineering*, vol 5. Springer, Berlin
- Conneau A, Kiela D, Schwenk H, Barrault L, Bordes A (2017) Supervised learning of universal sentence representations from natural language inference data. In: Palmer M, Hwa R, Riedel S (eds) *Proceedings of the 2017 conference on empirical methods in natural language processing, EMNLP 2017*, Copenhagen, Denmark, September 9–11, 2017. Association for Computational Linguistics, pp 670–680. <https://doi.org/10.18653/V1/D17-1070>
- Darimont R, Ponsard C, Lemoine M (2018) Goal-driven elaboration of OCL enriched UML class diagrams. In: *MODELS workshops*, pp 118–131
- Das S, Deb N, Cortesi A, Chaki N (2021) Sentence embedding models for similarity detection of software requirements. *SN Comput Sci* 2(2):1–11
- Deb N, Chaki N (2020) Goal model maintenance. In: *Business standard compliance and requirements validation using goal models*. Springer, pp 81–130
- Diaconescu A, Frey S, Müller-Schloer C, Pitt J, Tomforde S (2016) Goal-oriented holonics for complex system (self-) integration: concepts and case studies. In: *2016 IEEE 10th international conference on self-adaptive and self-organizing systems (SASO)*. IEEE, pp 100–109
- Feng Z, He K, Peng R, Wang J, Ma Y (2009) Towards merging goal models of networked software. In: *SEKE*, pp 178–184
- Giorgini P, Mylopoulos J, Sebastiani R (2005) Goal-oriented requirements analysis and reasoning in the tropos methodology. *Eng Appl Artif Intell* 18(2):159–171
- Gomaa WH, Fahmy AA et al (2013) A survey of text similarity approaches. *Int J Comput Appl* 68(13):13–18
- Hablutzel KR, Jain A, Grubb AM (2022) A divide & conquer approach to collaborative goal modeling with merge in early-re. In: *2022 IEEE 30th international requirements engineering conference (RE)*, pp 14–25. IEEE
- Hassine J, Amyot D (2016) A questionnaire-based survey methodology for systematically validating goal-oriented models. *Requir Eng* 21(2):285–308. <https://doi.org/10.1007/s00766-015-0221-7>
- Ilyas M, Kung J (2009) A similarity measurement framework for requirements engineering. In: *2009 fourth international multi-conference on computing in the global information technology*. IEEE, pp 31–34
- ITU-T (2018) Recommendation Z.151 (10/18) User Requirements Notation (URN) language definition, Geneva, Switzerland. <http://www.itu.int/rec/T-REC-Z.151/en>
- jUCMNav: v7.0.0. <https://github.com/JUCMNAV/>, University of Ottawa, Canada. Last Accessed May 2023
- Jurafsky D, Martin JH (2000) *Speech & language processing*. Pearson Education India, Bengaluru
- Kumar R, Mussbacher G (2018) Textual user requirements notation. In: *System analysis and modeling. languages, methods, and tools for systems engineering: 10th international conference, SAM 2018*, Copenhagen, Denmark, October 15–16, 2018, proceedings 10. Springer, pp 163–182
- Li R, Zhao X, Moens MF (2022) A brief overview of universal sentence representation methods: a linguistic view. *ACM Comput Surv (CSUR)* 55(3):1–42
- Li T, Horkoff J, Mylopoulos J (2014) Integrating security patterns with security requirements analysis using contextual goal models. In: *IFIP working conference on the practice of enterprise modeling*. Springer, pp 208–223
- Li T, Mylopoulos J (2014) Modeling and applying security patterns using contextual goal models. In: *iStar*. Citeseer
- Liu Y, Liu L, Liu H, Gao S (2020) Combining goal model with reviews for supporting the evolution of apps. *IET Softw* 14(1):39–49
- Mandelin D, Kimelman D, Yellin D (2006) A Bayesian approach to diagram matching with application to architectural models. In: *Proceedings of the 28th international conference on software engineering*, pp 222–231
- Maoz S, Ringert JO, Rumpe B (2010) A manifesto for semantic model differencing. In: *International Conference on model driven engineering languages and systems*. Springer, pp 194–203
- Melnik S (2004) *Generic model management: concepts and algorithms*. In: *Lecture notes in computer science*, vol 2967. Springer. <https://doi.org/10.1007/b97859>
- Melnik S (2004) *Generic model management: concepts and algorithms*, vol 2967. Springer, Berlin
- Mihany FA, Moussa H, Kamel A, Ezzat E, Ilyas M (2016) An automated system for measuring similarity between software requirements. In: *Proceedings of the 2nd Africa and middle east conference on software engineering*, pp 46–51
- Miller GA, Beckwith R, Fellbaum C, Gross D, Miller KJ (1990) Introduction to WordNet: an on-line lexical database*. *Int J Lexicogr* 3(4):235–244. <https://doi.org/10.1093/ijl/3.4.235>
- Mohagheghi P, Gilani W, Stefanescu A, Fernandez MA (2013) An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empir Softw Eng* 18(1):89–116

34. Mohammed MA, Alshayeb M, Hassine J (2022) A search-based approach for detecting circular dependency bad smell in goal-oriented models. *Softw Syst Model* 21(5):2007–2037
35. Nejati S, Sabetzadeh M, Chechik M, Easterbrook S, Zave P (2007) Matching and merging of statecharts specifications. In: 29th international conference on software engineering (ICSE'07). IEEE, pp 54–64
36. Petersohn D, Ma WW, Lee DJL, Macke S, Xin D, Mo X, Gonzalez J, Hellerstein JM, Joseph AD, Parameswaran AG (2020) Towards scalable dataframe systems. *Proc VLDB Endow* 13(11):2033–2046
37. Pottinger RA, Bernstein PA (2003) Merging models based on given correspondences. In: Proceedings 2003 VLDB conference. Elsevier, pp 862–873
38. Reimers N, Gurevych I (2019) Sentence-bert: sentence embeddings using siamese bert-networks. In: Inui K, Jiang J, Ng V, Wan X (eds) Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3–7, 2019. Association for Computational Linguistics, pp 3980–3990. <https://doi.org/10.18653/V1/D19-1410>
39. Sabetzadeh M, Easterbrook S (2005) An algebraic framework for merging incomplete and inconsistent views. In: 13th IEEE international conference on requirements engineering (RE'05). IEEE, pp 306–315
40. Sabetzadeh M, Easterbrook S (2006) View merging in the presence of incompleteness and inconsistency. *Requir Eng* 11(3):174–193
41. Van Lamsweerde A (2008) Requirements engineering: from craft to discipline. In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, pp 238–249
42. Wang J, Dong Y (2020) Measurement of text similarity: a survey. *Information* 11(9):421
43. Wang J, He K, Gong P, Wang C, Peng R, Li B (2008) Rgps: a unified requirements meta-modeling frame for networked software. In: Proceedings of the 3rd international workshop on applications and advances of problem frames, pp 29–35
44. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell
45. Wright HK, Kim M, Perry DE (2010) Validity concerns in software engineering research. In: Proceedings of the FSE/SDP workshop on future of software engineering research. ACM, pp 411–414. <https://doi.org/10.1145/1882362.1882446>
46. Yu ES (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of ISRE'97: 3rd IEEE international symposium on requirements engineering. IEEE, pp 226–235. <https://doi.org/10.1109/ISRE.1997.566873>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.