RationalGRL: A Framework for Argumentation and Goal Modeling

Marc van Zee[1], Sepideh Ghanavati[2], and Floris Bex[3]

[1]Computer Science and Communication (CSC)
University of Luxembourg
marcvanzee@gmail.com

[2]Department of Computer Science
Texas Tech University, USA
sepideh.ghanavati@ttu.edu

[3]Department of Information and Computing Sciences
Utrecht University
f.j.bex@uu.nl

Abstract. Goal modeling languages capture the relations between an information system and its environment using high-level goals and their relationships with lower level goals and tasks. The process of constructing a goal model usually involves discussions between a requirements engineer and a group of stakeholders. While it is possible to capture part of this discussion process in a goal model, for instance by specifying alternative solutions for a goal, not all of the arguments can be found in the resulting model. For instance, the discussion on whether to accept or reject a certain goal and the ultimate rationale for acceptance or rejection cannot be captured in current goal modeling languages. Based on a case study in which stakeholders discuss requirements for a Traffic Simulator, we apply argumentation techniques from artificial intelligence to a goal modeling approach. Thus, we combine a traditional goal modelling approach, the Goal-oriented Requirements Language (GRL), with a formal Practical Reasoning Argument Scheme (PRAS) for reasoning about goals into a new framework (RationalGRL). RationalGRL provides a methodology, formal semantics and tool support to capture the discussions and outcomes of the argumentation process that leads to a goal model.

Keywords Goal modeling · Argumentation · Practical Reasoning · Goal-oriented requirements engineering

1 Introduction

Requirements Engineering (RE) is an approach to assess the role of a future information system within a human or automated environment. An important goal in RE is to produce a consistent and comprehensive set of requirements covering different aspects of the system, such as general functional requirements, operational environment constraints, and so-called non-functional requirements such as security and performance.

Among the initial activities in RE are the "early-phase" requirements engineering activities, which include those that consider how the intended system should meet organizational goals, why it is needed, what alternatives may exist, what the implications of the alternatives are for different stakeholders, and how the interests and concerns of stakeholders might be addressed [41]. These activities fall under the umbrella of goal modeling. There are a large number of established RE methods using goal models in the early stage of requirements analysis (e.g., [22, 11, 9, 7, 6], overviews can be found in [34, 19]). Several goal modeling languages have been developed in the last two decades as well. The most popular ones include $i*$ [42], Keep All Objects Satisfied (KAOS) [35], the NFR framework [7], TROPOS [14], the Business Intelligence Model (BIM) [16], and the Goal-oriented Requirements Language (GRL) [2].

A goal model is often the result of a discussion process between a group of stakeholders. For small-sized systems, goal models are usually constructed in a short amount of time, involving stakeholders with a similar background. Therefore, it is often not necessary to record all of the details of the discussion process that led to the final goal model. However, goal models for many complex, real-world information systems – e.g., air-traffic management systems, systems that support industrial production processes, or government and health-

care services – are not constructed in a short amount of time, but rather over the course of several workshops with stakeholders and requirements engineers. In such situations, failing to record the discussions underlying a goal model in a structured manner may harm the success of the RE phase of system development.

The first difficulty is that the goal modeling phase, particularly in large projects, is dynamic and that goal models continuously change and evolve. Stakeholders' preferences are rarely absolute, relevant, stable, or consistent [23], and stakeholders may change their opinion about a modeling decision in between two goal modeling sessions, which in turn may require revisions of the goal model. If the rationales behind these revisions are not properly documented, alternative ideas and opposing views that could potentially have led to different goal models are lost, as the resulting goal model only shows the end product of a long process and not the discussions during the modeling process. Furthermore, other stakeholders, such as developers who were not the original authors of the goal model, may have to make sense of a goal model in order to, for example, use it as input in a later RE stage or in the development phase. If preferences, opinions and rationales behind the goal models are not stored explicitly, it may not only be more difficult to understand the model, but the stakeholders may also end up having the same unnecessary discussions throughout the goal modeling phase.

A further problem is that the rationale behind goal modeling decisions is usually static, that is, current goal modeling languages have limited support for reasoning about changing beliefs and opinions, and their effect on the goal model. A stakeholder may change his or her opinion, but it is not always directly clear what its effect is on the goal model. Similarly, with existing goal modelling languages one can change a part of the goal model, but it is not possible to reason about whether or not this new goal model is consistent with the underlying beliefs and arguments. This becomes even more problematic if the stakeholders constructing the goal model change, since modeling decisions made by one group of stakeholders may conflict with the underlying beliefs of another group of stakeholders. The disconnect between the goal models and their underlying beliefs and opinions may further lead to a poor understanding of the problem and solution domain, which is an important reason of RE project failure [8].

To summarize, what is needed is a way of recording the rationales (beliefs, opinions, discussions, ideas) underlying a goal model. It should be possible to see how these rationales changed during the goal modeling process, and the rationales should be clearly linked to the various elements of the resulting goal model. In order to be able to do this, we propose a framework with tool-support that combines traditional goal modeling approaches with argumentation techniques from Artificial Intelligence (AI) research [4]. We have identified **five important requirements** for our framework:

1. The argumentation techniques should be close to the actual discussions of stakeholders or designers in the early requirements engineering phase.
2. The framework must have formal traceability links between elements of the goal model and underlying arguments.
3. Using these traceability links, it must be possible to compute the effect of changes in the underlying argumentation on the goal model, and vice versa.
4. There should be a methodology for the framework to guide the practitioners in its application in real cases.
5. The framework must have software tool support.

Following on from our previous work [36, 38], we develop a framework called *RationalGRL*, which combines the Goal-oriented Requirements Language (GRL) [2] with a technique from argumentation theory and discourse modeling called *argument schemes* (or argumentation schemes [39]). Argument schemes are reusable patterns of reasoning that capture the typical ways in which humans argue and reason. Associated with argumentation schemes are so-called *critical questions*, which can point to typical sources of doubt or implicit assumptions people make when arguing in a particular way. Argument schemes they are very well suited for modeling discussions about a goal model, as they can guide users in systematically deriving conclusions and making assumptions explicit [26].

One argument scheme that is important when reasoning about goals is the argument scheme for practical reasoning [40, 3], which has been used for, among other things, dialogues about safety critical actions [32] and software design discussions [5]. Inspired by the work on practical reasoning from Artificial Intelligence, most notably Atkinson and Bench-Capon [3], we have developed a list of argument schemes that can be used to analyse and guide stakeholders' discussions about goal models. Our approach thus provides a rationalization to the elements of the goal model in terms of underlying arguments, and helps in understanding why parts of the model have been accepted and others have been rejected. Our list of argument schemes was constructed by performing an extensive case study in which we analyzed a set of transcripts containing more than 4 hours of discussions among designers of a traffic simulator information system. This ensures that the argumentation schemes we propose are close to actual real-world discussions stakeholders have (**requirement 1**).

The meta-model of the RationalGRL framework clearly specifies the traceability links between the arguments based on the schemes and the GRL models (**requirement 2**). In addition to this meta-model, we provide formal semantics for RationalGRL by formalising the GRL language in propositional logic and rendering arguments about a GRL model as a formal argumentation framework [12]. We then formally capture the link between argumentation and goal modelling as a set of algorithms for applying argument schemes and critical questions about goal models. These formal traceability links allow us to compute the effect of the arguments and counterarguments proposed in a discussion on a GRL model (**requirement 3**). In other words, we can determine whether the elements of a GRL model are acceptable given potentially contradictory opinions of stakeholders. Thus, we add a new formal evaluation technique for goal models that allows us to assess the *acceptability* of elements of a goal model (in addition to their *satisfiability* [2]).

Because we want RE practitioners in the field to be able to use our RationalGRL framework (**requirement 4**), we also propose a methodology for using RationalGRL, which consists of developing goal models and posing arguments based on schemes in an integrated way. To show that the RationalGRL methodology can be used in a real case, we illustrate the steps of our methodology with the traffic simulator case study. Finally, we have developed a web-based prototype[1] for building goal models and arguing about them, which acts as a supporting tool to the RationalGRL methodology (**requirement 5**).

TODO for all(by F): check text below when we finish the paper

The rest of this article is organized as follows. Section 2 introduces our running example, the Goal-oriented Requirements Language (GRL) [2], the Practical Reasoning Argument Scheme (PRAS) [3] and discusses some of our previous work on combining GRL and PRAS. Section 4 provides a brief and high-level overview of our framework, together with a metamodel and the methodology. Section 3 contains an in depth explanation of how we obtained an initial set of argument schemes and critical questions by annotating transcripts from discussions about an information system, and in Section **??** we provide several examples of these schemes and questions. In Section 6 we provide formal semantics for GRL and show how argumentation semantics [12] can be used to compute which arguments are accepted and which are rejected. We also develop various algorithms for the argument schemes in this section. In Section 5.2 we provide a brief overview of the prototype tool we developed for RationalGRL. Finally, Section 7 contains a discussion, covering related work, future work, and a conclusion.

[1] `insertURL`

2 Background: Goal-oriented Requirements Language and Argumentation

In this section, we first introduce our running example, after which we introduce the Goal-oriented Requirements Language (GRL) [2], which is the goal modeling language we use to integrate with the argumentation framework. Next, we introduce argumentation: we discuss the *practical reasoning argument scheme (PRAS)* [3], an argument scheme that is used to form arguments and counter-arguments about situations involving goals, and we give informal examples of how argument and counterargument can influence the status of beliefs about goals. Finally, we briefly discuss the possibilities integrating PRAS and GRL.

2.1 Running example: Traffic Simulator

We use a traffic simulator design case to explain the concepts and framework in this paper. Our examples and case study are based on a recent series of experiments by Schriek et al. [29], who in turn base their work on the so-called Irvine experiment [33], which presents a well-known design reasoning assignment in software engineering. In this assignment (see Appendix A), designers are provided with a problem description, requirements, and a description of the desired outcomes: The client of the project is Professor E, who teaches civil engineering courses at an American university. In order for the professor to teach students the various theories concerning traffic (such as queuing theory), traffic simulator software needs to be developed in which students can create visual maps of an area, regulate traffic, and so forth. Schriek and colleagues asked designers (groups of students) to discuss the requirements of this traffic simulator. These discussions were recorded and transcribed, and we used these transcripts for an extensive case study (Section **??** on the basis of which we developed our RationalGRL framework (Section 3). Furthermore, we also use the traffic simulator case for a simple running example in this section (Figures 1, 3).

2.2 Goal-oriented Requirements Language (GRL)

GRL is a visual modeling language for specifying intentions, business goals, and non-functional requirements of multiple stakeholders [2]. GRL is part of the User Requirements Notation, an ITU-T standard, that combines goals and non-functional requirements with functional and operational requirements (i.e. use case maps). GRL

can be used to specify alternatives that have to be considered, decisions that have been made, and rationales for making decisions. A GRL model is a connected graph of intentional elements that optionally are part of the actors. All the elements and relationships used in GRL are shown in Figure 2.

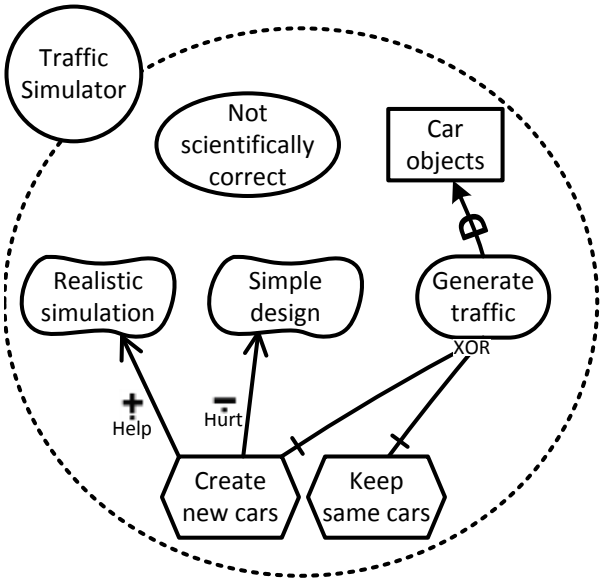

Fig. 1: Partial GRL Model of the traffic simulator example

Figure 1 illustrates a simplified GRL diagram from the traffic simulator design exercise. An actor ( ) represents a stakeholder of a system or the system itself (*Traffic Simulator*, Figure 1). Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available, and softgoals to be satisfied. Softgoals ( ) differentiate themselves from goals ( ) in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable, often in a binary way. Softgoals (e.g. *Realistic simulation*) are often related to non-functional requirements, whereas goals (such as *Generate Cars*) are related to functional requirements. Tasks ( ) represent solutions to (or operationalizations of) goals and softgoals. In Figure 1, we have the two tasks *Create new cars* and *Keep same cars*: in order to achieve goal *Generate cars*, the simulation can either constantly generate new ones or keep the same cars and have them reappear after they disappear off screen. In order to be achieved or completed, softgoals, goals, and tasks may require resources ( ) to be available (e.g., *Car Objects*). Finally, design rationale can be captured using beliefs ( ). For example, the idea that the simulation should be *Not scientifically correct* is captured as a belief in the model.

Different links connect the elements in a GRL model. AND, IOR (Inclusive OR), and XOR (eXclusive OR) decomposition links ( ) allow an element to be decomposed into sub-elements. In Figure 1, the goal *Generate cars* is XOR-decomposed to the tasks *Create new cars* and *Keep same cars*, as they are alternative ways of achieving the goal *Generate cars*. Contribution links ( ) indicate desired impacts of one element on another element. A contribution link has a qualitative contribution type or a quantitative contribution. Task *Create new cars* has a *help* qualitative contribution to the task *Realistic simulation*, and a *hurt* qualitative contribution to the task *Simple design*. Dependency links ( ) model relationships between actors or resources. Here, the goal *Generate cars* depends on the resource *Car objects*.

GRL is based on $i*$ [42] and the NFR Framework [7], but it is not as restrictive as $i*$. Intentional elements and links can be more freely combined, the notion of agents is replaced with the more general notion of actors, i.e., stakeholders, and a task does not necessarily have to be an activity performed by an actor, but may also describe properties of a solution. GRL has a well-defined syntax and semantics. Furthermore, GRL provides support for providing a scalable and consistent representation of multiple views/diagrams of the same goal model (see [13, Ch.2] for more details). GRL is also linked to Use Case Maps via URNLinks ( ), which provide traceability between concepts and instances of the goal model and behavioral design models. Multiple views and traceability links are a good fit with our current research: we aim to add traceability links between intentional elements and their underlying arguments.

GRL has six evaluation algorithms which are semi-automated and allow the analysis of alternatives and design decisions by calculating the satisfaction value of the intentional elements quantitatively, qualitatively or in a hybrid way. jUCMNav, GRL tool-support, also allows for adding new GRL evaluation algorithms [27]. GRL also has the capability to be extended through metadata, links, and external OCL constraints. This allows GRL to be used in many domains without the need to change the whole modeling language.

The GRL model in Figure 1 shows the softgoals, goals, tasks and the relationship between the different intentional elements in the model. However, the rationales and arguments behind certain intentional elements are not shown in the GRL model. Some of the questions that might be interesting to know about are the following:

- Why is belief *Not scientifically correct* not linked to any of the goals or tasks?

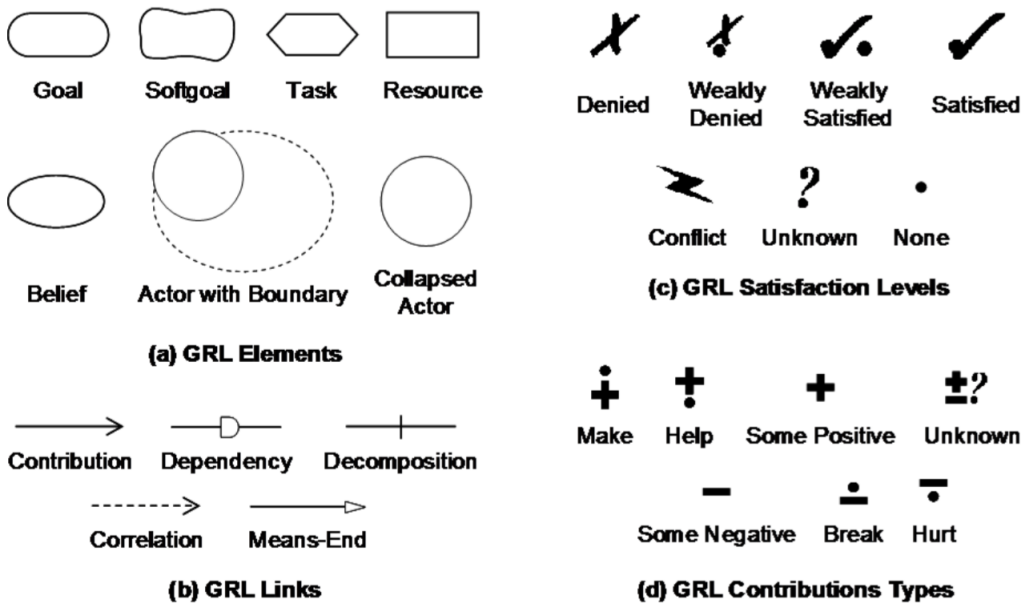

Fig. 2: Basic elements and relationships of GRL

- What does *Keep same cars* mean?
- Why does task *Create new cars* contribute negatively to *Simple design* and positively to *Realistic simulation*?
- Why does *Generate cars* XOR-decompose into two tasks?

These are the type of the questions that we cannot answer by just looking at GRL models. The model in Figure 1 does not contain information about discussions that led to the resulting model, such as various clarification steps for the naming, or alternatives that have been considered for the relationships. The idea behind the original GRL specification is that beliefs can be used to capture such design rationales that make later justification and review of a model easier. However, beliefs cannot be connected to links - this makes answering the third and fourth question above impossible. Furthermore, beliefs are after-the-fact design rationales and do not capture the types of critical questions given above.

2.3 Practical Reasoning Argument Scheme (PRAS)

Reasoning about which goals to pursue and actions to take is often referred to as *practical reasoning*, and has been studied extensively in philosophy and artificial intelligence. One approach is to capture practical reasoning with argument schemes [40]. Applying an argument scheme results in an argument in favor of, for example, taking an action. This argument can then be tested with critical questions about, for instance, whether the action is possible given the situation, and a negative answer to such a question leads to a counterargument to the original argument for the action.

Atkinson and Bench-Capon [3] develop and formalize the *Practical Reasoning Argument Scheme* (PRAS). A simplified version of this argument scheme is as follows:

> G is a goal,
> Performing action A realizes goal G,
> *Therefore*
> Action A should be performed

Here, G and A are variables, which can be instantiated with concrete goals and actions to provide a specific practical argument. For example, a concrete argument about the traffic simulator is as follows:

> *Generate Cars* is a goal,
> Performing action *Keep same cars* realizes goal *Generate cars*,
> Therefore
> Action *Keep same cars* should be performed

Note that PRAS is an argument scheme that captures a full inference step: "G, A realizes G *Therefore* A". There are, however, also schemes that capture simpler reasoning patterns, such as claims of the form "A does not realize G". We will discuss these schemes below.

In argumentation, conclusions which are at one point acceptable can later be rejected because of new information. For example, we may argue that, in fact, performing action *Keep same cars* does not realize goal *Generate cars*, thus giving a counterargument to the above

instantiation of PRAS. Atkinson et al. [3] define a set of so-called critical questions that point to typical ways in which an argument based on PRAS can be criticized by. Some examples of critical questions are as follows.

CQ1 Will the action realize the desired goal?
CQ2 Are there alternative ways of realizing the same goal?
CQ3 Does performing the action have a negative side effect?

The idea is that answers to critical questions are counterarguments to the original PRAS argument. These counterarguments also follow a scheme; for example, a negative answer to CQ1 follows the scheme "Action A will not realize goal G", which can be instantiated (e.g. "*Keep same cars* does not realize *Generate cars"*) to form a counterargument to the original argument.

Another way to criticize an argument for an action is to suggest an alternative action that realizes the same goal (CQ2). For example, we can argue that performing *Create new cars* also realizes the goal *Generate cars*. Also, it is possible that performing an action has a negative side effect (CQ3). For example, while the action *Create new cars* realizes the goal *Generate cars*, it has a negative side effect, namely hurting *Simple design*: having the simulation constantly create new cars is fairly complex design choice.

In argumentation, counterarguments are said to *attack* the original arguments. Given a set of arguments and attacks between these arguments, we can compute which arguments are accepted and which are rejected using different argumentation semantics [12][2]. Figure 3 shows three arguments from the traffic simulation example, where arguments are rendered as boxes and attack relations as arrows. There are two arguments based on PRAS: argument A1 for *Keep Same Cars* and argument A2 for *Create new cars*. Argument A2 proposes an alternative way of realizing the same goal *Generate cars* with respect to argument A1 and vice versa (cf. CQ2), so A1 and A2 mutually attack each other, denoted by the arrows between A1 and A2. Argument A3 says that *Create new cars* has a negative effect on *Static Simulation*, so A3 attacks A2, as it points to a negative side-effect of *Create new cars* (CQ3). The intuition here is that an argument is acceptable if any argument that attacks it is itself rejected. In Figure 3, argument A3 is accepted because it has no attackers. This makes A2 rejected (indicated by the lighter grey color), because its attacker A3 is accepted. A1 is then also accepted, since its only attacker, A2, is rejected.

[2] Formal definitions of argumentation frameworks and semantics will be given in section 3. In this section, we will briefly discuss the intuitions behind these concepts.

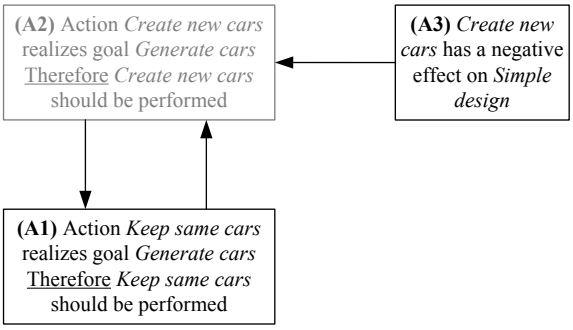

Fig. 3: PRAS arguments and attacks in the traffic simulation example.

Looking at PRAS and its critical questions, one can see how it could be used to argue about goals and actions or, more specifically, about goal models. However, we cannot literally use PRAS and its critical questions, as there are elements of the GRL language, such as actors and resources, which cannot be found in PRAS. Furthermore, it is not directly clear whether the critical questions as proposed by Atkinson and Bench-Capon [3] actually apply to GRL models. In fact, our case study (Section 3) shows that when discussing requirements, people very often do not structure their reasoning nicely in the way that PRAS presents it. That is, you do not see the discussants setting up an argument "We have goal G, A realizes G *Therefore* we should perform A". A typical discussion is much more unstructured, as is clear from the transcript excerpts in Appendix B. So if we would use the version of PRAS presented in this section for our argumentation, we would violate requirement 1: The argumentation techniques should be close to the actual discussions of stakeholders or designers in the early requirements engineering phase. Our solution was to develop our own set of argument schemes and critical questions by analyzing transcripts of discussions about the traffic simulator. This set of schemes and questions and our case study are described in the next section.

3 Argument Schemes for Goal Modeling: a Case Study

Recall that **requirement 1** of our RationalGRL framework is that the argumentation techniques should be close to the actual discussions of stakeholders or designers in the early requirements engineering phase. In order to get a sense of such discussions, we performed an extensive case study. The objective of our case study was to see which types of discourse are used during a discussion of system requirements, and how these discourse types can be captured as argument schemes and critical

questions regarding goals and tasks. In order to study this, we manually coded transcripts of such discussions. For our coding, we used a list of argument schemes and critical questions based on GRL and PRAS. In this section we present our case study. All original transcripts, codings, and models are available in our online repository[3].

In order to obtain actual requirements discussions, we turned to a recent series of experiments by Schriek et al. [29]. In these experiments, Schriek and colleagues gave the traffic simulator assignment (Appendix A) to 12 groups of two or three students in a Software Architecture course at MSc level. These groups had a maximum of two hours to design a traffic simulator, which included a discussion of the requirements of this traffic simulator. The students did not use any goal modeling technique in the course or during the discussions. The students were asked to record their design sessions, and the recordings were subsequently transcribed. We used three of these transcripts, totaling 153 pages, for our case study.

Before we started coding the transcripts, we drew up an initial list of 10 argument schemes (AS0-AS9 in Table 1), representing *claims* about the goal model containing the requirements of the system. AS0 to AS4 are schemes that concern a single element of the goal model. So, for example, AS0 represents the claim 'a is a relevant actor for the system', and AS3 represents the claim 'G is a goal for the system'. AS5 to AS9 are claims about the links between GRL elements. Our initial list also contained 18 critical questions, inspired by the questions associated with the original Practical Reasoning Argumentation Scheme [3]. CQ2 to CQ6b (Table 1) are examples of these critical questions, other examples are 'Is the softgoal legitimate?' and 'Are there alternative ways to contribute to the same softgoal?'.

Using the initial list of arguments and critical questions, we coded three transcripts of requirements discussions. The codings were performed by one author and subsequently checked by another author. As the transcripts contain spoken language, the codings involved some interpretation. For example, the students almost never literally say 'actor a has task T'. Rather, they say things such as '...we have a set of actions. Save map, open map, ...' (Table 3, Appendix B) and '... in that process there are activities like create a visual map, create a road' (Table 4, Appendix B). Furthermore, in some cases the critical questions are explicit. For example, CQ10b is found in the transcripts as '...is this an OR or an AND?' (Table 5, Appendix B). In other cases, however, the question remains implicit but we added it in the coding. For example, CQ12 is not found in the transcripts, but the related counterargument is: '...you don't have to specifically add a traffic light' (Table 3, Appendix B).

[3] **TODO for M(by F): `insertURL`**

Scheme/Question		t_1	t_2	t_3	**total**
AS0	Actor	2	2	5	**9**
AS1	Resource	2	4	5	**11**
AS2	Task/action	20	21	17	**58**
AS3	Goal	0	2	2	**4**
AS4	Softgoal	3	4	2	**9**
AS5	Goal decomposes into tasks	4	0	4	**8**
AS6	Task contributes (negatively) to softgoal	8	3	0	**11**
AS7	Goal contributes (negatively) to softgoal	0	1	1	**2**
AS8	Resource contributes to task	0	4	3	**7**
AS9	Actor depends on actor	0	1	3	**4**
AS10	Task decomposes into tasks	11	14	11	**36**
CQ2	Task is possible?	2	2	1	**5**
CQ5a	Does the goal decompose into the tasks?	0	1	0	**1**
CQ5b	Goal decomposes into other tasks?	1	0	0	**1**
CQ6b	Task has negative side effects?	2	0	0	**2**
CQ10a	Task decompose into other tasks?	1	2	0	**3**
CQ10b	Decomposition type correct?	1	0	1	**2**
CQ11	Is the element relevant/useful?	2	3	2	**7**
CQ12	Is the element clear/unambiguous?	3	10	3	**16**
-	Generic counterargument	0	2	2	**4**
TOTAL		69	80	69	**222**

Table 1: Occurrences of argument schemes and critical questions in the transcripts.

During the coding, new argument schemes and critical questions were added to the list. For example, we found that the discussants often talk about tasks decomposing into sub-tasks, so we added AS10 and CQ10a. Furthermore, because there were many discussions on the relevance and the clarity of the names of elements (goals, tasks, etc.), two generic critical questions CQ12 and CQ13 were added. The final results of the coding can be found in Table 1; for results per transcript, please consult the online repository. We found a total of 159 instantiation of argument schemes AS0-AS11 in transcripts. The most used argument scheme was AS2: "Actor A has task T", however, each argument scheme is found in transcripts at least twice. A large portion (about 60%) of argument schemes involved discussions about tasks of the information system (AS2, AS10). We coded 41 applications of critical questions. Many critical questions (about 55%) involved clarifying the name of an element, or discussing its relevance (CQ12, CQ13).

Our coding further led us to identify three different operations, that is, different effects an argument or critical question can have on a goal model: an argument can introduce a new goal model element (INTRO); it can disable (i.e. attack) a goal model element (DISABLE); or it can replace a goal model element (REPLACE).

Consider, for example, Table 3 in Appendix B. First, an argument is posed that introduces a number of tasks. Counterarguments are then given against two of these tasks (*add traffic light* and *remove intersection*), which are subsequently disabled. An example of replacement is given in Table 5 (Appendix B): what used to be an AND-decomposition is changed into an OR-decomposition. We will further discuss the possible operations in Section 4.

4 The RationalGRL Framework

Having explored the types of arguments and questions in requirements discussions, we now turn to an overview of our RationalGRL framework. We will show through a language definition and informal examples from our case study that it is possible to trace elements of the goal model back to their underlying arguments (**requirement 2**), and that it is possible to determine the effect of changes in the underlying argumentation on the goal model, and vice versa (**requirement 3**). A more formal, logical rendition of this traceability can be found in Section 6.

The RationalGRL framework includes two parts: Argumentation and GRL goal modeling. The GRL part of RationalGRL allows for the creation of GRL goal models by analyzing the non-functional requirements in the requirements specification document and by refining the high-level goals into operationalized tasks. For the argumentation part, arguments and counterarguments can be put forward about various parts of this goal model. These two parts, GRL and argumentation, are developed iteratively and each side can impact the other side so that the models can be refined or new critical questions and argument schemes can be instantiated. For example, answering a critical question *Is the task* A *possible?* can result in removing or adding a task in the GRL model. Similarly, if, for example, we add a new intentional element to the GRL model, it can lead to a new critical question relevant to this intentional element and its relationships.

Figure 4 presents an overview of RationalGRL framework. At the bottom there are two activities: *practical reasoning & argumentation* and *goal model construction*. As already explained, these two activities influence each other: adding elements to a goal model gives rise to new critical questions about these elements, and answering critical questions with arguments may add or delete elements from the GRL model. The activities give rise to two different models: a *RationalGRL model* (left-hand side) and a *GRL model* (right-hand side). In the RationalGRL model, we have the goal model elements (goals, tasks etc) and the arguments for and against these elements; the GRL model contains only the goal model elements. Thus, the class of GRL models is essentially a subset of the class of RationalGRL models. Given the framework, it then becomes possible to trace a goal model back to the original argumentative discussion about goals, tasks and requirements.

In the rest of this section, we discuss the individual parts of the GRL framework. In Section 4.1, we continue our discussion of the argument schemes and critical questions for practical reasoning and argumentation, fitting these schemes and questions into our framework. In Section 4.2, we then discuss the language for RationalGRL models. In Section 4.3, we then provide extensive examples from our case study, illustrating the interplay between practical reasoning and argumentation on the one hand and RationalGRL models on the other hand.

4.1 The RationalGRL Argument Schemes and Critical Questions for Practical Reasoning

A core aspect of the RationalGRL framework are the argument schemes, which should be close to the actual types of reasoning stakeholders or designers perform in the early requirements engineering phase (requirement 1 of our framework). Recall from section 3 that we ended up with a list of argument schemes and critical questions that were found in the transcripts (Table 1). Using this list as a basis, we further refined our set of argument schemes and critical questions for RationalGRL into the list shown in Table 2. Note that this list of argument schemes and critical questions is not exhaustive. It is an initial list that we have obtained by coding transcripts. However, our framework is fully extensible, meaning that new argument schemes and critical questions can be added depending on the problem domain.

Schemes AS0-AS4 and AS12-AS13 are arguments for an element of a goal model, and AS5-AS11 are related to links in a goal model. The last scheme (Att) is a scheme for a generic counterargument against any type of argument that has been put forward. Arguments based on these schemes can be used to discuss a GRL goal model. Making an argument based on one of the

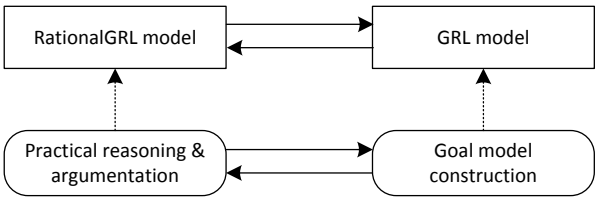

Fig. 4: The RationalGRL Framework

Argument scheme		Critical Questions		Effect
AS0	a is an actor	CQ0	Is the actor relevant?	DISABLE (no)
AS1	Actor a has resource R	CQ1	Is the resource available?	DISABLE (no)
AS2	Actor a can perform task T	CQ2a	Is the task possible?	DISABLE (no)
		CQ2b	Does the task contribute negatively to some softgoal?	DISABLE (yes)
AS3	Actor a has goal G	CQ3	Can the desired goal be realized?	DISABLE (no)
AS4	Actor a has softgoal S	CQ4	Is the softgoal a legitimate softgoal?	DISABLE (no)
AS5	Goal G decomposes into task T	CQ5a	Does the goal decompose into the task?	DISABLE (no)
		CQ5b	Does the goal decompose into other tasks?	INTRO (yes)
		CQ5c	Is the decomposition type correct?	REPLACE (no)
AS6	Task T contributes (negatively) to softgoal S	CQ6a	Does the task contribute to the softgoal?	DISABLE (no)
		CQ6b	Are there alternative ways of contributing to the same softgoal?	INTRO (yes)
		CQ6c	Does the task contribute (negatively) to some other softgoal?	INTRO (yes)
AS7	Goal G contributes to softgoal S	CQ7a	Does the goal contribute to the softgoal?	DISABLE (no)
		CQ7b	Does the goal contribute to some other softgoal?	INTRO (yes)
AS8	Task T depends on resource R	CQ8	Is the resource required in order to perform the task?	DISABLE (no)
AS9	Actor a depends on actor b	CQ9	Does the actor depend on any actors?	INTRO (yes)
AS10	Task T_i decomposes into task T_j	CQ10a	Does the task decompose into the task?	DISABLE (no)
		CQ10b	Does the task decompose into other tasks?	INTRO (yes)
		CQ10c	Is the decomposition type correct?	REPLACE (no)
AS11	Element IE is relevant	CQ11	Is the element relevant/useful?	DISABLE (no)
AS12	Element IE has name n	CQ12	Is the name clear/unambiguous?	REPLACE (no)
Att	Generic counterargument	Att	Generic counterargument	DISABLE

Table 2: List of argument schemes (AS0-AS13), critical questions (CQ0-CQ12), and the effect of answering them (right column).

schemes effectively adds the corresponding GRL element to the model. See, for example, Table 3 in Appendix B: the participants argue for the addition of several tasks to the goal model using argument scheme AS2.

As we saw in Sections 2.3 and 3, an important part of arguing about goal models is asking the right critical questions. The critical questions presented in Table 2 are therefore related to their respective argument schemes. These questions can be answered with "yes" or "no", and the type of answer has an effect on the original argument (INTRO, DISABLE, REPLACE). This will be further explained in Section 4.3.

4.2 The RationalGRL Modelling Language

The language used to construct RationalGRL models is an extension of GRL: it includes all the GRL elements shown in Figure 2. However, there are also new elements corresponding to argumentation-related concepts. Figure 5 shows these elements.

- *Argument*: This represents an argument that does not directly correspond to a GRL element.
- *Rejected (Disabled) GRL element*: If an argument or GRL element is attacked by an argument that itself is not attacked, then this GRL element will be rejected.
- *Attack Link*: An attack link can occur between an argument and another argument or GRL element. It means that the source argument attacks the target argument or GRL element.

The complete metamodel of the language can be found in Figure 6. This metamodel represents the abstract grammar of the language, independently of the notation. The metamodel also formalizes the GRL concepts and constructs introduced in Section 2.2[4].

The metamodel consists of two packages, *practical reasoning & argumentation* and *goal model construction*, which correspond to the relevant activities in

[4] Note that for readability, some GRL concepts, such as contribution strength, have been omitted from the figure.

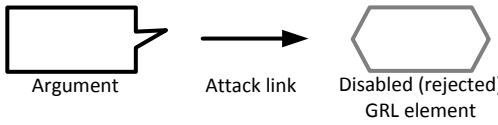

Fig. 5: The new elements and link of RationalGRL

the RationalGRL framework (cf. Figure 4). The goal model construction package consists of GRLModelElements, which can be either GRLLinkableElements or ElementLinks. A GRLLinkableElement can again be specialized into an Actor or an IntentionalElement (which is either a Softgoal, Goal, Task, Resource, or a Belief). Intentional elements can be part of an actor, and GRLLinkableElements are connected through ElementLinks of different types (i.e., Contribution, Decomposition, or Dependency). Finally, a GRLmodel is composed of GRLModelElements.

The practical reasoning and argumentation package depicts the concepts we introduced in Section 4.1. An ArgumentScheme represents a scheme containing variables. CriticalQuestions are possible ways to attack or elaborate an argument based on a scheme; each critical question applies to exactly one scheme, but for each scheme there may be more than one applicable critical question. When an argument scheme is instantiated, we obtain an Argument. Therefore, each argument is associated with exactly one scheme, but a scheme can be instantiated in multiple ways. When a critical question is answered, we may obtain an AttackLink, an Argument or both, depending on the answer. Note that it is also possible to an AttackLink can also be associated with no critical questions. This allows the user to create attacks between arguments, which do not necessarily correspond to one of the critical questions. A RationalGRLmodel is composed out of arguments and attack relations.

Notice that there are various OperationTypes in the Argumentation package. In RationalGRL, these operations are performed by instantiating an argument scheme or answering a critical question in a certain way. An INTRO operation introduces a new RationalGRL element. A DISABLE operation creates a new argument that attacks another argument or GRL element, effectively disabling it. The REPLACE operation replaces a RationalGRL element with a new element. Instantiating an argument scheme from Table 2 always leads to an INTRO operation, that is, it always introduces a new RationalGRL element. Answering a critical question can have different effects depending on the critical question and the answer. Table 2 shows these effects for the different critical questions and answers. For example, answering CQ0 with "no" disables the argument based on AS0.

There are two important links between the *practical reasoning & argumentation* and *goal model construction* packages. First, each GRLModelElement is an Argument. This means that each model element inherits the AcceptStatus as well, allowing GRL elements to be accepted or rejected. This, furthermore, means that argument schemes can be applied to all GRL elements, capturing the intuition that each GRL element can be regarded as an instantiated argument scheme. Note that besides arguments about elements of the GRL model, we also have a GenericArgument which is simply a counter-argument to an existing argument that does not relate to any of the GRL elements. Finally, the relation between GRLModelElement and Argument means that, as we already briefly indicated when discussing the framework in Figure 4, the class of RationalGRLmodel is a superclass of GRLmodel: besides arguments about GRL elements, we can also have arguments that does not relate to any of the GRL elements.

4.3 From Practical Reasoning to RationalGRL Models: examples from the case study

We now turn to the interactions between the *practical reasoning & argumentation* (i.e. the bottom left element of the framework in Figure 4) on the one hand, and *RationalGRL models* (i.e. the top left element of the framework in Figure 4) on the other hand. We provide informal examples of the links between the practical reasoning found in our case study transcripts and RationalGRL models. The formal grounding for the connection between practical reasoning and RationalGRL models can be found in the RationalGRL Metamodel (Section 4.2) and in more detail in the logical formalization of the Argument Schemes and Critical Questions (Section 6.3). The connection between the RationalGRL models shown in this section and regular GRL models is further formally defined in section 6.4.

Example 1 - Introducing GRL elements with arguments (INTRO) We start by showing how instantiating argument schemes leads to the introduction of new RationalGRL elements in a model. Take the example in Figure 7, which is based on the transcript excerpt shown in Table 5. On the left side, the arguments found in the transcript are shown, together with the argument scheme they are based on. The participants in the discussion argue that *System* has a goal and two tasks, and that the goal AND-decomposes into the two tasks. By arguing in this way, new GRL elements are introduced. These GRL elements are shown on the right side of Figure 7; the dashed arrows indicate the links between the practical reasoning and argumentation on the left and the RationalGRL model on the right.

Example 2: Disabling GRL elements by answering critical questions (DISABLE) The transcript excerpt of this example is shown in Table 3 in Appendix B and comes from transcript t_1. In this example, participants first sum up functionality of the traffic simulator, which

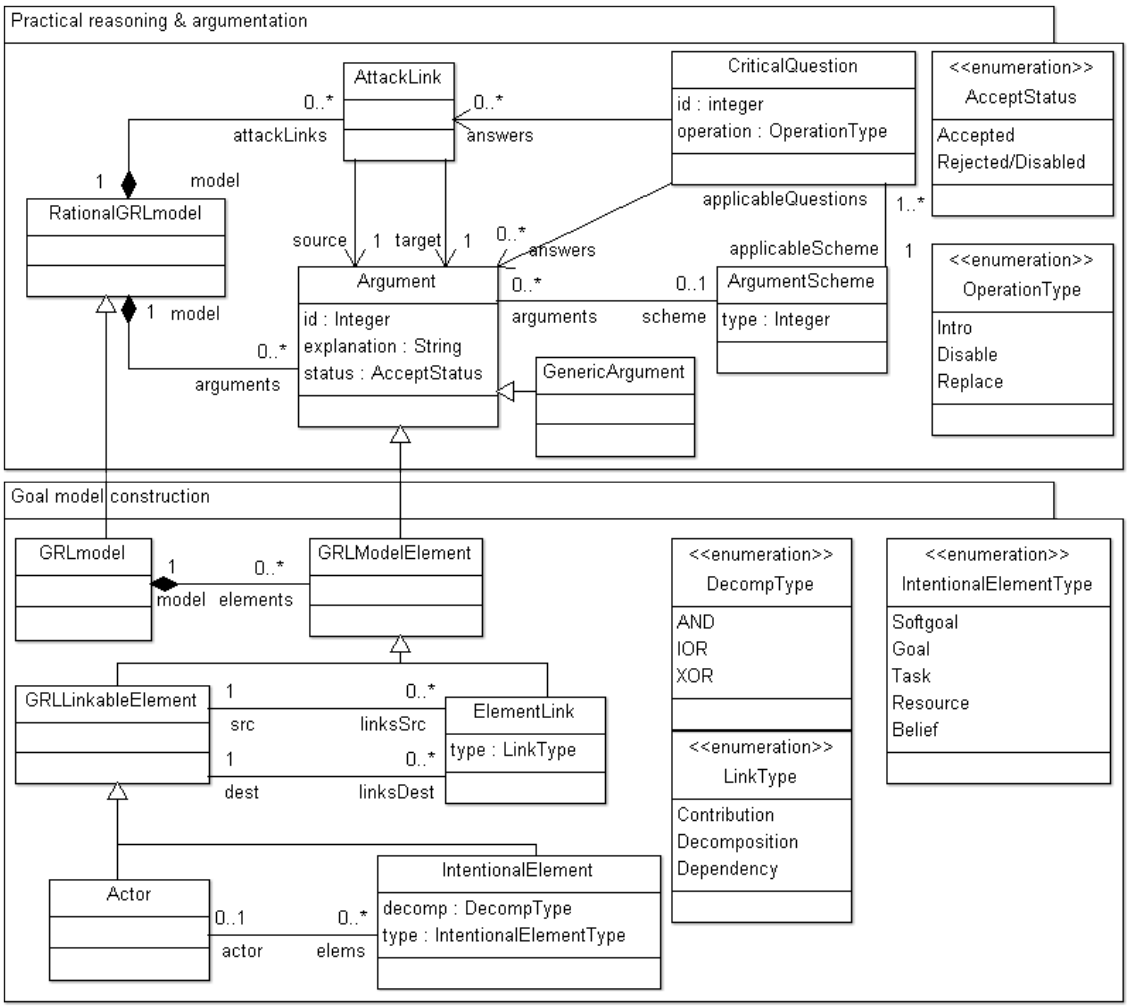

Fig. 6: The RationalGRL metamodel

can be captured as instantiations of AS2. On left side of Figure 8 one such instantiation is shown, which leads to the addition of the task *Add traffic light* in the RationalGRL model on the right side of Figure 8. However, participant P1 notes that the problem description states that all intersections have traffic lights by default, so the task *Add traffic light* is not necessary. This is captured using critical question CQ11. A negative answer to this question (cf. Table 2) should disable the original argument based on AS2 by attacking it. On the left side of Figure 8 a new argument (CQ11) attacks the original argument based on (AS2). This new argument is also added to the RationalGRL model on the right of Figure 8, where it attacks the original task *Add traffic light*. This attack leads to the original argument being *rejected* (cf. Section 2.3 and Section **??**), indicated by it being greyed out. As a result of this, the corresponding GRL task *Add traffic light* is also disabled.

Example 3: Changing a decomposition type by answering critical questions (REPLACE) The transcript excerpt of this example is shown in Table 5 in the appendix and comes from transcript t_3. It consists of a discussion about the type of decomposition relationship for the goal *Simulate Traffic*. Recall that in Example 1, an AND-decomposition was introduced for this goal with AS5 (Figure 7). In the discussion CQ10b – "Is the decomposition type correct?" – is explicitly asked. The answer is "No, it should be OR". The original argument for AND-decomposition is now attacked by the argu-

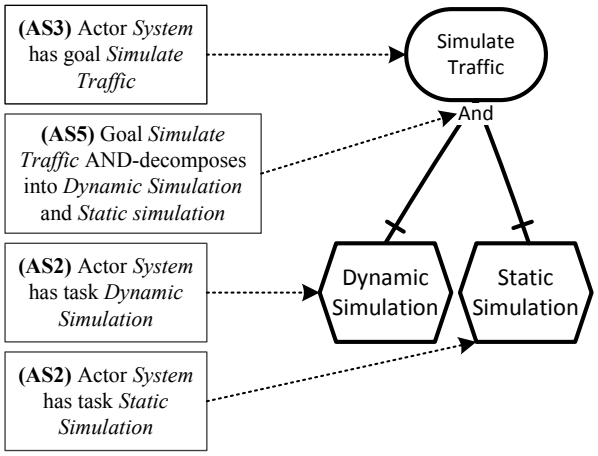

Fig. 7: Introducing new GRL elements with argumentation (INTRO).

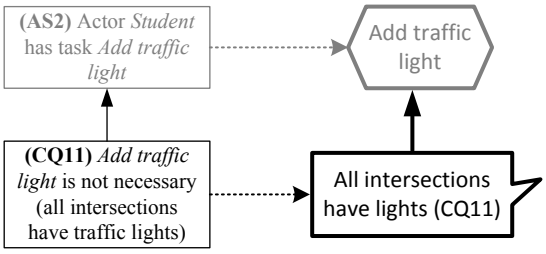

Fig. 8: Disabling GRL tasks with argumentation (DISABLE).

ment for the OR-decomposition, and the new argument is linked to the OR-decomposition in the RationalGRL model (Figure 9).

Example 4: Clarifying a task by answering critical questions (REPLACE) The transcript excerpt of this example is shown in Table 4 in Appendix B and comes from transcript t_1. The discussion starts with an instantiation of argument scheme AS2: "Actor *Student* has task *Set car influx*". This argument is then challenged with critical question CQ12: "Is the task *Set car influx* specific enough?". This is answered negatively, creating a new argument "Actor *Student* has task *Set car influx per road*", which attacks the original argument for *Set car influx*. Note how the new task *Set car influx per road* also attacks (and disables) the original RationalGRL task *Set car influx* (Figure 10).

The discussion about the task then continues. Again, there is the question (CQ12) whether the new task *Set car influx per road* is clear enough. This time the answer is positive, and participant P1 also gives a reason for this positive answer: setting car influx per road was in the original requirements (cf. requirement 4 in Appendix A). This reason is then introduced as a belief in the RationalGRL diagram, with a correlation link to the task it is about. Thus, beliefs can be used as they are intended, namely as rationales for certain choices.

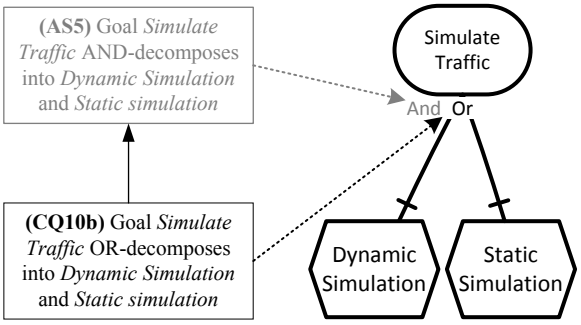

Fig. 9: Replacing a GRL decomposition with argumentation (REPLACE).

Example 5: Defending the addition of an actor (DISABLE) The transcript excerpt of this example is shown in Table 6 in the appendix and comes from transcript t_3. First participant *P1* puts forth the suggestion to include actor *Development team* in the model. This is, then, questioned by participant *P2*, who argues that the professor will develop the software, so there will not be any development team. This is shown in Figure 11: first the actor *Development Team* is introduced with an argument based on AS0. This argument is then attacked by answering critical question CQ0.

Further in the discussion, participant *P2* argues that the development team should be considered, since the professor does not develop the software. This is captured using a generic counterargument (*Att* in Table 2), which attacks the earlier argument based on CQ0. Figure 12

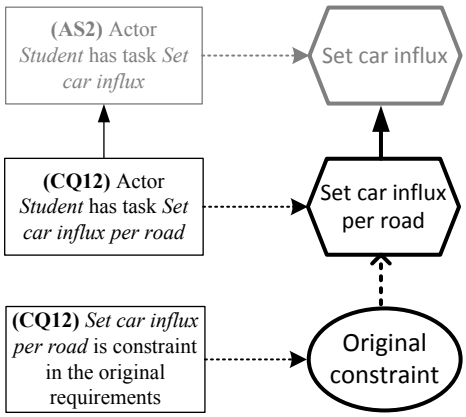

Fig. 10: Renaming a GRL task with argumentation (REPLACE).

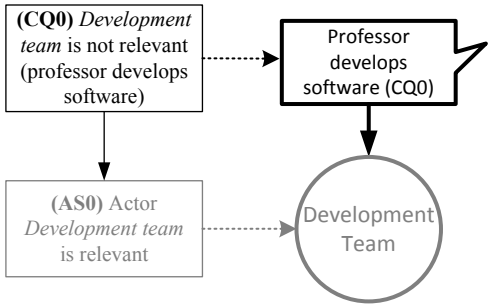

Fig. 11: Disabling a GRL actor with argumentation (DISABLE).

shows the situation after the counterargument has been put forward: the argument (Att) now attacks the argument (CQ0), which in turn attacks the original argument (AS0). As a result, the argument (AS0) is acceptable (cf. Section 2.3 and Section **??**), which causes the actor in the RationalGRL model to be enabled again.

5 The RationalGRL Methodology and Tool Support

In the previous sections, we have shown how the RationalGRL framework can capture stakeholder discussions, and how interactions between two types of reasoning, practical reasoning and goal modeling, leads to two interlinked models, RationalGRL and GRL models. In this section we clarify how practitioners can actually use the RationalGRL framework by proposing a methodology[5]

[5] The methodology presented here was first presented at the 2017 iStar workshop [**?**].

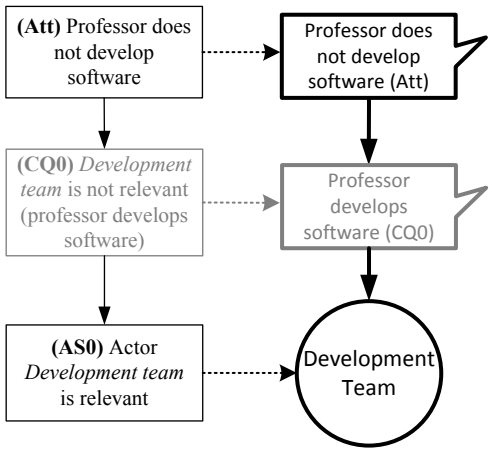

Fig. 12: Defending a GRL actor by attacking its disabling attacker (DISABLE).

(**requirement 4**) and discussing a prototype RationalGRL tool (**requirement 5**).

5.1 RationalGRL Methodology

We propose the methodology shown in Figure 13 to develop a (Rational)GRL model.

(1) Instantiate Argument Schemes (AS) – First, we start with our list of argument schemes (Table 2). Whilst discussing the requirements, we then select schemes from the list and instantiate them to form arguments for GRL model elements. In this way we build or modify the GRL model by introducing new GRL elements (INTRO, see example 1 in Section 4.3). Note that it is also possible to start modifying an existing GRL model which was not built using the RationalGRL methodology: each GRL element corresponds to an argument (i.e. an instantiated scheme), so it is possible to instantiate argument schemes based on an existing GRL model.

(2) Answer Critical Questions (CQs) – After building or modifying the initial GRL model, we can start asking the relevant critical questions. Because each element in the GRL model corresponds to an instantiated scheme, we can look at Table 2) to see which questions are relevant given our GRL model.

(3) Decide on Intentional Elements and their Relationships – By answering a critical question, one of three operations are performed on the GRL model: INTRO, DISABLE or REPLACE. Any of these operations impact the arguments and corresponding GRL intentional elements, modifying the initial GRL model into a RationalGRL model (see examples in Section 4.3). After these modifications, we can keep on asking critical questions (e.g. about elements that were introduced by previously answering a critical question) until we are satisfied with our model.

(4) Modify GRL Models – In this step, we modify the regular GRL model based on the RationalGRL model of step (3). That is, one of the following situation can happen with respect to the initial GRL model: 1) a new intentional element or a new link is introduced; 2) an existing intentional element or an existing link gets disabled (removed) from the model; or 3) an existing intentional element or link is replaced by a new one. This results in a new, modified GRL model, which can be used as the basis for another cycle of the methodology.

We can continue these four steps until there is no more intentional element or link to analyze or we reach a satisfactory model. In the next section, we will give an example of how our tool can be used together with the methodology to build a GRL model.

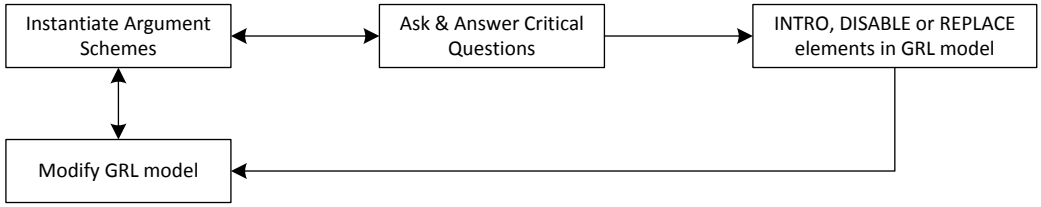

Fig. 13: The RationalGRL Methodology

5.2 The RationalGRL Tool

TODO for Marc(by Marc): Currently this section comes from my thesis where I present the tool as future work. Here we should explain it

GRL has a well-documented and well-maintained tool called jUCMNav [27]. This tool is an extension to Eclipse. Although it is a rich tool with many features, we also believe it is not very easy to set it up. This seriously harms the exposure of the language, as well as the ability for practitioners to use it. We have started to implement a simple version of GRL as an online tool in Javascript. This makes it usable from the browser, without requiring the installation of any tool. The tool can be used from the following address:

```
http://marcvanzee.nl/
RationalGRL/editor
```

A screenshot of the tool is shown in Figure 14. As shown, there are two tabs in the tool, one for "Argument" and one for "GRL". The argument part has not been implemented yet, and the GRL part only partly, but the idea behind the tool should be clear. Users are able to work on argumentation and on goal modeling in parallel, where the argumentation consists of forming arguments and counterarguments by instantiating argument schemes and answering critical questions.

An important aspect of the tool is that users can switch freely between these two ways of modeling the problem. One can model the entire problem in GRL, or one can do everything using argumentation. However, we believe the most powerful way to do so is to switch back and forth. For instance, one can create a simple goal model in GRL, and then turn to the argumentation part, which the users can look at the various critical questions for the elements, which may trigger discussions. These discussions results in new arguments for and against the elements in the goal model. Once this process is completed, one may switch to the goal model again, and so on. We believe that in this way, there is a close and natural coupling between modeling the goals of an organization as well as rationalizing them with arguments.

6 RationalGRL: Logical Framework

In Section 3 we developed a list of critical questions and argument schemes by analyzing transcripts of discussions about the development of a traffic simulator. The resulting list is shown in Table 2. We then presented the RationalGRL framework in secton 4, consisting of a modeling language, a metamodel, and various examples of using the argument schemes and critical questions. In section 5 we explained how this framework can be used by practitioners by explaining the methodology and tool support.

In this section we present a formalization of RationalGRL based on formal logic. This is done for multiple reasons: (i) Most approaches in formal argumentation use formal logic, allowing us to employ existing technique directly in order to compute which arguments are accepted and which are rejected, (ii) we can be more precise about how critical questions are answered, (iii) we can show that RationalGRL models can be translated in to valid GRL models in a very precise way, and (iv) the formal approach is a basis for automating the framework in terms of tool support.

In the first two subsections we formalize a static representation of our framework: We provide a formal specification of a GRL model based on the GRL metamodel (Section 6.1) in the first subsection, and we extend this with arguments and attack links in the second subsection, hereby obtaining a formal specification of a RationalGRL model. We then provide a formal translation procedure from a RationalGRL model to a GRL model.

In the third subsection we turn to the dynamics of our framework. We develop algorithms for instantiating argument schemes and for answering critical questions.

The two main results of this section are that (i) the translation procedure from a RationalGRL model to a GRL model is correct, meaning that if the RationalGRL model is valid, the resulting GRL model is also valid, (ii) the algorithms for applying argument schemes and critical questions are correct as well, meaning that if the Rational model is valid before, it is also valid afterwards. These two results together entail that our framework and the corresponding operations on it are consistent.

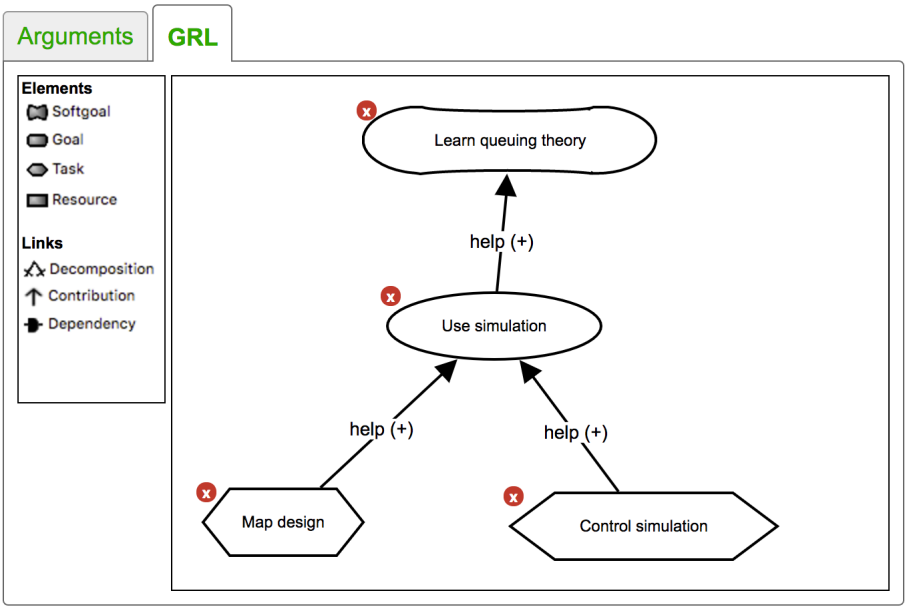

Fig. 14: Screenshot of the prototype tool

6.1 Formal Specification of GRL

In this subsection we formalize a GRL model based on the GRL metamodel and the jUCMNav implementation. We first formalize elements of GRL (intentional elements and actors), and then formalize the links.

Intentional Element and Actors

We start with some general definition that we use in subsequent definitions.

Definition 1 (General definitions). *Throughout this section, we adopt the convention that variables start with a lowercase letter (e.g, $id, i, j, name, ie, goal$, and sets and constants start with an uppercase letter (e.g., $Type, AND, Goal$).*

We define the following sets:

- $Type = \{Softgoal, Goal, Task, Resource\}$,
- $Names$ *is a finite set of string.*
- $DecompType = \{AND, OR, XOR\}$.
- $ContribValue = \{Break, Hurt, Some\ negative, Unknown, Some\ positive, Help, Make\}$,

Next we define an intentional element.

Definition 2 (Intentional Element). *An intentional element $ie \in \mathbb{N} \times Type \times Names \times DecompType$ is a relation, where $ie = (id, type, name, decomptype)$ means:*

- $id \in \mathbb{N}$ *is a unique identifier for the element,*
- $type \in Type$ *specifies the type of the element,*
- $name \in Names$ *is a string description of the element,*
- $decomptype \in DecompType$ *refers to the type of decomposition.*[6]

A set of intentional elements is denoted by IE.

Example TODO for Marc(by Marc): Give example here

The definition above is sufficient to capture all intentional elements used in GRL. However, we present some syntactic sugar in the next definition by abbreviating the definition above in various ways. This does not add anything new to the previous definition, but it simplifies some of the notation.

Definition 3 (Notation). *We adopt three conventions simplifying our notation:*

- *We refer to the element of an intention element ie using the dot (".") notation. That is, we may refer to the id, type, name and decomposition type with respectively $ie.id$, $ie.type$, $ie.name$, and $ie.decomptype$.*

[6] Note that the decomposition type is relevant only if the element is in fact decomposed into other elements, but since the jUCMNav implementation defines the decomposition type on the element, we do the same here.

– *We refer to an intentional element with $ie.id = i$ simply by ie_i.*[7] *For instance, we may refer to the intentional element $ie = (0, Goal, \text{Make profit}, AND)$ with ie_0 and write $ie_0.type = Goal$, $ie_0.name =$ Make profit, and $ie_0.decompositiontype = AND$.*
– *We can also refer to intention elements of a specific type simply by $type_{id}$. For instance, we can abbreviate the element in the previous item with $goal_0$ and write $goal_0.name =$ Make profit, and $goal_0.decomptype = AND$.*

Definition 4 (Actor). *An actor $act \in \mathbb{N} \times Names$ is a relation where $act = (id, name)$ means:*

– *$id \in \mathbb{N}$ refers to the unique identifier of the actor,*
– *$name$ refers to a string description of its name.*

Similar to intentional elements, we may refer to $act = (id, name)$ with act_{id} and write $act_{id}.name$ to refer to its name.

A set of actors is denoted by Act.

In the next definition we specify relations between actors and intentional elements. Each actor can own one or multiple intentional elements. We chose to define this simply as a actor-element relation, which is in line with the jUCMNav implementation.

Definition 5 (Actor-IE Relations). *An* Actor-IE relation *$r_{ActIE} \in \mathbb{N} \times \mathbb{N}$ is a relation (i, j) meaning that an actor with id i has intentional element with id j.*

A set of Actor-IE relations is denoted by R_{ActIE}.

Links

At this point we have defined all intentional elements in GRL and a containment relation between actors and intentional elements. We now turn to the GRL links.

Definition 6 (Contribution Link). *A* contribution *link: $contrib \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times Contribvalue$ is a relation such that $contrib = (i, j, k, value)$ means*

– *i is the unique identifier of the link,*
– *j is the unique identifier of the IE from which the contribution originates,*
– *k is the unique identifier of the IE to which is contributed.*

Intuitively, $contrib = (i, j, k, value)$ means that ie_i contributes to ie_j with $value$.

A set of contribution links is denoted by $Contrib$.

[7] Note that the reference ie_i is never ambiguous in GRL since identifiers for intentional elements are always unique (see Definition **??**).

Definition 7 (Decomposition Link). *A* decomposition *link: $decomp \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ is a relation such that $decomp = (i, j, k)$ means*

– *i is the unique identifier of the link,*
– *j is the unique identifier of the decomposing IE,*
– *k is the unique identifier of the IE that is being decomposed.*

Intuitively, $decomp = (i, j, k)$ means that ie_i decomposes ie_j.[8]

A set of decomposition links is denoted by $Decomp$.

Definition 8 (Dependency Link). *A* dependency *link: $dep \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ is a relation such that $dep = (i, j, k)$ means*

– *i is the unique identifier of the link,*
– *j is the unique identifier of the dependee IE,*
– *k is the unique identifier of the dependent IE.*

Intuitively, $dep = (i, j, k)$ means that ie_j depends on ie_j. **TODO for Sepideh(by Marc): Is this correct? For a dependency relation $A \rightarrow B$, who is dependent on who?**

A set of dependency links is denoted by Dep.

Similar to intentional elements and actors, we may refer to links with an id using the subscript notation, e.g., $contrib_0$ is a contribution link with $contrib.id = 0$.

The definitions above come with various implicit assumptions in order to form a valid GRL model. For instance, if an Actor-IE relation (i, j) occurs in a GRL model, then there should exist an actor with id i, and there also should exist an IE with id j. We make these assumptions explicit in the following definition.

Definition 9 (GRL Model). *A* GRL model *$GRL = (IE, Act, R_{ActIE}, Contr, Decomp, Dep)$ consists of:*

– *A set IE of intentional elements (Def. 2),*
– *A set Act of actors (Def. 4),*
– *A set R_{ActIE} of Actor-IE relations (Def. 5),*
– *A set $Contr$ of contribution links (Def. 6),*
– *A set $Decomp$ of decomposition links (Def. 7),*
– *A set Dep of dependency links (Def. 8).*

Furthermore, the following conditions are satisfied:

1. *ids are globally unique across IEs, Links, and Actors, i.e., let $X, Y \in \{IE, Act, Contr, Decomp, Dep\}$. For all X_i and Y_j: if $i = j$ then $X = Y$ and $X_i = Y_j$.*
2. *All intentional elements of actors exist: $\forall (i, j) \in R_{ActIE} : act_i \in Act \wedge ie_j \in IE$.*

[8] Note that the decomposition type is defined on IE_k, see Definition 2.

3. *An intentional element belongs at most to one actor:* $\forall ie_i \in IE : |\{(k,i) \in R_{ActIE}\}| \leq 1$.
4. *Contribution links connect intentional elements:* $\forall (i,j,k,value) \in Contrib : \{ie_j, ie_k\} \subseteq IE$.
5. *Decomposition links connect intentional elements:* $\forall (i,j,k) \in Decomp : \{ie_j, ie_k\} \subseteq IE$.
6. *Dependency links connect intentional elements:* $\forall (i,j,) \in Dep : \{ie_j, ie_k\} \subseteq IE$.

TODO for Marc(by Marc): Provide explanation and example here.

6.2 Formal specification of RationalGRL

In order to develop a logical framework for RationalGRL, we extend the GRL logical framework of the previous section by adding three elements (see Figure 5):

- A new element called *generic argument*,
- A new link called *attack link*,
- An labeling of the elements, assigning to each element (an IE, link, or generic argument) the label IN (accepted) or OUT (reject).

In the next section we explain how the labeling is computed, but in this section we only provide a formal definition.

We start with the new element which we call the *Generic Argument*.

Definition 10 (Generic Argument). *A generic argument* $arg \in \mathbb{N} \times Names$ *is a relation such that* $arg = (id, name)$ *is an argument with unique identifier* id *and name* $name$*. We may refer to the argument with id* i *simply with* arg_i*. A set of arguments is denoted by* Arg*.*

Note that one of the constraints of a GRL model (Def. 9) is that GRL links (Def. 6, 7, and 8) should connect IEs, which means that in GRL generic arguments cannot be connected with GRL links. This is correct, since generic arguments are not part of GRL, so they should also not occur in links.

We next define the attack link, which is the only link that RationalGRL adds to GRL.

Definition 11 (Attack Link). *An attack link* $att \in \mathbb{N} \times \mathbb{N}$ *is a relation such that* $att = (i,j)$ *means:*

- i *is the unique identifier of the element that is performing that attack,*
- j *is the unique identifier of the element that is being attacked.*

Intuitively, $att = (i,j)$ *means that element* i *is attacking element* j*.*

Note that the previous definition did not put any constraint on which elements can be connected with attack links. This is what we make explicit in the definition of a RationalGRL model later.

We next add the last part of the RationalGRL addition, namely the labeling of elements. This labeling determines whether an element is accepted (which means it shows up in the GRL model if it is an IE or a link), or rejected (which means it won't be shown).

Definition 12 (RationalGRL labeling). *A RationalGRL labeling function* $Lab : \mathcal{N} \rightarrow in, out$ *that is total on the ids of the RationalGRL elements. This means that* $Lab(id)$ *is defined for each* $ie.id, contr.id, decomp.id, dep.id, arg.id$*.*

Definition 13 (RationalGRL Model). *A* RationalGRL model $RationalGRL = (IE, Act, R_{ActIE}, Contr, Decomp, Dep, Arg, Att, Label)$ *consists of:*

- *A GRL model* $(IE, Act, R_{ActIE}, Contr, Decomp, Dep)$
- *A set of generic arguments* Arg *(Def. 10),*
- *A set of attack links (Def. 11).*
- *A labeling function* Lab *(Def. 12).*

Furthermore, the following conditions are satisfied. In the following, let $Elems = \{IE, Act, Contr, Decomp, Dep, Arg\}$*:*

1. *All conditions on a GRL model are satisfied, (Def. 9),*
2. *ids are globally unique across IEs, Links, Actors, and Generic arguments, i.e., let* $X, Y \in Elems$*. For all* X_i *and* Y_j*: if* $i = j$ *then* $X = Y$ *and* $X_i = Y_j$*.*
3. *Attack links connect generic arguments, IEs, actors, and links with each other, i.e., let* $\forall (i,j) \in Att : \exists X \in Elems$ *s.t.* $X_i \in X$ *and* $\exists Y \in Elems$ *s.t.* $Y_i \in Y$*.*

TODO for Marc(by Marc): Explain conditions here and give some examples

Argumentation semantics

In order to determine which arguments are accepted or not, we define *argumentation semantics*. We used the standard approach here, which is known as *Dung's semantics*. The following notions are preliminary.

Definition 14 (Attack, conflict-freeness, defense, and admissibility [12]). *Suppose an argumentation framework* $AF = (Arg, Att)$*, two sets of arguments* $S \cup S' \subseteq Arg$*, and some argument* $A \in Arg$*. We say that*

- S attacks A *if some argument in* S *attacks* A*,*

– S attacks S' *if some argument in* S *attacks some argument in* S',
– S *is* conflict-free *if it does not attack itself,*
– S defends A *if for each* B *such that* B *attacks* A, S *attacks* B,
– S *is* admissible *if* S *is conflict-free and defends each argument in it.*

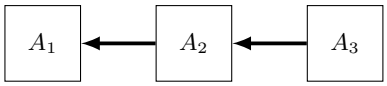

Fig. 15: Example argumentation framework.

Let us explain these definitions using the example argumentation framework in Figure 15, where A_2 attacks A_1 and A_3 attacks A_2. There are two admissible sets: $\{A_3\}$ and $\{A_1, A_3\}$, in which A_3 defends A_1 against its attacker A_2. Sets containing both A_2 and either A_1 or A_3 are not conflict free, and the sets $\{A_1\}$ and $\{A_2\}$ do not defend themselves against A_2 and A_3, respectively.

Given the notion of admissible sets, we can then define our argumentation semantics. There are a large number of different semantics to determine which arguments are acceptable; in this article, we choose preferred semantics.

Definition 15 (Preferred semantics [12]). *A preferred extension of an argumentation framework* AF *is a maximal (w.r.t. set inclusion) admissible set of* AF.

In our example from Figure 15, there is one preferred extension, namely $\{A_1, A_3\}$.

TODO for Marc(by Marc): Give example with concrete RationalGRL model

6.3 Algorithms for argument schemes

In the previous section we formalize a *static* representation of the RationalGRL framework. In this section we formalize the *dynamics*. We do so by developing algorithms for applying argument schemes and critical questions in the context of a RationalGRL model 13. These algorithms produce new elements and attack relations. We can then use argumentation semantics (Definition 15) to compute sets of accepted arguments. The content of these arguments is then used to compute the resulting RationalGRL model, together with enabled and disabled GRL elements and their underlying arguments.

The algorithms for the argument schemes consist of forming a new argument and adding it to the set of arguments. No attack relations are introduced.

Algorithm 1 Applying AS0: Actor a is relevant

1: $RationalGRL = (IE, Act, R_{ActIE}, Contr, Decomp,$
2: $\quad Dep, Arg, Att, Label)$
3: **procedure** $AS_0(a)$
4: $\quad id \leftarrow mintId()$
5: $\quad Act \leftarrow Act \cup (id, a)$
6: **end procedure**

Algortihm 1 for argument scheme AS0

Explanation For all of the argument schemes and critical questions algorithms, we assume an underlying RationalGRL model as defined in Def. 13. This model is shown in the first two lines.

The algorithm takes one argument, namely the name of the actor a. On line 4 of the algorithm, a new (unique) id is minted. On line 5, the the tuple (id, a) is added to the set of actors Act, meaning that a new actor with id id and name a is added to the RationalGRL model (Def. 4). In Figure 12, the application of the argument scheme AS_0(Development team), results in one argument:

$$Args = \{\{actor(0), name(0, \text{Development team})\}\}.$$

Algorithm 2 Applying AS1: Actor a_{id} has resource n

1: **procedure** $AS_1(a_{id}, n)$
2: $\quad id \leftarrow id + 1$
3: $\quad A \leftarrow \{resource(id), name(id, n), has(a_{id}, id)\}$
4: $\quad Args \leftarrow Args \cup A$
5: **end procedure**

Example *Algorithm 2 for argument scheme AS1:* This argument scheme takes two arguments, the identifier a_{id} of the actor and the resource name n. The algorithm is similar to the previous one, with the difference that the newly added argument contains the statement $has(a_{id}, id)$ as well, meaning that the actor with id a_{id} has element id (which is a resource). As an example, let us formalize the argument corresponding to resource *External library* of actor *Traffic tycoon* in Figure **??**. First, we assume some id is associated with the actor:

$$\{actor(0), name(0, traffic\_tycoon)\}.$$

Then we can formalize the argument for the resource as follows:

$$\{resource(1), name(1, external\_library), has(0, 1)\}.$$

Proof of Correctness In order to show that this algorithm is correct, we should show that the new Rational-GRL model still satisfies all the conditions of Def. 9. This is clearly the case, since adding a single element does not invalidate any condition.

Argument scheme AS1 to AS4 are all very similar, in the sense that they all assert that some element belongs to an actor. Therefore, we only provide the algorithm for AS1 and we assume the reader can easily construct the remaining algorithms AS2-AS4.

Algorithm 3 Applying AS5: Goal g_{id} decomposes into tasks $T_1, \ldots, T_n$

```
1: procedure AS5(g_id, {T_1, ..., T_n}, type)
2:     T_id = ∅
3:     for T_i in {T_1, ..., T_n} do
4:         if ∃_{A∈Args}{task(t_id), name(t_id, T_i)} ⊆ A then
5:             T_id ← T_id ∪ {t_id}
6:         else
7:             id ← id + 1
8:             A ← {task(id), name(id, T_i)}
9:             Args ← Args ∪ A
10:            T_id ← T_id ∪ {id}
11:        end if
12:    end for
13:    id ← id + 1
14:    A ← {decomp(id, g_id, T_id, type)}
15:    Args ← Args ∪ A
16: end procedure
```

Algorithm 3 for argument scheme AS5

Explanation The procedure in Algorithm 3 takes three arguments: g_{id} is the identifier of goal G, $T = (T_1, \ldots, T_n)$ is a list of decomposing task names, and $type \in \{and, or, xor\}$ is the decomposition type. The difficulty of this algorithm is that each of the tasks are stated in natural language, and it is not directly clear whether these tasks are already in the GRL model or not. Therefore, we have to check for each tasks whether it already exists, and if not, we have to create a new task. On line 2, the set T_{id} is initialized, which will contain the ids of the tasks $T_1, \ldots, T_n$ to decompose into. In the for-loop, the if-statement on line 4 checks whether some argument already exists for the current task T_i, and if so, it simply adds the identifier of the task (t_{id}) to the set of task identifiers T_{id}.[9] Otherwise (line 6), a new task is created and the new identifier id is added to the set of task identifiers. After the for loop on line 13, an argument for the decomposition link itself is constructed, and it is added to the set of arguments $Args$.

[9] Note that the existing task may be disabled, but this does not matter, since the decomposition relation will be suppressed for disabled elements.

Example Let us explain this algorithm with the XOR-decomposition of goal `Generate cars` of Figure **??**. Suppose the following arguments are constructed already:

- $\{goal(0), name(0, generate\_cars)\}$,
- $\{taks(1), name(1, keep\_same\_cars\}$.

Suppose furthermore that someone wants to put forward the argument that goal `Generate cars` XOR-decomposes into tasks `Keep same cars` and `Create news cars`. Formally: $AS_5(0, \{generate\_cars, keep\_same\_cars\}, xor)$.
The algorithm will first set $T_{id} = \emptyset$, and then iterate over the two task names. For the first task $generate\_cars$, there does not exist an argument $\{task(t_{id}), name(t_{id}, generate\_cars)\}$ yet, so a new argument is created. Suppose the following argument is created: $\{task(2), name(2, generate\_cars)\}$. After this, 2 is added to T_{id}. For the second task an argument exists already, namely $\{task(1), name(1, keep\_same\_cars)\}$, so 1 is simply added to T_{id}. After the for loop, we have $T_{id} = \{1, 2\}$. Next, the decomposition argument is created, which is $\{decomp(3, 0, \{1, 2\}, xor)\}$. This argument is added to $Args$ and the algorithm terminates.

Algorithm 4 Applying AS6: Task t_{id} contributes to softgoal s

```
1: procedure AS6(t_id, s)
2:     if ∃_{A∈Args}{softgoal(i), name(i, s)} ⊆ A then
3:         s_id ← i
4:     else
5:         id ← id + 1
6:         A ← {softgoal(id), name(id, t)}
7:         Args ← Args ∪ A
8:         s_id ← id
9:     end if
10:    id ← id + 1
11:    A ← {contr(id, t_id, s_id, pos)}
12:    Args ← Args ∪ A
13: end procedure
```

Proof of Correctness

Algorithm 4 for argument scheme AS6

Explanation The procedure in Algorithm 4 takes two arguments: t_{id} is the identifier of task T, and s is the soft-

goal name that is contributed to. The idea behind this algorithm is very similar to the previous one, with the difference that in the current algorithm we create a single relation, while we created a set of relations in the previous algorithm. First, the if-statement on line 2 checks whether the softgoal exists already, and if not, an argument is added for it. This ensures that all softgoals have corresponding arguments. After the if-statement, the argument for the contribution link is created and it is added to the set of arguments $Args$.

Example Let us again illustrate this with a simple example from Figure **??**. Suppose the following argument exists already: $\{task(0), name(0, keep\_same\_cars\}$, and suppose someone would like to add an argument that the task `Keep same cars` contributes positively to softgoal `Dynamic simulation`, i.e. $AS_6(0, dynamic\_simulation)$. The algorithm first checks whether an argument already exists for the softgoal, and when it finds out it does not exist, creates the argument $\{softgoal(1), name(1, dynamic\_simulation)\}$ and adds it to $Args$. Then, the argument for the contribution is added to $Args$ as well, which is $\{contr(2, 0, 1, pos)\}$.

Algorithms for argument schemes AS7-AS11: The algorithms for $AS7$ to $AS11$ all have a very similar structure as Algorithm 4 and have therefore been omitted. Again, we assume the reader can reconstruct them straightforwardly.

Proof of correctness

Algorithms for critical questions

We now develop algorithms for our critical questions. Recall that answering a critical question can have four effects, and we discuss each of these effects separately.

Algorithm 5 Applying DISABLE: Element i is disabled

```
1: procedure DISABLE(i)
2:     id ← id + 1
3:     A ← {disabled(i))}
4:     Args ← Args ∪ A
5: end procedure
```

Algorithm 5 (DISABLE) for critical questions CQ0-CQ5a, CQ6a, CQ7a, CQ8, CQ11, and CQ12: The disable operation consists of adding an argument stating the GRL element with identifier i is disabled. Let us reconsider the example of Figure 11. This example consists of an instantiation of argument scheme AS0, which is attacked by an argument that resulted from answering critical question. The instantiation of AS0 leads to the argument $A = \{actor(0), name(0, dev\_team\}$. After applying $DISABLE(0)$ we obtain the arguments:

$$Args = \{\{actor(0), name(0, dev\_team\}, \{disabled(0)\}\}.$$

Note that our implementation of this attack does not lead to an actual attack in the argumentation framework.

Algorithm 6 Answering CQ5b: "Does goal G decompose into any other tasks?" With: "Yes, namey into tasks $t_1, \ldots, t_k$"

```
1: procedure CQ5B(g_id, {i_1, ..., i_n}, type, {t_1, ..., t_k})
2:     T_id = {i_1, ..., i_n}
3:     for t_i in {t_1, ..., t_k} do
4:         if ∃_{A∈Args}{task(t_id), name(t_id, t_i)} ⊆ A then
5:             T_id ← T_id ∪ {t_id}
6:         else
7:             id ← id + 1
8:             A ← {task(id), name(id, t_i)}
9:             Args ← Args ∪ A
10:            T_id ← T_id ∪ {id}
11:        end if
12:    end for
13:    id ← id + 1
14:    A_new = {decomp(id, g_id, T_id, type)}
15:    for A in {decomp(_, g_id, _, _)} ⊆ A | A ∈ Args} do
16:        Att ← Att ∪ {(A_new, A)}
17:    end for
18:    Args ← Args ∪ {A_new}
19: end procedure
```

Algorithm 6 (REPLACE) for critical questions CQ5b: This algorithm is executed when critical question CQ5b is answered, which is a critical question for argument scheme AS5. Therefore, it assumes an argument for a goal decomposition already exists of the following form (see Algorithm 3):

$$\{decomp(d, g_{id}, \{i_1, \ldots, i_n\}, type).$$

The goal of the algorithm is to generate a new argument of the form $decomp(d, g_{id}, \{i_1, \ldots, i_5\} \cup \{j_1, \ldots, j_k\}, type)$, where $\{j_1, \ldots, j_k\}$ are the identifiers of the additional decomposing tasks $\{t_1, \ldots, t_k\}$.

The algorithm takes as input the goal identifier g_{id}, the set of existing decomposing task identifiers $i_1, \ldots, i_n$, the decomposition type, and the names of the new tasks $t_1, \ldots, t_k$ that should be added to the decomposition. The first part of the algorithm is familiar from Algorithm 3: For each task name we check whether it already exists as an argument (line 4), and if it doesn't (line 6) we add a new argument for it. After the for-loop (line 13), a

new argument is created for the new decomposition relation (14). After this, the for-loop on line 15 ensures that the new argument attacks all previous arguments for this decomposition (note that the variable "_" means "do not care"). Only at the very end the new argument is added (line 18), to ensure it does not attack itself after the for loop of line 15-17.

An example of this algorithm is shown in Figure 16.[10] Before the critical question is applied, the following arguments have been put forward:

- $\{goal(0), name(0, show\_simulation)\}$
- $\{task(1), name(1, generate\_traffic)\}$
- $\{task(2), name(2, compute\_lights)\}$
- $\{decomp(3, 0, \{1, 2\}, and)\}$.

Next, Algorithm 6 is called as follows: $CQ5b(0, \{1, 2\}, and, \{show\_controls\})$. That is, the existing decomposition is challenged by stating that goal $show\_simulation$ not only decomposes into $generate\_traffic$ and $compute\_lights$, but it also decomposes into $show\_controls$. Since this task does not exist yet, it is created by the algorithm, which also ensures the new argument for the decomposition link attacks the previous argument for the decomposition link.

Algorithms for critical questions CQ10a and CQ10b (REPLACE): These algorithms have a very similar structure as Algorithm 6 and have therefore been omitted.

Algorithm 7 Answering CQ13: "Is the name of element i clear?" With: "No, it should be n"

```
1: procedure CQ13(i, n)
2:     ArgsN ← {A ∈ Args | name(i, x) ∈ A}
3:     B ← B'\{name(i, _)} with B' ∈ ArgsN
4:     A ← B ∪ {name(i, n)}
5:     Args ← Args ∪ {A}
6:     for C in ArgsN do
7:         Att ← Att ∪ {(A, C)}
8:     end for
9: end procedure
```

Algorithms for critical question CQ13 (REPLACE): This algorithm is used to clarify/change the name of an element. It takes two parameters: the element identifier i and the new name n. The idea behind the algorithm is that we construct a new argument for n, and to ensure that this argument attacks all previous arguments for giving a name to this element. Here we can use the following fact: Suppose $Args$ is a set of arguments, each containing an atom about the name for an element i, i.e. for all $A \in Args : name(i,) \in Args$, then all arguments in $Args$ are the same except for the $name$ atoms, i.e. for all $A, B \in Args : A \backslash \{name(i,)\} = B \backslash \{name(i,)\}$. This means that if we would like to attack every argument of $Args$ with a new argument that replaces the $name$ atom, we can simply take any argument in $Args$, remove the previous $name$ atom, add the new one and attack all arguments in $Args$. This is exactly what the algorithm does.

On line 2, all arguments that have been put forward for this element and contain $name(i, x)$ are collected into the set $ArgsN$. On line 3, some arguments $B' \in ArgsN$ minus the $name$ statement is assigned to B (note that it does not matter which one we pick), and on line 4 B is joined with the new $name$ statement and stored in A, which is then added to the set of arguments $Args$. The for-loop on lines 6-8 ensures all previous arguments for names of the element are attacked by the new argument.

An example of of Algorithm 7 is shown in Figure 17. Let us consider the last application of CQ13 (bottom argument). Before this application, the following arguments have been put forward:

- A_1: $\{actor(0), name(0, student)\}$
- A_2:$\{task(1), name(1, create\_road), has(0, 1)\}$
- A_3 $\{task(1), name(1, choose\_pattern), has(0, 1)\}$
- A_4:$\{task(1), name(1, pattern\_pref), has(0, 1)\}$

The algorithm is now called as follows: $CQ13(1, road\_pattern)$, i.e., the new name of the element should be $road\_pattern$. Let us briefly run through the algorithm. After executing line 2 we obtain $ArgsN = \{A_2, A_3, A_4\}$, since only those arguments contain $name(1, \_)$. Next, on line 3, $B = \{task(1), has(0, 1)\}$, i.e., B is the general argument for the task without the $name$ statement. After line 4 we have

$$A = \{task(1), has(0, 1), name(1, road\_pattern),$$

which is added to $Args$ and attacks arguments A_2, A_3, and A_4.

Algorithms for critical questions CQ6b, CQ6c, CQ6d, CQ7b, and CQ9 (INTRO): The introduction algorithms for the critical questions are all very similar to the INTRO algorithms for argument schemes (Algorithm 2). They have therefore been omitted.

Algorithm for Att (Generic counter argument: Applying a generic counter argument is very simple, and simply results on an attack on the original argument. We illustrate this by continuing our example from Figure 18 (Algorithm 1). The example is shown in Figure 18, where we see that a generic counter argument simply attacks the argument to disable the actor.

[10] Note that part of the arguments (the statements about actors) have been omitted from the figure for readability.

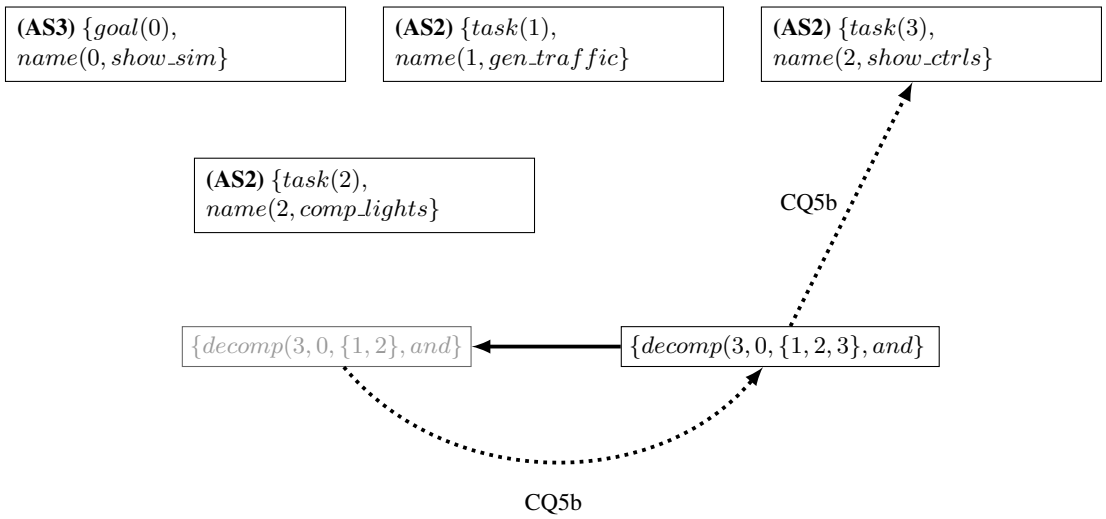

Fig. 16: Example of applying critical question CQ5b (Algorithm 6)

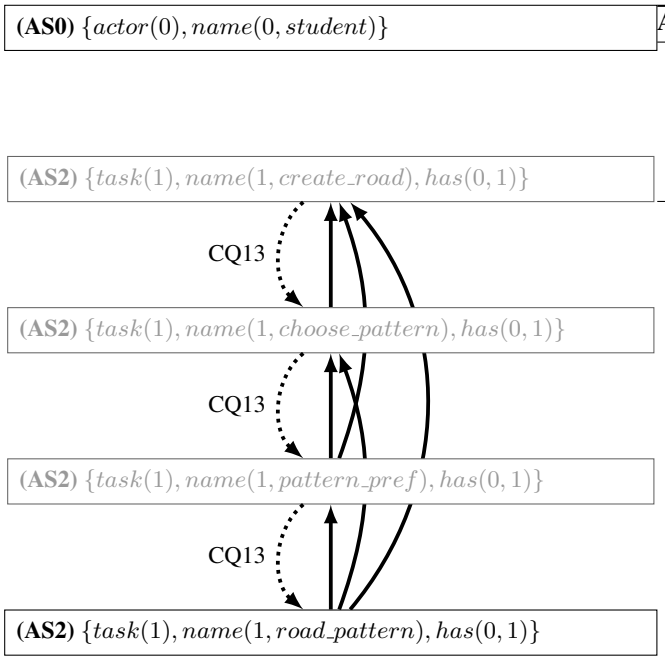

Fig. 17: Applying critical question CQ13 (Algorithm 7) to the example in Figure 10.

Algorithm 8 Generic counterargument to argument A

1: **procedure** ATTACK(A)
2: $A_{new} = \{\}$
3: $Args \leftarrow Args \cup \{A_{new}\}$
4: $Att \leftarrow Att \cup \{(A_{new}, A)\}$
5: **end procedure**

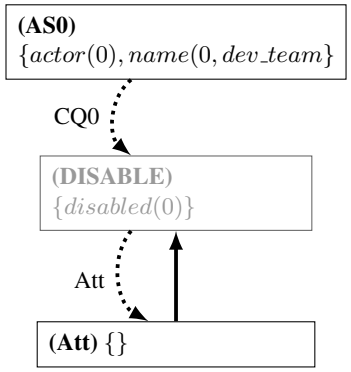

Fig. 18: Formalization of the arguments in Figure 12.

6.4 Constructing GRL models

TODO for F,M(by F): It would be good to have a formal definition of how to construct models given an argumentation framework. If we define a model M at the start of this section, this should be fairly easy.

Constructing GRL models from the arguments is extremely simple: We simply compute the extensions of the argumentation frameworks, and collect all atomic sentences in the accepted arguments. This forms out GRL model. Let us briefly do so for the examples of the previous subsection:

– Figure **??**: Since there are no attacks between the arguments, all atomic sentences are accepted. This results in the following specification:

```
actor(0).
name(0,dev_team).
disabled(0).
```

This again corresponds to the GRL model on the right-hand side of Figure 11.

- Figure 16: There is one rejected argument and five accepted ones. The resulting specification is:

```
goal(0). name(0,show_simulation).
task(1). name(1,generate_traffic).
task(2). name(2,compute_lights).
task(3). name(3,show_controls).
decomp(3,0,{1,2,3},and).
```

- Figure 17: There are only two accepted arguments. The resulting specification is:

```
actor(0). name(0, student).
task(1).  name(1,road_pattern).
has(0,1).
```

  This corresponds to the right-hand GRL model of Figure 10.

- Figure 18. There are two accepted arguments, but the *generic counterargument* does not contain any formulas. Therefore the resulting specification is:

```
actor(0).
name(0,dev_team).
```

  This corresponds to the right-hand GRL model of Figure 12.

7 Discussion

7.1 Related work

In previous work on the RationalGRL framework [36, 38], we have explored one way of combining PRAS and GRL. In this work, which takes a similar approach to [18], argument diagrams are translated to GRL models (an automatic translation tool is discussed in [37]). The argument diagrams are complex practical reasoning arguments, where the premises are the goals and the conclusion is some action we need to perform to realize those goals. Thus, we have essentially two complex diagrams, a practical reasoning argument diagram and a GRL goal diagram, and a mapping between them.

Connecting argumentation and goal modelling by providing a mapping between two types of diagram was, in our opinion, ultimately an unsatisfying solution given the problems and requirements described in Section 1. One problem is that the argument diagram is at least as complex as the GRL diagram, so any stakeholder trying to understand the discussion thus far has to parse two complex diagrams containing goals, alternative solutions, tasks, and so forth. Furthermore, in our previous work we did not provide the specific critical questions for goal models (see Section 3). This meant that any counterargument had to be constructed from scratch, as no guidance was given as to possible ways to criticize a GRL model. So the previous iterations of the RationalGRL framework violated requirement 1: argument diagrams do not closely mirror the actual discussions of stakeholders in which ideas are proposed and challenged. Furthermore, not having specific argument schemes and critical questions for goal modelling makes it hard to develop a guiding methodology for the use of argumentation in goal modelling (requirement 4).

The need for justifications of modeling choices plays an important role in different requirements engineering methods using goal models. High-level goals are often understood as reasons for representing lower-level goals (in other words, the need for low-level goals is justified by having high-level goals) and other elements in a goal model such as tasks and resources. Various refinements and decomposition techniques, often used in requirements engineer (See [34] for an overview), can be seen as incorporating argumentation and justification, in that sub-goals could be understood as arguments supporting parent goals. In that case, a refinement alternative is justified if there are no conflicts between sub-goals (i.e., it is consistent), as few obstacles as possible sub-goals harm sub-goal achievement, there are no superfluous sub-goals (the refinement is minimal), and the achievement of sub-goals can be verified to lead to achieving the parent goal (if refinement is formal [10]). This interpretation is one of the founding ideas of goal modeling. However, while this interpretation may seem satisfactory, argumentation and justification processes differ from and are complementary to refinement in several respects, such as limited possibilities for rationalization and lack of semantics (see Jureta [18] for more details).

There are several contributions that relate argumentation-based techniques with goal modeling. The contribution most closely related to ours is the work by Jureta *et al.* [18]. Jureta *et al.* propose "Goal Argumentation Method (GAM)" to guide argumentation and justification of modeling choices during the construction of goal models. One of the elements of GAM is the translation of formal argument models to goal models (similar to ours). In this sense, our RationalGRL framework can be seen as an instantiation and implementation of part of the GAM. The main difference between our approach and GAM is that we integrate arguments and goal models using argument schemes, and that we develop these argument schemes by analyzing transcripts. GAM instead uses structured argumentation.

The RationalGRL framework is also closely related to frameworks that aim to provide a design rationale (DR) [30], an explicit documentation of the reasons behind decisions made when designing a system or artefact. DR looks at issues, options and arguments for and against the various options in the design of, for example,

a software system, and provides direct tool support for building and analyzing DR graphs. One of the main improvements of RationalGRL over DR approaches is that RationalGRL incorporates the formal semantics for both argument acceptability and goal satisfiability, which allow for a partly automated evaluation of goals and the rationales for these goals.

Arguments and requirements engineering approaches have been combined by, among others, Haley *et al.* [15], who use structured arguments to capture and validate the rationales for security requirements. However, they do not use goal models, and thus, there is no explicit trace from arguments to goals and tasks. Furthermore, like [18], the argumentative part of their work does not include formal semantics for determining the acceptability of arguments, and the proposed frameworks are not actually implemented. Murukannaiah *et al.* [**?**] propose Arg-ACH, an approach to capture inconsistencies between stakeholders' beliefs and goals, and resolve goal conflicts using argumentation techniques.

TODO for all(by Marc): Review the following notes:

- **RE17 paper: merging argumentation theory with law "Using Argumentation to Explain Ambiguity in Requirements Elicitation Interviews" Yehia Elrakaiby, Alessio Ferrari, Paola Spoletini, Stefania Gnesi and Bashar Nuseibeh**
- **more details on Murukannaiah [RE15]**
- **related work on argumentation theory**
- **First paper we wrote they also said something about Bashar Nuseibeh**
- **Also mention argumentation in AI and how that relates to our work**
- **Also paper by Daniel Amyot that he once sent to us: "Synergy between Activity Theory and goal/scenario modeling for requirements elicitation, analysis, and evolution"**
- **Maybe this one: "Complete Traceability for Requirements in Satisfaction Arguments" Anitha Murugesan, Michael Whalen, Elaheh Ghassabani and Mats Heimdahl (University of Minnesota, United States)**
- **At least we can say our work is different from this if it is unrelated..**
- **We should make the comparison more in depth.**
- **Look at related work of Jureta and see how it is related with us.**
- **Maybe the papers above (e.g., Bashar) have related work that relates to us as well.**

7.2 Open issues

We see a large number of open issues that we hope will be explored in future research. We discuss five promising directions here.

Architecture principles

One aspect of enterprise architecture that we did not touch upon in this article are *(enterprise) architecture principles*. Architecture principles are general rules and guidelines, intended to be enduring and seldom amended, that inform and support the way in which an organization sets about fulfilling its mission [21, 28, 31]. They reflect a level of consensus among the various elements of the enterprise, and form the basis for making future IT decisions. Two characteristics of architecture principles are:

- There are usually a small number of principles (around 10) for an entire organization. These principles are developed by enterprise architecture, through discussions with stakeholders or the executive board. Such a small list is intended to be understood *throughout the entire organization*. All employees should keep these principles in the back of their hard when making a decision.
- Principles are meant to guide decision making, and if someone decides to deviate from them, he or she should have a good reason for this and explain why this is the case. As such, they play a normative role in the organization.

Looking at these two characteristics, we see that argumentation, or justification, plays an important role in both forming the principles and adhering to them:

- Architecture principles are *formed* based on underlying arguments, which can be the goals and values of the organization, preferences of stakeholders, environmental constraints, etc.
- If architecture principles are *violated*, this violation has to be explained by underlying arguments, which can be project-specific details or lead to a change in the principle.

In a previous paper, we [24] propose an extension to GRL based on enterprise architecture principles. We present a set of requirements for improving the clarity of definitions and develop a framework to formalize architecture principles in GRL. We introduce an extension of the language with the required constructs and establish modeling rules and constraints. This allows one to automatically reason about the soundness, completeness and consistency of a set of architecture principles. Moreover, principles can be traced back to high-level goals.

It would be very interesting future work to combine the architecture principles extension with the argumentation extension. This would lead to a framework in which principles cannot only be traced back to goals, but also to underlying arguments by the stakeholders.

Extensions for argumentation

The amount of argumentation theory we used in this article has been rather small. Our intention was to create a bridge between the formal theories in argumentation and the rather practical tools in requirements engineering. Now that the initial framework has been developed, is it worth exploring what tools and variations formal argumentation has to offer in more detail.

For instance, until now we have assumed that every argument put forward by a critical questions always defeats the argument it questions, but this is a rather strong assumption. In some cases, it is more difficult to determine whether or not an argument is defeated. Take, for example, the argumentation framework in Figure 19 with just A1 and A2. These two arguments attack each other, they are alternatives and without any explicit preference, and it is impossible to choose between the two. It is, however, possible to include explicit preferences between arguments when determining argument acceptability [1]. If we say that we prefer the action `Create new cars` (A2) over the action `Keep same cars` (A1), we remove the attack from A1 to A2. This makes A2 the only undefeated argument, whereas A1 is now defeated. It is also possible to give explicit arguments for preferences [25]. These arguments are then essentially attacks on attacks. For example, say we prefer A3 over A1 because 'it is important to have realistic traffic flows' (A4). This can be rendered as a separate argument that attacks the attack from A1 to A3, removing this attack and making {A3, A4} the undefeated set of arguments.

Allowing undefeated attacks also make the question of which semantics to choose more interested. In our current (a-cyclic) setting, all semantics coincide, and we always have the same set of accepted arguments. However, once we allow for cycles, we may choose accepted arguments based on semantics which, for instance, try to accept/reject as many arguments as possible (preferred semantics), or just do not make any choice once there are multiple choices (grounded). Another interesting element of having cycles is that one can have multiple extensions. This corresponds to various *positions* are possible, representing various sets of possibly accepted arguments. Such sets can then be shown to the user, who can then argue about which one they deem most appropriate.

Finally, in this article we have only explored one single argument scheme, but there are many other around.

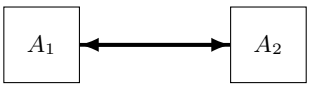

Fig. 19: Preferences between arguments

In his famous book "Argumentation schemes", Walton describes a total of 96 schemes. Murukannaiah *et al.* [26] already explain how some of these schemes may be use for resolving goal conflicts, and it is worth studying what this would look like in our framework as well.

Empirical study

Although we develop our argument schemes and critical questions with some empirical data, we did not yet validate the outcome. This is an important part, because it will allow us to understand whether adding arguments to goal modeling is actually useful. We have developed an experimental setup for our experiment, which we intend to do during courses at various universities. However, we cannot carry out this experiment until the tool is finished.

Formal framework

The formal framework we present in this article is very simple, and does not provide a lot of detail. We believe it would be interesting to develop a more robust characterization of a GRL model using logical formulas. Right now, we have no way to verify whether the goal models we obtain through out algorithms are actually valid GRL models. This is because we allow any set of atoms to be a GRL model, which is clearly very permissive and incorrect. Once we develop a number of such constraints, we can ensure (and even proof) our algorithms do not generate invalid GRL models.

For instance, suppose we assert that an *intentional element* is a goal, softgoal, task, or resource:

$$(softgoal(i) \vee goal(i) \vee task(i) \\ \vee resource(i) \rightarrow IE(i).$$

We can then formalize an intuition such as: "Only intentional elements can be used in contribution relations" as follows

$$contrib(k, i, j, ctype) \rightarrow \\ (IE(i) \wedge IE(j) \wedge IE(j).$$

Interestingly, such constraints are very comparable to *logic programming* rules. We therefore see it as interesting future research to explore this further, specifically in the following two ways:

– Develop a set of constraints on sets of atoms of our language, which correctly describe a GRL model. Show formally that using our algorithms, each extension of the resulting argumentation framework corresponds to a valid GRL model, i.e., a GRL model that does not violate any of the constraints.
– Implement the constraints as a logic program, and use a logic programming language to compute the resulting GRL model.

7.3 Conclusion

TODO for all(by Marc): Finish this

The introduction of this article contains five requirements we identified for our framework. We use the conclusion to discuss how RationalGRL meets our initial requirements.

1. The argumentation techniques should be close to actual discussions stakeholders or designers have. We analyze a set of transcripts containing discussions about the architecture of an information system.

2. The framework must have the means to formally model parts of the discussion process. In order to generate goal models based on formalized discussions (requirement 2), we, first, formalize the list of arguments from requirement 1 in an argumentation framework. We formalize the critical questions as algorithms modifying the argumentation framework. We use argumentation techniques from AI in order to determine which arguments are accepted and which are rejected. We propose an algorithm to generate a GRL model based on the accepted arguments. This helps providing traceability links from GRL elements to the underlying arguments (requirement 3).

We implement our framework in an online tool called RationalGRL (requirement 4). The tool is implemented using Javascript. It contains two parts, goal modeling and argumentation. The goal modeling part is a simplified version of GRL, leaving out features such as evaluation algorithms and key performance indicators. The argumentation part is new, and we develop a modeling language for the arguments and critical questions. The created GRL models in RationalGRL can be exported to jUCMNav [] the Eclipse-based tool for GRL modeling, for further evaluation and analysis.

Our final contribution is a methodology on how to develop goal models that are linked to underlying discussions. The methodology consists of two parts, namely argumentation and goal modeling. In the argumentation part, one puts forward arguments and counter-arguments by applying critical questions. When switching to the goal modeling part, the accepted arguments are used to create a goal model. In the goal modeling part, one simply modifies goal models, which may have an effect on the underlying arguments. This might mean that the underlying arguments are no longer consistent with the goal models.

Acknowledgments

Marc van Zee is funded by the National Research Fund (FNR), Luxembourg, by the Rational Architecture project.

References

1. L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3):197–215, 2002.
2. D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. S. K. Yu. Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 25:841–877, August 2010.
3. K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171(10):855–874, 2007.
4. T. J. Bench-Capon and P. E. Dunne. Argumentation in Artificial Intelligence. *Artificial intelligence*, 171(10-15):619–641, 2007.
5. E. Black, P. McBurney, and S. Zschaler. Towards Agent Dialogue as a Tool for Capturing Software Design Discussions. In *Proceedings of the 2nd International Workshop on Theory and Applications of Formal Argumentation*, pages 95–110. Springer, 2013.
6. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.
7. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.
8. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.
9. A. Dardenne, A. Van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.
10. R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 179–190, New York, NY, USA, 1996. ACM.
11. P. Donzelli. A goal-driven and agent-based requirements engineering framework. *Requirements Engineering*, 9(1):16–39, 2004.

12. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.
13. S. Ghanavati. *Legal-urn framework for legal compliance of business processes*. PhD thesis, University of Ottawa, 2013.
14. P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.
15. C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh. Arguing security: Validating security requirements using structured argumentation. In *in Proc. of the Third Symposium on RE for Information Security (SREIS'05)*, 2005.
16. J. Horkoff, D. Barone, L. Jiang, E. Yu, D. Amyot, A. Borgida, and J. Mylopoulos. Strategic business modeling: representation and reasoning. *Software & Systems Modeling*, 13(3):1015–1041, 2014.
17. ITU-T. Recommendation Z.151 (11/08): User Requirements Notation (URN) – Language Definition. `http://www.itu.int/rec/T-REC-Z.151/en`, 2008.
18. I. Jureta, S. Faulkner, and P. Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Requirements Engineering*, 13(2):87–115, May 2008.
19. E. Kavakli and P. Loucopoulos. Goal modeling in requirements engineering: Analysis and critique of current methods. In J. Krogstie, T. A. Halpin, and K. Siau, editors, *Information Modeling Methods and Methodologies*, pages 102–124. Idea Group, 2005.
20. S. D. kenniscentrum. DEMO: The KLM case. `http://www.demo.nl/attachments/article/21/080610_Klantcase_KLM.pdf`, Last accessed on 22 August, 2012.
21. M. Lankhorst. *Enterprise architecture at work: modelling, communication, and analysis*. Springer, 2005.
22. L. Liu and E. Yu. Designing information systems in social context: a goal and scenario modelling approach. *Information systems*, 29(2):187–203, 2004.
23. J. G. March. Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, pages 587–608, 1978.
24. D. Marosin, M. van Zee, and S. Ghanavati. Formalizing and modeling enterprise architecture (ea) principles with goal-oriented requirements language (grl). In *Proceedings of the 28th International Conference on Advanced Information System Engineering (CAiSE16)*, June 2016.
25. S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9):901–934, 2009.
26. P. K. Murukannaiah, A. K. Kalia, P. R. Telangy, and M. P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *Proceedings of the 23rd International Requirements Engineering Conference (RE 2015)*, pages 156–165. IEEE, 2015.
27. G. Mussbacher and D. Amyot. Goal and scenario modeling, analysis, and transformation with jUCMNav. In *ICSE Companion*, pages 431–432, 2009.
28. M. Op 't Land and E. Proper. Impact of principles on enterprise engineering. In H. sterle, J. Schelp, and R. Winter, editors, *ECIS*, pages 1965–1976. University of St. Gallen, 2007.
29. C. Schriek, J. M. E. van der Werf, A. Tang, and F. Bex. Software Architecture Design Reasoning: A Card Game to Help Novice Designers. In *Proceedings of the 10th European Conference on Software Architecture (ECSA 2016), Copenhagen, Denmark*, pages 22–38. Springer, 2016.
30. S. J. B. Shum, A. M. Selvin, M. Sierhuis, J. Conklin, C. B. Haley, and B. Nuseibeh. Hypermedia support for argumentation-based rationale. In *Rationale management in software engineering*, pages 111–132. Springer, 2006.
31. The Open Group. *The Open Group – TOGAF Version 9*. Van Haren Publishing, Zaltbommel, The Netherlands, 2009.
32. P. Tolchinsky, S. Modgil, K. Atkinson, P. McBurney, and U. Cortés. Deliberation dialogues for reasoning about safety critical actions. *Autonomous Agents and Multi-Agent Systems*, 25(2):209–259, 2012.
33. UCI. Design Prompt: Traffic Signal Simulator. `http://www.ics.uci.edu/design-workshop/files/UCI_Design_Workshop_Prompt.pdf`. Accessed: 2016-12-27.
34. A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. 5th IEEE Int. Symposium on RE*, pages 249–262, 2001.
35. A. van Lamsweerde. Requirements engineering: from craft to discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 238–249. ACM, 2008.
36. M. van Zee, F. Bex, and S. Ghanavati. Rationalization of Goal Models in GRL using Formal Argumentation. In *Proceedings of RE: Next! track at the Requirements Engineering Conference 2015 (RE'15)*, August 2015.
37. M. van Zee, D. Marosin, F. Bex, and S. Ghanavati. The rationalgrl toolset for goal models and argument diagrams. In *Proceedings of the 6th International Conference on Computational Models of Argument (COMMA'16), Demo abstract*, September 2016.
38. M. van Zee, D. Marosin, S. Ghanavati, and F. Bex. Rationalgrl: A framework for rationalizing goal models using argument diagrams. In *Proceedings of the 35th International Conference on Conceptual Modeling (ER'2016), Short paper*, November 2016.
39. D. Walton, C. Reed, and F. Macagno. *Argumentation schemes*. Cambridge University Press, 2008.
40. D. N. Walton. *Practical reasoning: goal-driven, knowledge-based, action-guiding argumentation*, volume 2. Rowman & Littlefield, 1990.
41. E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proc. of the 3rd IEEE Int. Symposium on RE*, pages 226–235, 1997.
42. E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society.

A UCI Design Workshop Prompt

Design Prompt: Traffic Signal Simulator

Problem Description

For the next two hours, you will be tasked with designing a traffic flow simulation program. Your client for this project is Professor E, who teaches civil engineering at UCI. One of the courses she teaches has a section on traffic signal timing, and according to her, this is a particularly challenging subject for her students. In short, traffic signal timing involves determining the amount of time that each of an inter- section's traffic lights spend being green, yellow, and red, in order to allow cars in to flow through the intersection from each direction in a fluid manner. In the ideal case, the amount of time that people spend waiting is minimized by the chosen settings for a given intersection's traffic lights. This can be a very subtle matter: changing the timing at a single intersection by a couple of seconds can have far- reaching effects on the traffic in the surrounding areas. There is a great deal of theory on this subject, but Professor E. has found that her students find the topic quite abstract. She wants to provide them with some software that they can use to "play" with different traffic signal timing schemes, in different scenarios. She anticipates that this will allow her students to learn from practice, by seeing first-hand some of the patterns that govern the subject.

Requirements

The following broad requirements should be followed when designing this system:

1. Students must be able to create a visual map of an area, laying out roads in a pattern of their choosing. The resulting map need not be complex, but should allow for roads of varying length to be placed, and different arrangements of intersections to be created. Your approach should readily accommodate at least six intersections, if not more.
2. Students must be able to describe the behavior of the traffic lights at each of the intersections. It is up to you to determine what the exact interaction will be, but a variety of sequences and timing schemes should be allowed. Your approach should also be able to accommodate left-hand turns protected by left-hand green arrow lights. In addition:
   (a) Combinations of individual signals that would result in crashes should not be allowed.
   (b) Every intersection on the map must have traffic lights (there are not any stop signs, over- passes, or other variations). All intersections will be 4-way: there are no "T" intersections, nor one-way roads.
   (c) Students must be able to design each intersection with or without the option to have sensors that detect whether any cars are present in a given lane. The intersection's lights' behavior should be able to change based on the input from these sensors, though the exact behavior of this feature is up to you.
3. Based on the map created, and the intersection timing schemes, the students must be able to simulate traffic flows on the map. The traffic levels should be conveyed visually to the user in a real-time manner, as they emerge in the simulation. The current state of the intersections' traffic lights should also be depicted visually, and updated when they change. It is up to you how to present this information to the students using your program. For example, you may choose to depict individual cars, or to use a more abstract representation.
4. Students should be able to change the traffic density that enters the map on a given road. For ex- ample, it should be possible to create a busy road, or a seldom used one, and any variation in between. How exactly this is declared by the user and depicted by the system is up to you. Broadly, the tool should be easy to use, and should encourage students to explore multiple alternative approaches. Students should be able to observe any problems with their map's timing scheme, alter it, and see the results of their changes on the traffic patterns. This program is not meant to be an exact, scientific simulation, but aims to simply illustrate the basic effect that traffic signal timing has on traffic. If you wish, you may assume that you will be able to reuse an existing software package that provides relevant mathematical functionality such as statistical distributions, random number generators, and queuing theory.

You may add additional features and details to the simulation, if you think that they would support these goals.

B Transcript excerpts

C GRL Specification

Speaker	*Text*	*Coding*
0:15:11 (P1)	And then, we have a set of actions. Save map, open map, add and remove intersection, roads	**[20 task (AS2)]** Student has tasks "save map", "open map", "add intersection", "remove intersection", "add road", "add traffic light" [INTRO]
0:15:34 (P2)	Yeah, road. Intersection, add traffic lights	
0:15:42 (P1)	Well, all intersection should have traffic lights so it's	**[21 critical question CQ11 for 20]** Is the task "Add traffic light" useful/relevant?
0:15:44 (P2)	Yeah	
0:15:45 (P1)	It's, you don't have to specifically add a traffic light because if you have	**[22 answer to 21]** Not useful, because according to the specification all intersections have traffic lights.
0:15:51 (P2)	They need-	[DISABLE]

Table 3: Adding tasks, disabling unnecessary task "Add traffic light" (transcript t_1)

Speaker	*Text*	*Coding*
0:22:52 (P1)	We also have to be able to change the inflow of cars. How many cars come out in here on the side	**[31 task (AS2)]** Student has task "Set car influx" [INTRO]
0:23:20 (P1)	So, sets, yeah, car influx.	
0:23:41 (P2)	If you can only control the set amount of influx from any side of this sort of random distribution, I think that is going to be less interesting than when you can say something like, this road is frequently travelled.	**[32 critical question CQ12 for 31]** Is "Set car influx" specific enough? **[33 answer to 32]** No, "Set car influx" becomes "Set car influx per road" [REPLACE]
0:24:12 (P2)	So setting it per road, I think is something we want	**[34 critical question CQ12 for 33]** Is "Set car influx per road" clear?
0:24:15 (P1)	Yeah, that was also one of the constraints I believe	**[35 answer to 34]** Yes, "Set car influx per road" was one of the original constraints.

Table 4: Clarifying the name of a task (transcript t_1)

Speaker	*Text*	*Coding*
0:18:55 (P1)	Yeah. And then two processes, static, dynamic and they belong to the goal simulate.	**[17 goal (AS3)]** System has goal "Simulate" [INTRO] **[18 task (AS2)]** System has tasks "Static simulation", "Dynamic simulation" [INTRO] **[20 decomposition (AS5)]** Goal "Simulation" AND-decomposes into "Static simulation" and "Dynamic simulation" [INTRO]
0:30:10 (P1)	Yeah. But this is- is this an OR or an AND?	**[26 critical question CQ10b for 20]** Is the decomposition type of "simulate" correct? **[27 answer to 26]** No, it should be an OR decomposition. [REPLACE]
0:30:12 (P2)	That's and OR	
0:30:14 (P3)	I think it's an OR	
0:30:15 (P1)	It's for the data, it's an OR	
0:30:18 (P3)	Yep	

Table 5: Incorrect decomposition type for goal *Simulate* (transcript t_3)

Respondent	Text	Annotation
0:10:55.2 (P1)	Maybe developers	**[4 actor (AS0)]** Development team
0:11:00.8 (P2)	Development team, I don't know. Because that's- in this context it looks like she's gonna make the software	**[5 critical question CQ0 for 4]** Is actor "development team" relevant? **[6 answer to 5]** No, it looks like the professor will develop the software.
0:18:13.4 (P2)	I think we can still do developers here. To the system	**[16 counter argument for 6]** According to the specification the professor doesn't actually develop the software.
0:18:18.2 (P1)	Yeah?	
0:18:19.8 (P2)	Yeah, it isn't mentioned but, the professor does-	
0:18:22.9 (P1)	Yeah, when the system gets stuck they also have to be [inaudible] ok. So development team	

Table 6: Discussion about the relevance of an actor (transcript t_3)

```
%% GRL Elements
actors([1,24,43]).
ies([2...17, 25...34, 44, 45]).
links([18...23, 35...42, 47, 48, 49]).

%%%%% Actor student %%%%%%
name(1, student).

% IE types of actor student
softgoal(2).
goal(3).
tasks(4). task(5). ... task(17).

% Containments of actor student
has(1, 2). has(1, 3). ... has(1, 17).

% IE names of actor student
name(2, learn_queuing_theory_from_practice).
name(3, use_simulator).
name(4, map_design).
name(5, open_map).
name(6, add_road).
name(7, add_sensor_to_traffic_light).
name(8, control_grid_size).
name(9, add_intersection).
name(10, run_simulation).
name(11, save_map).
name(12, control_simulation).
name(13, control_car_influx_per_road).
name(14, adjust_car_spawing_rate).
name(15, adjust_timing_schemes_of \
           sensorless intersections).
name(16, remove_intersection).
name(17, add_traffic_light).

% Links of actor student
contr(18, 3, 2, pos).
contr(19, 4, 3, pos).
contr(20, 11, 3, pos).
decomp(21, 4, [5...11], and).
decomp(22, 4, [5...11, 13], and).
decomp(23, 12, [13,14,15], and).

% Disabled elements of actor student
disabled(16). disabled(17). disabled(22).

%%%% Actor Traffic Tycoon %%%%%%%/
name(24, traffic_tycoon).

% IE types of actor Traffic Tycoon
softgoal(25). softgoal(26).
goal(27). goal(28).
task(29). ... task(32).
resource(33). resource(34).

% Containments of actor Traffic Tycoon
has(24, 25). ... has(24,34).

% IE names of actor Traffic Tycoon
name(25, dynamic_simulation).
name(26, realistic_simulation).
name(27, show_simulation).
name(28, generate_cars).
name(29, keep same_cars).
name(30, create_new_cars).
name(31, show_map_editor).
name(32, store_load_map).
name(33, external_library).
name(34, storage).

% Links of actor Traffic Tycoon
contr(35, 29, 25, neg).
contr(36, 29, 26, pos).
contr(37, 30, 25, pos).
contr(38, 30, 26, neg).
decomp(39, 28, [29, 30], xor).
decomp(40, 27, [28, 33], and).
decomp(41, 31, [32], and).
decomp(42, 32, [34], and).

%%%% Actor Teacher  %%%%%%%
name(43, teacher).

% IE types of actor Teacher
softgoal(44).
task(45).

% Containments of actor Teacher
has(43, 44). has(43,45).

% IE names of actor Teacher
name(44, students_learn_from_practice).
name(45, pass_students_if_simulation_is_correct).

% Disabled elements of actor Teacher
disabled(45).

%%%%%% Dependencies  %%%%%
goal(46).
name(46, value_has_changed).

dep(47, 32, 46).
dep(48, 46, 14).
dep(49, 32, 11).

%%%%%% Rules for containment  %%%%%
has(Act,E1) :- has(Act, E2), decomposes(_,E2,X,_),
       member(E1,X).
has(Act,E1) :- has(Act,E1), contr(E2, E1,_).
```