RationalGRL: A Framework for Argumentation and Goal Modeling

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

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

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

[3]Department of Computer Science
Texas Tech University
sepideh.ghanavati@ttu.edu

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 its 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 [45]. 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 (overviews can be found in [20,38]). Several goal modeling languages have been developed in the last two decades as well. The most popular ones include $i*$ [45], Keep All Objects Satisfied (KAOS) [39], the NFR framework [7], TROPOS [14], the Business Intelligence Model (BIM) [17], 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 healthcare 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: 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 modeling sessions, which 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 [5]. 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 [40, 42], we develop a framework called *RationalGRL*, which combines the Goal-oriented Requirements Language (GRL) [2] with a technique from argumentation theory called *argument schemes* (or argumentation schemes [43]). 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 [27].

One argument scheme that is important when reasoning about goals is the argument scheme for practical reasoning [44, 3], which has been used for, among other things, dialogues about safety critical actions [36] and software design discussions [6]. 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 [9]. 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 [9] 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.

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. [33], who in turn base their work on the so-called Irvine experiment [37], 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 2, 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. The GRL elements and relationships used in this paper are shown in Figure 1. Note that we only consider a subset of the full GRL specification in this paper. In particular, the full GRL specification considers different levels of positive and negative contributions and satisfaction levels of goals and tasks, which are not directly relevant for this work. Furthermore, the full GRL specification includes beliefs to capture design rationale, whereas in this paper, the reasoning and rationales behind the models are captured using arguments – for a brief discussion on beliefs, see Section 7.1.

Figure 2 illustrates a simplified GRL diagram from the traffic simulator design exercise. An actor represents a stake-

[1] `insertURL`

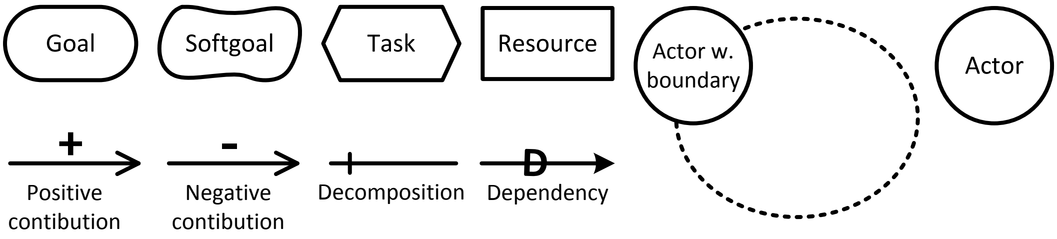

Fig. 1: Basic elements and relationships of GRL

holder of a system or the system itself (*Traffic Simulator*, Figure 2). 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 2, 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*).

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 2, 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 impacts of one element on another element, which can be positive or negative. Task *Create new cars* has a positive contribution to the task *Realistic simulation*, and a negative 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*.

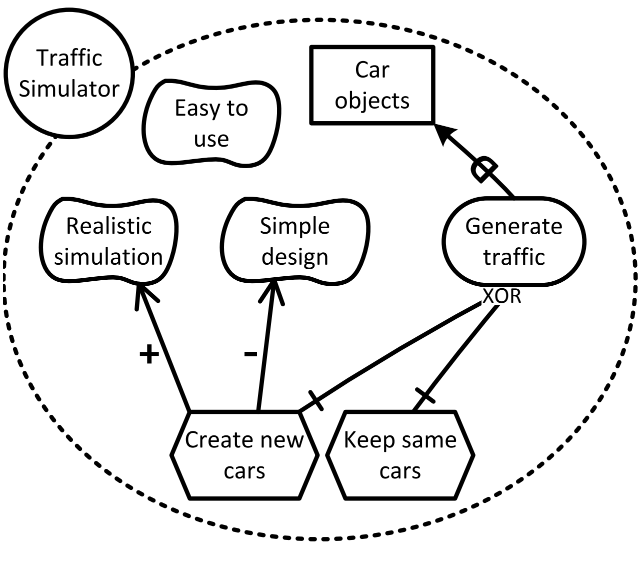

Fig. 2: Partial GRL Model of the traffic simulator example

GRL is based on $i*$ [**?**] 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, 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 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 [28]. 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 2 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 softgoal *Easy to use* not linked to any of the goals or tasks?
- 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 2 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 [44]. 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.

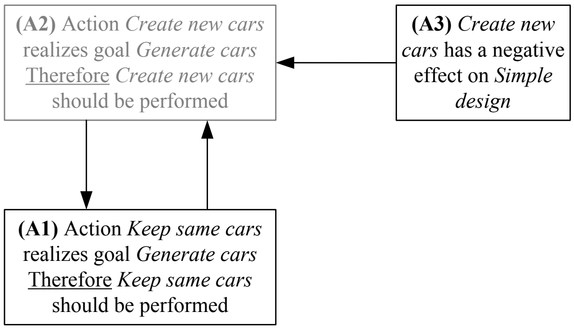

Fig. 3: PRAS arguments and attacks in the traffic simulation example.

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 [9][2]. Figure 3 shows three arguments

[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.

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.

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].

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.

In order to obtain actual requirements discussions, we turned to a recent series of experiments by Schriek et al. [33]. 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

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

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).

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 subtasks, 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 goal models by analyzing the non-functional requirements and 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, 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

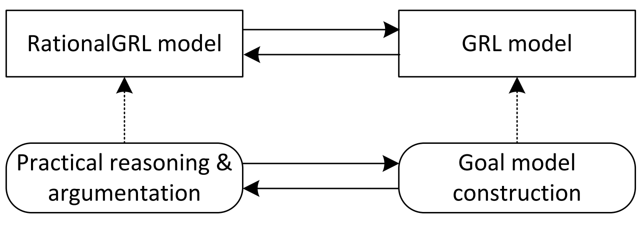

Fig. 4: The RationalGRL Framework

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 have negative side-effects?	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).

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 goal model. Making an argument based on one of the 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.

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.

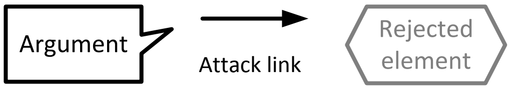

Fig. 5: The new elements and link of RationalGRL

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 1. 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 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 RatinalGRLmodel 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

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

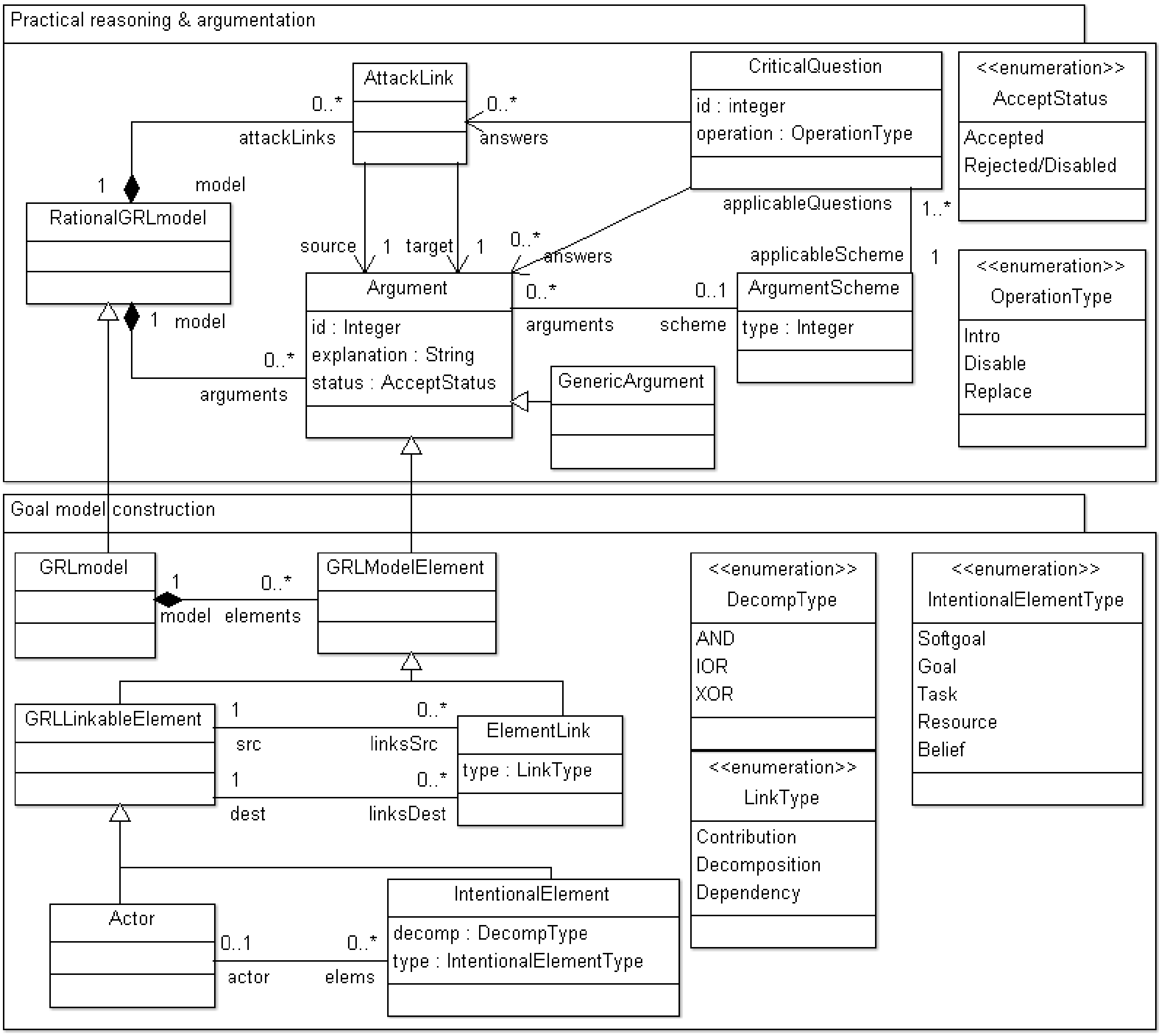

Fig. 6: The RationalGRL metamodel

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 6.4). The connection between the RationalGRL models shown in this section and regular GRL models is further formally defined in section **??**.

*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 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

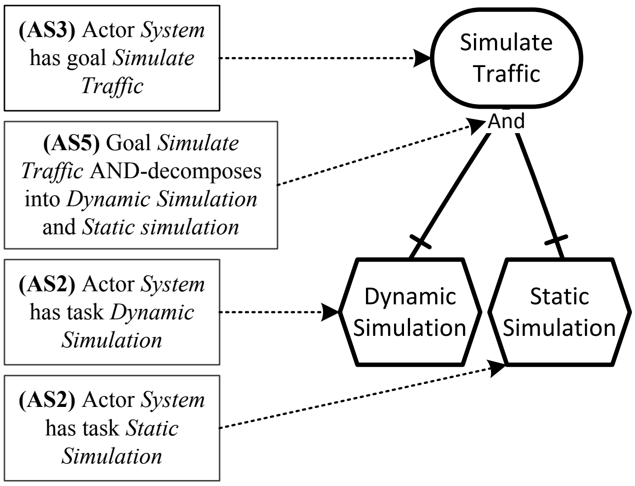

Fig. 7: Introducing new GRL elements with argumentation (INTRO).

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

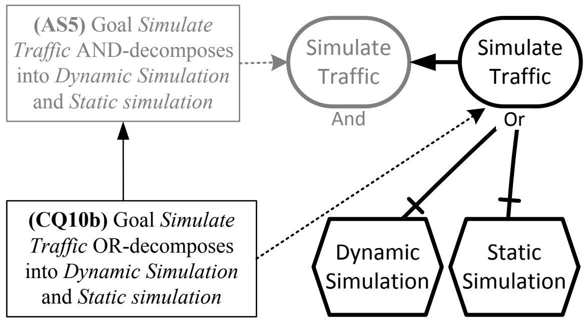

Fig. 9: Replacing a GRL decomposition with argumentation (REPLACE).

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 argument 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).

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

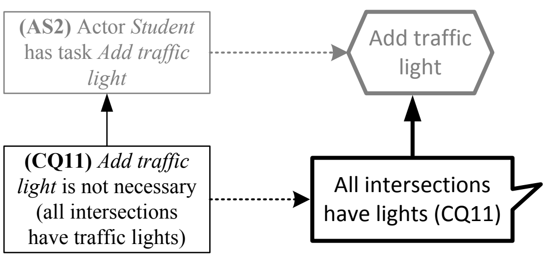

Fig. 8: Disabling GRL tasks with argumentation (DISABLE).

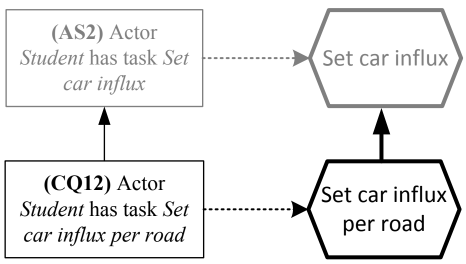

Fig. 10: Renaming a GRL task with argumentation (REPLACE).

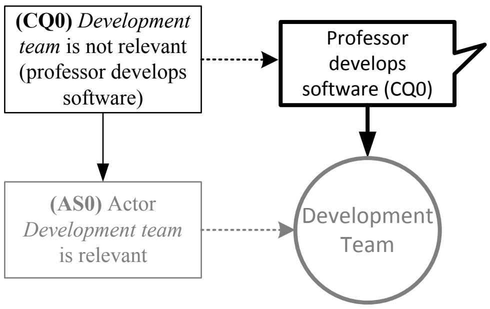

Fig. 11: Disabling a GRL actor with argumentation (DISABLE).

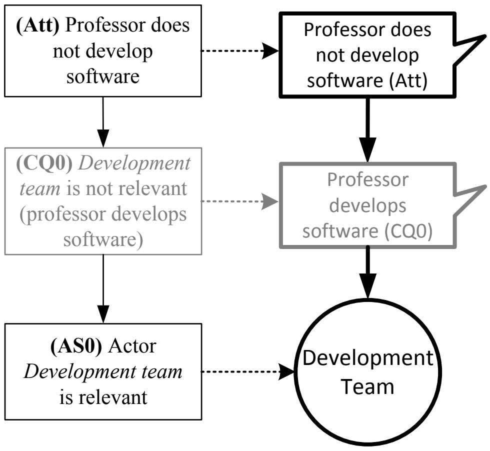

Fig. 12: Defending a GRL actor by attacking its disabling attacker (DISABLE).

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 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

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] (**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) – We start with the list of argument schemes (Table 2). Whilst discussing the requirements, we 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 elements (INTRO). 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 start asking the 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.

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

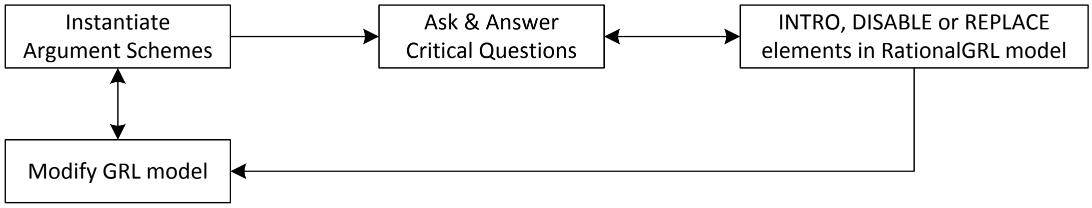

Fig. 13: The RationalGRL Methodology

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 [28]. 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 **??** 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 and visa versa in a 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 4.2) 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.

In the third subsection we develop algorithms in order to translate a GRL model into a RationalGRL model, and visa versa.

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

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.

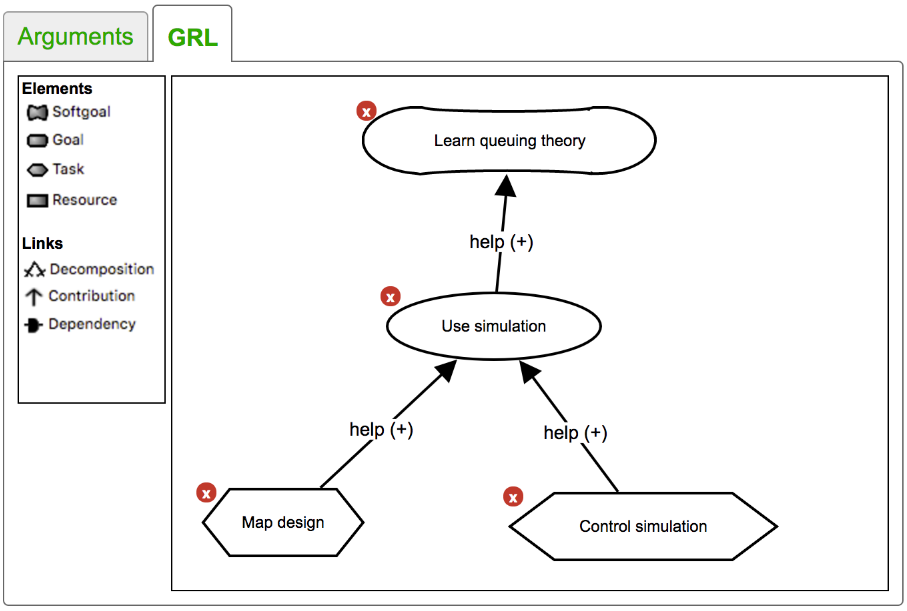

Fig. 14: Screenshot of the prototype tool

6.1.1 Intentional Element, beliefs 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:

- $IETypes = \{Softgoal, Goal, Task, Resource\}$,
- $Types = IETypes \cup \{Actor, Contr, Dep, Decomp, Belief, GenArg\}$
- $Names$ is a finite set of strings.
- $DecompTypes = \{AND, OR, XOR\}$.
- $ContribValues = \{Break, Hurt, Some\ nega-tive, Unknown, Some\ positive, Help, Make\}$,

Next we define an intentional element.

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

- $id \in \mathbb{N}$ is a unique identifier for the element,
- $type \in IETypes$ 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]

[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.

A set of intentional elements is denoted by IE.

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 a tuple using the dot (".") notation. That is, we may for instance refer to the id, type, name and decomposition type of an IE with respectively $ie.id$, $ie.type$, $ie.name$, and $ie.decomptype$.
- We refer to a set of elements with the same id i using the i subscript on the set. For instance a set of IEs with id i is denoted by IE_i, and if this is a single element, we denote it 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 $ie_0.type = Goal$, $ie_0.name = \text{Make profit}$, 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 $goal_0.name = \text{Make profit}$, and $goal_0.decomptype = AND$.

[7] In GRL, there always exists at most one element for every id (see Def. 11), in RationalGRL, however, this condition does not hold (see Def. 16).

Rationale and example Throughout this section we use a slightly adapted version of the example from Figure 2, which is shown in Figure 15. Our formalization of GRL is very much in line with the way in which GRL models are represented in the open-source Eclipse-based tool jUCMNav.[8] This tool is actively developed and has a rich number of features for the analysis of both GRL and URN models. By keeping our formalization in line with this tool, we simplify the translation step from models in the RationalGRL tool to models in the jUCMNav tool.

Some of the IEs in Figure 15 can be formalized using Def. 2 as follows:

- $(2, goal, \text{Generate traffic}, XOR)$: This represents the goal "Generate traffic", which is XOR-decomposed. Note the identifier "2" is not shown in the GRL model, but it is stored internally.
- $(5, resource, \text{Car objects}, AND)$: Although this element does not decompose into any other elements, it still contains an AND-decomposition. Our motivation for this choice is that it simplifies the formalization of decomposition links. It is in line with with GRL metamodel and the jUCMNav implementation.

Using the short-hand notation defined in Def. 3, we can make the following statements:

- $goal_2.name = \text{Generate traffic}$
- $goal_2.decomptype = XOR$
- $resource_5.name = \text{Car objects}$

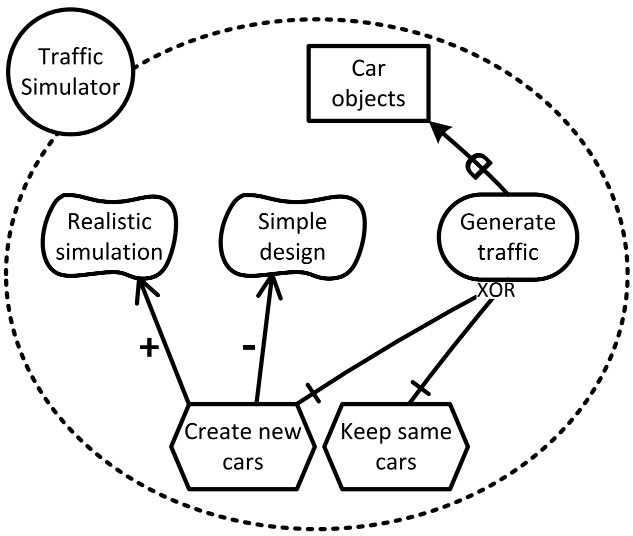

Fig. 15: Example GRL model

[8] See http://jucmnav.softwareengineering.ca/foswiki/ProjetSEG

Definition 4 (Belief) A belief $bel \in \mathbb{N} \times Names$ is a relation where $bel = (id, name)$ means:

- $id \in \mathbb{N}$ is the identifier of the belief,
- $name \in Names$ is a string description of the belief.

Rational and example Beliefs are different from IEs in the sense that they are not being used in decompositions, dependencies, or contributions. They merely provide additional explanation for an element. The only belief in Figure 15 is formalized as $(1, \text{Not scientifically correct})$.

Definition 5 (Actor) An actor $act \in \mathbb{N} \times Names$ is a relation where $act = (id, name)$ means:

- $id \in \mathbb{N}$ is the identifier of the actor,
- $name \in Names$ is 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.

Rationale and example An actor is simply a pair consisting of its identifier and its name. The relation between actors and their intentional element is formalized in the next definition.

We can formalize the actor of Figure 15 as $act = (0, \text{Traffic Simulator})$, and we can for instance state $act_0.name = \text{Traffic Simulator}$.

Definition 6 (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}.

Rationale and example The previous definition specifies relations between actors and intentional elements. Each actor can own one or multiple intentional elements, but each intentional element can only be owned by at most one actor (we will make these assumptions explicit in Def. 12).

In Figure 15, recall we formalized two IEs and the actor as follows:

- $(2, goal, \text{Generate traffic}, XOR)$
- $(5, resource, \text{Car objects}, AND)$
- $(0, \text{Traffic Simulator})$

We can then formalize the Actor-IE relationships for these elements as: $(0, 2)$ and $(0, 5)$.

6.1.2 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 7 (Belief link) A *belief* link: $bellink \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ is a relation such that $bellink = (i, j)$ means

- $i \in \mathbb{N}$ is the unique identifier of the belief element,
- $j \in \mathbb{N}$ is the unique identifier of the IE.

Intuitively, $bellink = (i, j)$ means that bel_i is a belief for ie_j.

A set of belief links is denoted by $Bellink$.

Rationale and example The belief links can only be used to connect a belief with an intentional element, and they are not used in any of the algorithms. Instead, they can be understood as metadata, they provide additional documentation on the rationale for making certain design choices. Therefore, belief links do not require unique identifiers.

The fact that belief links only connect beliefs with IEs is not made explicit in the definition above. We make this assumption, together with various assumption, explicit in the definition of a *valid GRL model* (Def. 12). Figure 15 contains one belief links. It connects the following two elements:

- $(1,$ Not scientifically correct$)$,
- $(4, softgoal,$ Simple design$, AND)$.

The belief link is then formalized as $(1, 4)$.

Definition 8 (Contribution Link) A *contribution* link: $contrib \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times Contribvalues$ is a relation such that $contrib = (i, j, k, value)$ means

- $i \in \mathbb{N}$ is the unique identifier of the link,
- $j \in \mathbb{N}$ is the unique identifier of the IE from which the contribution originates,
- $k \in \mathbb{N}$ is the unique identifier of the IE to which is contributed.
- $value \in ContribValues$ is the strength of the contribution.

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

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

Rationale and example Similar to IEs, links have identifiers as well. In Fig. 15, there are two contribution links. First, let us formalize the related IEs:

- $(3, softgoal,$ Realisitc simulation$, AND)$
- $(4, softgoal,$ Simple design$, AND)$
- $(6, task,$ Create new cars$, AND)$

We can formalize the contribution links as $(9, 6, 3,$ Help$)$ and $(10, 6, 4,$ Hurt$)$.

Definition 9 (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

- $i \in \mathbb{N}$ is the unique identifier of the link,
- $j \in \mathbb{N}$ is the unique identifier of the decomposing IE,
- $k \in \mathbb{N}$ is the unique identifier of the IE that is being decomposed.

Intuitively, $decomp = (i, j, k)$ means that ie_i decomposes ie_j.[9]

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

Rationale and example Decomposition links are simpler than contribution links, because they only contain information about which elements are being connected. Recall that the decomposition type is stored in the element that is being decomposed. In order to formalize the decomposition links in Figure 15, let us first formalize the related IEs:

- $(5, goal,$ Generate traffic$, XOR)$
- $(6, task,$ Create new cars$, AND)$
- $(7, task,$ Keep same cars$, AND)$

The decomposition links can be formalized as $(11, 6, 5)$ and $(12, 7, 5)$.

Definition 10 (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

- $i \in \mathbb{N}$ is the unique identifier of the link,
- $j \in \mathbb{N}$ is the unique identifier of the dependee IE,
- $k \in \mathbb{N}$ is the unique identifier of the dependent IE.

Intuitively, $dep = (i, j, k)$ means that ie_j depends on ie_j.

A set of dependency links is denoted by Dep.

Rationale and example The representation of dependency links is exactly the same as decomposition links. In Figure 15, let us formalize the IEs related to the only dependency link:

- $(2, resource,$ Car objects$, AND)$
- $(5, goal,$ Generate traffic$, XOR)$

The dependency link between these two elements is formalized as $(8, 5, 2)$.

Definition 11 (GRL Model) A *GRL model* $GRL = (IE, Bel, Act, R_{ActIE}, Bellink, Contr, Decomp, Dep)$ consists of:

- A set IE of intentional elements (Def. 2),
- A set Bel of beliefs (Def. 4),
- A set Act of actors (Def. 5),
- A set R_{ActIE} of Actor-IE relations (Def. 6),
- A set $Bellink$ of belief links (Def. 7),
- A set $Contr$ of contribution links (Def. 8),
- A set $Decomp$ of decomposition links (Def. 9),
- A set Dep of dependency links (Def. 10).

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

Rationale and example The definition of a GRL model collects all the previously defined tuples into a single definition. For completeness, we now provide the full specification of Figure 15. This model is formalized as $GRL = (IE, Bel, Act, R_{ActIE}, Bellink, Contr, Decomp, Dep)$ where:

$$\begin{aligned}
IE = \quad & \{(2, task, \text{Car objects}, AND),\\
& (3, softgoal, \text{Realistic simulation}, AND),\\
& (4, softgoal, \text{Simple design}, AND),\\
& (5, goal, \text{Generate traffic}, XOR),\\
& (6, task, \text{Create new cars}, AND),\\
& (7, task, \text{Keep same cars}, AND)\}\\
Bel = \quad & \{(1, \text{Not scientifically correct})\}\\
Act = \quad & \{(0, \text{Traffic Simulator})\}\\
R_{ActIE} = \quad & \{(0, i) \mid 2 \leq i \leq 7\}\\
Bellink = \quad & \{(1, 4)\}\\
Contr = \quad & \{(9, 6, 3, \text{Help}), (10, 6, 4, \text{Hurt})\}\\
Decomp = \quad & \{(11, 6, 5), (12, 7, 5)\}\\
Dep = \quad & \{(8, 5, 2)\}
\end{aligned}$$

Definition 12 (Valid GRL Model) A GRL model $GRL = (IE, Act, R_{ActIE}, Contr, Decomp, Dep)$ (Def. 11) is a *valid GRL model* iff the following conditions are satisfied:

1. ids are globally unique across IEs, Beliefs, Links, and Actors, i.e., let $X, Y \in \{IE, Act, Belief, Contr, Decomp, Dep\}$. 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: $\forall (i, j) \in R_{ActIE} : act_i \in Act \wedge ie_j \in IE$.
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: $\forall (i, j, k) \in Dep : \{ie_j, ie_k\} \subseteq IE$.
7. Belief links connect beliefs with intentional elements: $\forall (i, j) \in Bellink : bel_i \in Bel \wedge ie_j \in IE$.

Rationale and example The definition of a GRL model (Def. 11) comes with various implicit assumptions in order to form a valid GRL model. We make these assumptions explicit in the definition above.

Let us briefly verify that our previous formalization of Figure 15 satisfies all the constraints of Def. 12:

1. All elements in the formalization have different ids, so this constraint is satisfied.
2. R_{ActIE} contains one element $(0, i)$ for each IE with id i, so this constraint is satisfied as well. Note that other elements (beliefs and links) are not related to actors. This is in line with the jUCMNav implementation.
3. Since we have only one actor with id 0, and this is the only actor that appears in R_{ActIE}, this constraint is satisfied.
4. The contribution links connect elements with ids 3, 4, and 6, which are all IEs.
5. The decomposition links connect elements with ids 5, 6, and 7, which are all IEs.
6. The dependency link connects id 2 with 5, which are both IEs.
7. The belief link connects id 1 with id 4, which are respectively a belief and an IE.

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 two elements (see Figure 5):

– A new element called *generic argument*,
– A new link called *attack link*

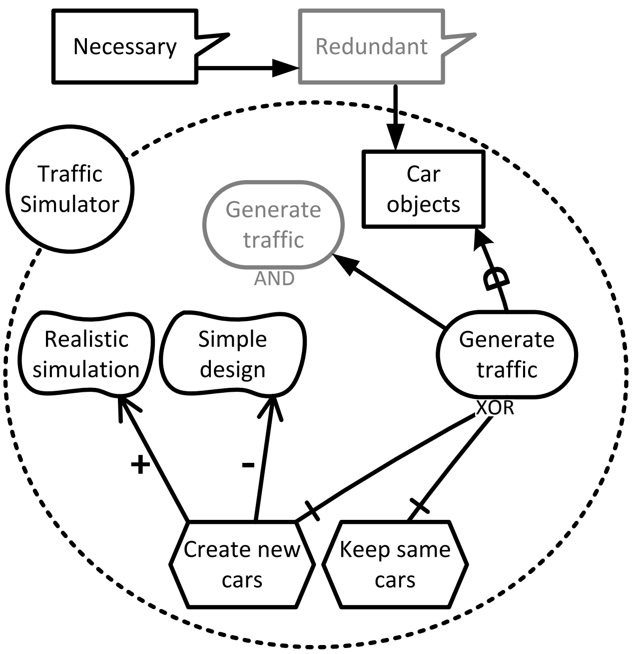

Fig. 16: Example RationalGRL model (extension of Fig. 15)

We illustrate the new elements using Figure 16, which is an extension of Figure 15.

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

Definition 13 (Generic Argument) A generic argument $ga \in \mathbb{N} \times Names$ is a relation such that $ga = (id, name)$ means

- $id \in \mathbb{N}$ is the identifier of the generic argument,
- $name \in Names$ is a string description of its name.

We may refer to the argument with id i simply with ga_i. A set of arguments is denoted by GA.

Rationale and example A generic counterargument is simply an argument that can be used to attack any previous argument, or an IE. It is different from a belief element, since a generic counterargument is used in the computation to determine whether a RationalGRL element is accepted or rejected. Note that one of the constraints of a GRL model (Def. 11) is that GRL links (Def. 8, 9, and 10) 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.

In Fig. 16, there are two generic counterargument which can be formalized as $(19, \text{Redundant})$ and $(20, \text{Necessary})$.

Definition 14 (Argument) An argument $A = (elem, type)$ is a pair such that:

- $elem$ is either an intentional element ie (Def. 2), an actor act (Def. 5), an Actor-IE relation r_{ActIE} (Def. 6), a contribution link $contr$, a decomposition link $decomp$ (Def. 9), a dependency link dep (Def. 10), or a generic argument ga (Def. 13).
- $type \in Type$ (Def. 1) is the type of the argument.

Rationale and example We define an argument as either a generic argument or any of the GRL elements or links. This captures the specification in the RationalGRL metamodel (Fig. 6) in which the class Argument is a superclass of GenericArgument and GRLModelElement, which again is a superclass of both ElementLink and GRLLinkableElement. A GRLLinkableElement is in turn a superclass of Actor and IntentionalElement. In sum, we define an argument simply as any one of the GRL elements or links, or a generic argument.

In Fig. 16, the actor, all IEs, all links, and all generic counterarguments are arguments. We will give a full formalization of this after the definition of a RationalGRL model (Def. 16).

Definition 15 (Attack Link) Given a set of arguments Arg, an attack link $att \in Arg \times Arg$ is a relation such that $att = (A_i, A_j)$ means:

- $A_i \in Arg$ is the argument performing that attack,
- $A_j \in Arg$ is the argument being attacked.

Intuitively, $att = (A_i, A_j)$ means that argument A_i is attacking argument A_j. A set of attack links is denoted by Att.

Rationale and example The attack link is the only link that RationalGRL adds to GRL. In Fig 16, there are three attack links. The first two attack links originate from generic counterarguments and involve the following three elements:

- $A_1 = ((1, Resource, \text{Car objects}, AND), Resource)$
- $A_{19} = ((19, \text{Necessary}), GenArg)$
- $A_{20} = ((20, \text{Redundant}), GenArg)$

These two attack links are formalized as (A_{20}, A_{19}) and (A_{19}, A_1).

The third attack link is an attack link created by *replacing* and existing argument (we will discuss this process in more detail in Sect. 6.4). It involves two version of the same IE:

- $A_4 = ((5, goal, \text{Generate traffic}, AND), Goal)$
- $A_{18} = ((5, goal, \text{Generate traffic}, XOR), Goal)$

This attack link is formalized at (A_{18}, A_4).

The last example shows an important difference between RationalGRL models and valid GRL models: While a valid GRL model disallows multiple elements with the same identifier (Def. 12, condition 1), RationalGRL models do not enforce this restriction. This is because it is possible to create multiple arguments for the same element, where argument contains different content for the same element. This is what the example above also shows. However, the set of *accepted* elements in a RationalGRL should all have unique identifier (see Def. 20).

Definition 16 (RationalGRL Model) A *RationalGRL model* $RatGRL = (Arg, Att)$ consists of a set of arguments $Args$ (Def. 14) and a set of attack links Att (Def. 15).

Rationale and example The definition of a RationalGRL model collects all the previously defined tuples into a single definition. For completeness, we now provide the full specification of Figure 16. Let us first enumerate all the arguments used in this example:

$$
\begin{aligned}
A_0 =& ((0, Actor, \text{Traffic simulator}), Actor)\\
A_1 =& ((1, task, \text{Car objects}, AND), Task),\\
A_2 =& ((2, softgoal, \text{Real. sim.}, AND), Softgoal),\\
A_3 =& ((3, softgoal, \text{Simple des.}, AND), Softgoal),\\
A_4 =& ((4, goal, \text{Generate traffic}, AND), Goal),\\
A_5 =& ((5, task, \text{Create new cars}, AND), Task),\\
A_6 =& ((6, task, \text{Keep same cars}, AND)\}, Task),\\
A_7 =& ((7, 4, 1), Dep),\\
A_8 =& ((8, 5, 2, \text{Help}), Contr),\\
A_9 =& ((9, 5, 3, \text{Hurt}), Contr),\\
A_{10} =& ((10, 5, 4), Decomp),\\
A_{11} =& ((11, 6, 4), Decomp),\\
A_i =& ((0, i-11), ActIE), \text{for } 12 \leq i \leq 17\\
A_{18} =& ((4, goal, \text{Generate traffic}, XOR), Goal),\\
A_{19} =& ((18, \text{Redundant}), GenArg),\\
A_{20} =& ((19, \text{Necessary}), GenArg),
\end{aligned}
$$

This model is then formalized as $RationalGRL = (Arg, Att)$ where:

$$
\begin{aligned}
Arg =& \{A_0, A_1, \ldots, A_{20}\}\\
Att =& \{(A_{18}, A_4), (A_{19}, A_1), (A_{20}, A_{19})\}
\end{aligned}
$$

All the arguments and the attack relations of this RationalGRL model are shown in Figure 17. Note the arguments for Actor-IE containment (arguments A_{12} to A_{20}) have been omitted from this figure for readability. It can be read from the figure that two arguments are currently rejected, namely A_4 and A_{19}. However, we did not yet make precise how exactly this is computed. We will do so in the following definitions.

In order to compute when an argument is accepted and when not we use argumentation semantics. We use the standard approach here, which is known as *Dung's semantics*. The following notions are preliminary.

Definition 17 (Argumentation Framework [9]) An *argumentation framework* $AF = (Arg, Att)$ consists of a set of arguments Arg and a set of attack relations $Att \subseteq Arg \times Arg$.

We see that the definition of an argumentation framework is very close to that of a RationalGRL model. This allows us to use the following results directly.

Definition 18 (Attack, conflict-freeness, defense, and admissibility [9]) 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

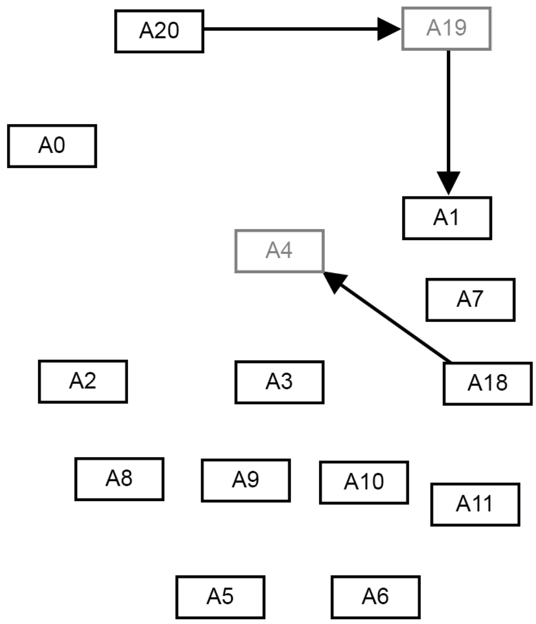

Fig. 17: Argumentation framework of RationalGRL model from Fig. 16

- *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.

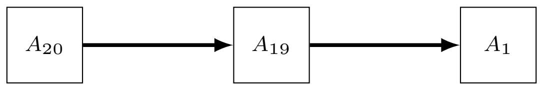

Fig. 18: Example argumentation framework, subset of Fig. 17.

Let us explain these definitions using the example argumentation framework in Figure 18, which is a subset of Figure 17 containing only arguments A_1, A_{19}, and A_{20}. In this example, there are two admissible sets: $\{A_{20}\}$ and $\{A_1, A_{20}\}$. In the last admissible set, A_{20} defends A_1 against its attacker A_{19}. Sets containing both A_{19} and either A_1 or A_{20} are not conflict free, and the sets $\{A_1\}$ and $\{A_{19}\}$ do not defend themselves against A_{19} and A_{20}, 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 focused on the preferred semantics.

Definition 19 (Preferred semantics [9]) A preferred extension of an argumentation framework (Arg, Att) is a maximal (w.r.t. set inclusion) admissible set of R.

In our example from Figure 18, there is one preferred extension, namely $\{A_{20}, A_1\}$. Returning to our running example of Figure 16, this means that the intentional elements Car objects (task) and the generic counerargument *Necessary* are both *accepted*, while the generic counter argument "Redundant" is rejected.

6.3 Translation algorithms

Now that we have formalized both a GRL model and a RationalGRL model, we develop algorithm in order to translate between these two models. Both of these two translation algorithms are straightforward. Their main task is to ensure that in RationalGRL models, the GRL elements and links are wrapped in an argument with the corresponding type. For each algorithm, we provide an explanation an an example of a translation.

GRL to RationalGRL Translation We start with the translation algorithm from GRL to RationalGRL, which is shown in Algorithm 1. The translation algorithm translates all of the GRL elements and links into arguments. It takes a GRL model as input. Recall that a GRL model is defined as $GRL = (IE, Bel, Act, R_{ActIE}, Bellink, Contr, Decomp, Dep)$ (Def. 11). Then, on line 2, the set of arguments is initialized with the empty set. Each of the five for loops in the algorithm translates a different GRL component into an argument. For instance, in the for loop starting at line 3, all IEs are added as pairs consisting of the IE itself, and its corresponding type. This is in line with the definition of an argument 14). Since the other for loops are very similar, we do not explain them here. At line 21, the algorithm returns a RationalGRL model $(Arg, \emptyset)$ containing all arguments in Arg, and an empty set of attack relations.

RationalGRL to GRL translation The translation from a RationalGRL model to a GRL model is given in Algorithm 2. It is very similar to the previous algorithm, but then works in the other direction. In this case, arguments are unpacked and put in the corresponding IE components. The most important difference between the previous algorithm and this one is that in the RationalGRL to GRL translation, argumentation semantics is used to determine the *preferred extension* (Def. 19). This is a set containing all accepted arguments. The GRL is then generated from all the accepted arguments. This is done by iterating over all arguments in the extension in line 5. The switch statement on line 6 then does a case distinction on the type of the arguments, and each case ensures the argument is put in the right GRL component. Finally, the algorithm returns a GRL model on line 18.

Algorithm 1 GRL to RationalGRL Translation

1: **procedure** $ToRationalGRL(GRL)$
2: $\quad Arg \leftarrow \emptyset$
3: $\quad$ **for** $ie \in IE$ **do**
4: $\quad\quad Arg \leftarrow Arg \cup \{(ie, ie.type)\}$
5: $\quad$ **end for**
6: $\quad$ **for** $act \in Act$ **do**
7: $\quad\quad Arg \leftarrow Arg \cup \{(act, Actor)\}$
8: $\quad$ **end for**
9: $\quad$ **for** $actIE \in ActIE$ **do**
10: $\quad\quad Arg \leftarrow Arg \cup \{(actIE, ActIE)\}$
11: $\quad$ **end for**
12: $\quad$ **for** $contr \in Contr$ **do**
13: $\quad\quad Arg \leftarrow Arg \cup \{(contr, Contr)\}$
14: $\quad$ **end for**
15: $\quad$ **for** $dep \in Dep$ **do**
16: $\quad\quad Arg \leftarrow Arg \cup \{(dep, Dep)\}$
17: $\quad$ **end for**
18: $\quad$ **for** $decomp \in Decomp$ **do**
19: $\quad\quad Arg \leftarrow Arg \cup \{(decomp, Decomp)\}$
20: $\quad$ **end for**
21: $\quad$ **return** $(Arg, \emptyset)$
22: **end procedure**

Algorithm 2 RationalGRL to GRL Translation

1: **procedure** $ToGRL(RatGRL)$
2: $\quad E \leftarrow ComputeExtension(Arg, Att)$
3: $\quad IE \leftarrow \emptyset$, $Act \leftarrow \emptyset$, $R_{ActIE} \leftarrow \emptyset$, $Contr \leftarrow \emptyset$
4: $\quad Decomp \leftarrow \emptyset$, $Dep \leftarrow \emptyset$
5: $\quad$ **for** $(Elem, T) \in E$ **do**
6: $\quad\quad$ **switch** T **do**
7: $\quad\quad\quad$ **case** $T \in Type$
8: $\quad\quad\quad\quad IE \leftarrow IE \cup \{Elem\}$
9: $\quad\quad\quad$ **case** $T = Actor$
10: $\quad\quad\quad\quad Act \leftarrow Act \cup \{Elem\}$
11: $\quad\quad\quad$ **case** $T = Contr$
12: $\quad\quad\quad\quad Contr \leftarrow Contr \cup \{Elem\}$
13: $\quad\quad\quad$ **case** $T = Dep$
14: $\quad\quad\quad\quad Dep \leftarrow Dep \cup \{Elem\}$
15: $\quad\quad\quad$ **case** $T = Decomp$
16: $\quad\quad\quad\quad Decomp \leftarrow Decomp \cup \{Elem\}$
17: $\quad$ **end for**
18: $\quad$ **return** $(IE, Act, R_{ActIE}, Contr, Decomp, Dep)$
19: **end procedure**

Valid RationalGRL model While we have defined a notion of a *valid GRL model* (Def. 12), we have not done so for a RationalGRL model yet. This is because it depends on the translation procedure. We define a RationalGRL model as valid if and only if the RationalGRL to GRL translation results in a valid GRL model. In this way, we do not have to reiterate all conditions on a GRL model, but use the translation algorithm.

Definition 20 (Valid RationalGRL Model) A RationalGRL model $RatGRL = (Arg, Att)$ with preferred extension $E = (A_1, \ldots, A_n)$ is a *valid RationalGRL model* iff $TranslateToGRL(RatGRL)$ (Alg. 2) is a valid GRL model (Def. 12).

The following theorem follows directly from the translation algorithms.

Theorem 1 (Translation correctness) *Given some valid GRL model (Def. 12, translating it to RationalGRL and then back to GRL results in the same valid GRL mode, i.e.*

$$ToGRL(ToRationalGRL(GRL)) = GRL.$$

6.4 Algorithms for argument schemes and critical questions

In the previous subsection we formalized 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 (Def. 16). These algorithms are applied to RationalGRL models and produce new arguments and attack relations. We can then use argumentation semantics (Def. **??**) 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.

As discussion in Section 4, ll the argument schemes and critical questions of Table 2 fall into one of the following four categories:

- *INTRO*: These algorithms add one or multiple new arguments to the RationalGRL model, without creating any attack links. The arguments are for new GRL elements or links.
- *DISABLE*: These algorithms add a single argument to the RationalGRL model, which attacks one of multiple existing arguments. The counterargument that is added does not correspond to any GRL element, but instead disables an existing one by attacking all arguments for it.
- *REPLACE*: These algorithms can be seen as a combination of *INTRO* and *DISABLE*. They add a new argument corresponding to a GRL element or link, and this new argument attacks a previous version of the element or link.

The three subsections of this section correspond to these three types of arguments.

In all of the following algorithms, we assume the following

- There algorithms are being applied to some valid RationalGRL model $RatGRL$ (Def. 20),
- The procedure $mindId()$ generates a new unique id.

6.4.1 INTRO algorithms

The following arguments schemes and critical questions of Table 2 fall into this category:

- AS0-AS12
- CQ5b, CQ6b, CQ6c, CQ7b, CQ9, CQ10b

These type of algorithms are short, and consist simply of adding an argument for the element that is being added. Since the structure of these algorithms are all very similar, we do not discuss them all.

Algorithm 3 AS0: a is an actor

```
1: procedure AS_0(n)
2:     actor ← (mintId(), a)
3:     Arg ← Arg ∪ {(actor, Actor)}
4: end procedure
```

Rationale and example Algorithm takes one argument, namely the name of the actor a. On line 2 of the algorithm, a new (unique) id is minted as the identifier of the new actor, which is assigned with its corresponding name to the variable $actor$. On line 3 the pair $(actor, Actor)$ is added as a new argument, indicating that an argument of type $Actor$ is added.

Consider for instance...**TODO for Marc**(by Marc)**: Add example**

Algorithm 4 AS1: Actor with id i has resource R

```
1: procedure AS_1(i, R)
2:     res_id ← mintId()
3:     res ← (res_id, Resource, R, AND)
4:     Arg ← Arg ∪ {(res, Resource), (i, res_id)}
5: end procedure
```

Rationale and example The original argument schemes AS1 is phrased as “Actor a has resource R”. We have slightly reworded this in Algorithm 4 to “Actor with id i has resource R”. The reason is that in GRL, actors have identifiers, so since we assume actor a exists already, we also assume an identifier for it exists. It would be straightforward to introduce a mapping from actor names to identifiers, but we have no done so here for simplicity.

The algorithm itself runs as follows: on line 2, a unique id is assigned to variabel res_id (“resource identifier”). On line 3, an argument for a resource with name R and identifier res_id is assigned to A. Finally, on line 4, two arguments are added to the set of arguments Arg: one for the resource, and one for the containment relations between the actor and the resource (Def. 6.

Since arguments schemes AS2-AS4 are very similar to AS1, we have omitted it here.

Algorithm 5 AS5: Goal with id i decomposes into task t

1: **procedure** $AS_5(i, t)$
2: $task\_id \leftarrow mintId()$
3: $task \leftarrow (task\_id, Task, t, AND)$
4: $Arg \leftarrow Arg \cup \{(task, Task), ((mintId(), i, task\_id), Decomp)\}$
5: **end procedure**

Rationale and example Similar to the previous algorithm, we have slightly reworded critical question AS5. We assume that a goal G exists already with identifier i, and that some new task with name t is a decomposition of G.

In Algorithm 5, on line 2 a unique identifier is created for the task, which is created on line 3. Finally, on line 4, two arguments are added to the set of arguments Arg. The first argument $(task, Task)$ is simply an argument for the task that is added. The second argument $((mintId(), i, task\_id), Decomp)$ is an argument for the decomposition link $(mintId(), i, task\_id)$ (Def. 9, going from the existing goal with identifier i to the new task with identifier $task\_id$.

The remaining argument schemes AS6-AS-12 are all very similar to the previous algorithms and have been omitted here.

The critical questions of type *INTRO* are very similar as well, with one exception: the require an answer. For instance, suppose CQ5b: "Does goal G decompose into other tasks?" is answered with: "Yes, namely into task T". In this case, we simply obtain an instantiation of argument scheme AS5: "Goal G decomposes into task T", which can be executed with Algorithm 5. This is the same for all the other critical questions of type *INTRO* as well. Therefore, we have omitted them here as well.

6.4.2 DISABLE algorithms

As discussed before, algorithms of type *DISABLE* consist of adding a new argument attacking an existing argument, which is an argument for an existing GRL element or link. The argument that is added is itself not an argument for a GRL element or link.

In all of these algorithms, we assume the critical question is answered affirmatively. For instance, for critical question CQ0 "Is the actor relevant?", we assume it is answered with "No" (see right-most column of Table 2).

Rationale and example Algorithm 6 is executed when critical question CQ0 is answered affirmatively (i.e., with "No"). First, on lines 2 and 3, an argument is created for the critical question and added to the set of arguments Arg. Since this argument is not an argument for a GRL element or link, it is formalized as a *generic counterargument* $((mintId(), CQ0), GenArg)$ (Def. 13). The for loop starting at line 4 then iterates over ll arguments $(actor_i, Actor)$, where i is the id of the actor that is no longer relevant. Recall from Def. 3 that we denote with $actor_i$ an actor with identifier i. Thus, this for loop iterates over all arguments for actors with identifier i. The reason why there could be multiple of such actors is that the actor can be refined by an algorithm of type *REPLACE*. We will explain this in more detail in the example below. Then, on line 5, an attack link is created from the generic argument A that is created to the argument for the actor A_i. After executing the algorithm, all existing arguments for the actor with identifier i are attacked by a newly created argument A.

Algorithm 6 CQ0: Is actor with id i relevant? No

1: **procedure** $CQ_0(i)$
2: $A \leftarrow ((mintId(), CQ0), GenArg)$
3: $Arg \leftarrow Arg \cup \{A\}$
4: **for** $A_i \in \{(actor_i, Actor) \in Arg\}$ **do**
5: $Att \leftarrow Att \cup \{(A, A_i)\}$
6: **end for**
7: **end procedure**

Consider for example...**TODO for Marc**(by Marc)**: Add example here**

Critical questions C1-CQ3 are all very similar to CQ0 and have therefore been omitted here.

Algorithm 7 CQ5a: Does the goal with id g_id decompose into task with id t_id? No

1: **procedure** $CQ5a(g\_id, t\_id)$
2: $A \leftarrow ((mintId(), CQ5a), GenArg)$
3: $Arg \leftarrow Arg \cup \{A\}$
4: **for** $A_i \in \{((k, m, n), Decomp) \in Arg \mid$
5: $m = g\_id, n = t\_id\}$ **do**
6: $Att \leftarrow Att \cup \{(A, A_i)\}$
7: **end for**
8: **end procedure**

Rationale and example Algorithm 7 is structurally very similar to Algorithm 6, with the only difference that instead of iterating over actors, we now iterate over decomposition links. This is done in line 5, where we iterate over all decomposition links (k, m, n) (Def. 9) such that m equals the goal identifier g_id, and n equals the task identifier t_id. This means we iterate over all decomposition links from the goal to the task, and we attack all arguments for these links on line 6.

Almost all of the remaining critical questions are similar in structure. CQ11 ("Is the element relevant/useful") is slightly different since the attack element is not of a specific type, but is simply any GRL element. However, the resulting algorithm is very similar to the previous two and has therefore been omitted. The only algorithm of type *DISABLE* that we still discuss is Att.

Algorithm 8 Att: Generic counter-argument on arguments $A_1, \ldots, A_n$

procedure $Att(A_1, \ldots, A_n)$
  $A \leftarrow ((mintId(), Att), GenArg)$
  $Arg \leftarrow Arg \cup \{A\}$
  for $A_i \in \{A_1, \ldots, A_n\}$ **do**
    $Att \leftarrow Att \cup \{(A, A_i)\}$
  end for
end procedure

Rationale and example Algorithm 8 can be regarded as the most general way of providing counter-arguments to arguments. In all of the previous *DISABLE* algorithm, the attack was on a specific type of argument, for instance an argument for an actor or an argument for a decomposition. In this algorithm, however, *any* set of arguments can be attacked by a new argument.

TODO for Marc(by Marc)**: Give example**

6.4.3 REPLACE algorithms

Recall that the *REPLACE* algorithms both attack all arguments for an existing element, and at the same time create a new argument that contains a different version of the element being attacked.

We present two algorithms for replacing elements: the first one replaces the decomposition type (CQ5c, CQ10c), and the second one replaces the name.

Algorithm 9 CQ5c: Is the decomposition type of element ie_i correct? No, it should be X

1: **procedure** $CQ5c(ie_i, X)$
2:   $A \leftarrow ((ie_i.id, ie_i.type, ie_i.name, X), ie_i.type)$
3:   $IEArgs \leftarrow \{(ie_i, ie_i.type) \in Arg\}$
4:   **for** $A_i \in IEArgs$ **do**
5:     $Att \leftarrow Att \cup \{(A, A_i)\}$
6:   **end for**
7:   **for** $\{(A_i, A_j) \in Att \mid A_j \in IEArgs\}$ **do**
8:     $Att \leftarrow (A_i, A)$
9:   **end for**
10:   **for** $\{(A_i, A_j) \in Att \mid A_i \in IEArgs\}$ **do**
11:     $Att \leftarrow (A, A_j)$
12:   **end for**
13:   $Arg \leftarrow Arg \cup \{A\}$
14: **end procedure**

Rationale and example While the original critical question CQ5c is specific to the decomposition between a goal and a task, Algorithm 9 is more generally applicable to any IE, since all IEs have a decomposition type in their definition (Def. 2.

Let us go through this algorithm step by step. On line 2, a new argument $((ie_i.id, ie_i.type, ie_i.name, X), ie_i.type)$ is created. Recall from Def. 3 that we may refer to elements of an IE with the "." notation. Thus, this argument is an argument for the IE that was input to the algorithm, except that the decomposition type is set to X. On line 3, the set $IEArgs$ is assigned with all existing arguments for the input IE. Then, in the first for loop on line 4, we add attack links from the argument that has just been created to all existing arguments for the IE. The next two for loops on respectively lines 7 and 10 ensure that all attack links that existing from and to the previous versions of the IE are also carried over to the new argument A. Finally, one line 13, the new argument is added to the set of arguments.

Let us explain this with an example. **TODO for Marc**(by Marc)**: Add example**

The other *REPLACE* algorithms are very similar to Algorithm 9. It can be used directly for CQ10c, but for CQ12 we should make a small modification. Instead of replacing the decomposition type of the IE, we should replace its name. Since this is a very minor modification we have omitted it here.

7 Discussion

7.1 Related work

Design Rationale Argumentation in software design has for some time now been the subject of the work on so-called *design rationale* (DR) [34], an explicit documentation of the reasons behind decisions made when designing a system or software architecture. DR looks at issues, options and arguments for and against these options in the design of, for example, a software system. Similar to the literature on goal modeling, much of the traditional DR literature provides modeling languages and diagramming tool support for building design rationales. It is in this diagramming functionality that the link with argument diagrams from philosophy, law and AI [32,15] has been made, where argument diagrams represent reasoning from premises to conclusions. More recent work on DR moves away from the idea that all decision have to be explicitly diagrammed and focuses more on empirically investigating how critical reflection can help when designing [31,33], or which parts of the design process are best explicitly documented [11].

Software design and requirements engineering are very closely related [29] and hence the insights from the DR literature are directly applicable to RE. The work on the RationalGRL framework essentially incorporates the core ideas from DR into goal-oriented requirements engineering by explicitly including arguments pro and con the various options into the goal model, and by proposing a methodology and critical questions that encourage reflection when thinking about the possible goals and functionality of a system.

Requirements Engineering There are a number of general approaches in the field of requirements engineering that explicitly take into account arguments. One early example comes from Haley et al. [16], who use formal logical arguments to show that the system behavior satisfies certain security requirements, and more informal arguments to capture and validate the assumptions underlying the system behavior. This system behavior is defined by the tasks it executes and thus arguments are given for and against system tasks, similar to the way beliefs and counterarguments can be provided for tasks in RationalGRL. What Haley et al. leave implicit in their argumentation are the goals of the stakeholders on which the system tasks depend – they include the goals in their framework and mention that there will often be conflicting goals between stakeholders, but do not explicitly model them. Furthermore, the argumentative part of their framework does not include formal semantics for resolving conflict between arguments or determining the acceptability of arguments. Yu et al. [46] further extend the framework by Haley et al., including algorithms for Dung-style [9] argumentation semantics and a database of specific ways in which to attack (or mitigate) risks, which can be likened to a set of critical questions for risks and security requirements (cf. Yu et al. [46] Section 3.1).

Another recent example of the use of arguments in goal-oriented requirements engineering is the work by Murukannaiah et al. [27], who propose Arg-ACH, an approach in which the beliefs underlying conflicting goals can be made explicit using argumentation. Murukannaiah et al. start with the basic technique of Analysis of Competing Hypotheses (ACH), where for conflicting goals the beliefs that are consistent and inconsistent with these goals are included in a matrix and counted. They then extend this technique into Arg-ACH: instead of just indicating whether a belief is consistent or inconsistent with a goal, each belief becomes an argument for or against the goal, which is then diagrammed using the Carneades tool [15]. Belief scores are assigned to arguments, which can be aggregated to provide one's belief in a goal. The arguments for and against goals can be based on argument schemes, and critical questions can be used to find new arguments for or against the goals or the existing arguments. One example provided in the paper is the argument scheme from expert opinion, which allows one to draw conclusions based on expert statements and subsequently question, for example, the objectivity and veracity of the expert using critical questions. Murukannaiah et al. conducted an experiment in which they had two groups, one with ACH and one with Arg-ACH, perform an analysis of several conflicting goals regarding security at transport hubs. They found that, while the group that used Arg-ACH took longer, they also covered more possibilities in their belief search and the conclusions were more consistent among the group.

One other example of argumentation in RE concerns the use of argumentation in requirements elicitation. Ionita et al. [18] propose an simple argumentation dialogue game in which risk assessors try to attack each other's arguments for certain risks attached to a system design. Dung's semantics [9] are then used to determine the risks that are still valid, and those that have been successfully rejected. Elrakaiby et al. [10] use argumentation to explain ambiguity. They define when a statement by a client who is being interviewed about the requirements of a system presents an inconsistency (either with the client's previous statements or with the requirement engineer's beliefs) or an insufficiency, that is, when an analyst needs more information from the client to accept a client's statement. The inconsistencies are then captured as mutually attacking arguments, and the insufficiencies as arguments against the original statements saying that, for example, the functionality expressed in the statement cannot be realised or is irrelevant. Elrakaiby et al. coded the data from 34 requirement elicitation interviews, identifying 39 inconsistencies (i.e. at least two arguments that mutually attack) and 29 insufficiencies (i.e., at least one argument attacking another).

It is clear from this literature that arguments play a core role in RE. Murukannaiah et al. [27] show that critical reflection using argument schemes and critical questions – in the same way that RationalGRL proposes – improves the quality of the reasoning in the RE process. Elrakaiby et al. [10] provide a case study similar to the current one, identifying, as we did, many counterarguments specifically with respect to realisability, relevance and clarity (cf. CQ2a, CQ3, CQ11 and CQ12 in Table 2). Like in RationalGRL, the use of Dung-style argumentation semantics to compute the acceptable claims in RE is further also advocated by the literature [46, 18, 10].

The current work on RationalGRL puts the insights from the above-mentioned literature in a broader framework. For example, there is a specific focus on security requirements [16, 46, 18] or the reasoning is about single goals or tasks instead of about the wider context as represented in a goal model [16, 46, 27, 10]. RationalGRL provides a generic and extensible framework for arguing about goals and tasks in RE. At the moment, there is only a "generic argument" in addition to arguments about goals and tasks. However, new argument schemes and critical questions about for example, security risks or expert opinions, can be easily added: the metamodel (Figure 6) accommodates this and the formal specification in Section 6 is set up in such a way that extending the definition of argument and adding new algorithms for specific critical questions is easy.

Goal Modeling Argumentation has been included – both explicitly and implicitly – in existing goal modeling languages. For example, the belief element in the original GRL specification [2] is meant to capture the rationales (reasons, arguments) behind the inclusion of goals and tasks in the model. Furthermore, relations between elements in a goal model also provide justifications: high-level goals are reasons for lower-level goals, tasks and resources. Hence, refinement and decomposition techniques used in requirements engineering [38] can be seen as explicit argumentation steps in the goal modeling process. Take, for example, CQ2 of PRAS (Section 2.3), which asks whether there are alternative ways of realizing the same goal. Providing an alternative sub-goal or -task in a goal model then an explicit argumentative move in the discussion. So in this sense, a goal model provides a justification for itself, particularly if we include belief elements for extra design rationalization. This idea is also prevalent in our RationalGRL framework: many arguments are in fact GRL elements, and many critical questions can be answered by introducing new GRL elements. However, as was already argued in Section 1 (and also by [19]), argumentation produces different, richer and complementary information to just the goal model. The goal model is the product of a process of argumentation, and does not include, for example, goals and tasks that were at some point considered but discarded. Furthermore, for goal models it is only possible to determine the satisfiability of goals given the possible tasks and resources; what cannot be determined is the acceptability of goals, that is, whether they are acceptable given potentially contradictory opinions of stakeholders.

There are several contributions to the literature that relate argumentation-based techniques with goal modeling. The first line of work is by Bagrheri and Ensan [4] and Mirbel and Villata [25] use abstract Dung argumentation frameworks (see Section 6) to capture individual goals and the relations between them as modeled in a goal model, so that consistent (i.e., acceptable) subsets of goals can be computed using the relevant argumentation semantics. For example, if in the goal model there is a conflict between goals G_1 and G_2, there is an attack between the arguments representing these goals and hence G_1 and G_2 cannot be in the same extension. Or if there is a dependency link between G_1 and G_2, then G_1 should be in every extension G_2 is in and vice versa. AND and OR decomposition are also modeled thus, that is, if G_3 AND-decomposes into G_1 and G_2 then G_1 and G_2 should be in every extension G_3 is in, and if G_3 OR-decomposes into G_1 and G_2 then either G_1 or G_2 should be in every extension G_3 is in.

Modeling goal models as argumentation frameworks allows one to compute the possible consistent sets of goals and tasks given a goal model. In RationalGRL, we have opted not to provide such an argumentation-theoretic semantics to goal models. The reason for this is that GRL already has quite fine-grained satisfiability semantics [2], which take into account conflicts, decompositions and dependencies. RationalGRL focuses on what is not captured in the work discussed above [4, 25], namely the arguments and beliefs underlying a goal model, and the way these arguments and beliefs interact with the elements of the goal model. If desired, however, it would be easy to provide Dung semantics similar to [4, 25] for goal models, as the elements of a goal model are already arguments in RationalGRL (Figure 6).

The contribution most closely related to ours is the work by Jureta *et al.* [19]. Jureta *et al.* propose "Goal Argumentation Method (GAM)" to guide argumentation and justification of modeling choices during the construction of goal models. GAM is a high-level decision process, in which alternative solutions for an RE problem are evaluated and compared using argumentation. Jureta et al. give a simple made-up example of an argumentative discussion regarding a meeting scheduler, in which a goal model is being built by the stakeholders proposing tasks, goals, and alternative solutions for goals. They include clarification as an important step in their GAM process, and discuss various types of clarity problems (vagueness, ambiguity, overgenerality, synonymy) and basic techniques for dealing with them (e.g. thesaurus checks, labeling vague expressions). Jureta et al. then present the argumentation part of their framework, where reasons (justifications) for conclusions are given as formal structured arguments[10]. Arguments and alternatives are then captured as structured arguments or argument diagrams with reasons for or against goals and tasks. Given these arguments the set of undefeated (i.e., acceptable) propositions can be computed to determine which alternative solution to a problem is acceptable. Thus, the arguments and beliefs underlying a goal model and possible alternative modelings are captured as formal arguments. Furthermore, a mapping from goal models to argument diagrams is given, so that it is possible to start arguing about an already existing goal model.

The GAM process is essentially a generic, high-level process for problem solving, not specifically tailored towards goal modeling. In this sense, the RationalGRL methodology in Figure 13 can be seen as a more specific version of GAM explicitly meant for goal modeling. The argument schemes and critical questions in RationalGRL provide more clear handles for goal modeling (cf. requirement 4 of our framework, Section 1).

[10] Informally, a structured argument is similar to the PRAS argument in Section 2.3, i.e., $a, b \xrightarrow{therefore} c$.

In previous work on the RationalGRL framework [40, 42], we effectively extended Jureta et al.'s work and translated argument diagrams to GRL models[11] (an automatic translation tool is discussed in [41]). Thus, we have essentially two complex diagrams, an argument diagram and a goal diagram, and a mapping between them. This 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 has to parse two complex diagrams containing goals, alternative solutions, tasks, and so forth. 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.

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 [22, 30, 35]. 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 [26]. 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

[11] In [19] only a mapping from goal models to structured arguments is given, and the step from structured arguments or argument diagrams to goal models is never formally defined.

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.

Fig. 19: Preferences between arguments

Finally, in this article we have only explored one single argument scheme, but there are many other around. In his famous book "Argumentation schemes", Walton describes a total of 96 schemes. Murukannaiah *et al.* [27] 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.

One idea is to capture requirements engineering and software design processes as explicit dialogues between parties [12]. Software design discussions Black et al., Prakken and Wierenga.

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:

$$
\begin{aligned}
&(softgoal(i) \vee goal(i) \vee task(i) \\
&\qquad \vee resource(i) \rightarrow IE(i).
\end{aligned}
$$

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

$$
\begin{aligned}
&contrib(k, i, j, ctype) \rightarrow \\
&(IE(i) \wedge IE(j) \wedge IE(j).
\end{aligned}
$$

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.

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. E. Bagheri and F. Ensan. Consolidating multiple requirement specifications through argumentation. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 659–666. ACM, 2011.
5. T. J. Bench-Capon and P. E. Dunne. Argumentation in Artificial Intelligence. *Artificial intelligence*, 171(10-15):619–641, 2007.
6. 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.
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. 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.
10. Y. Elrakaiby, A. Ferrari, P. Spoletini, S. Gnesi, and B. Nuseibeh. Using argumentation to explain ambiguity in requirements elicitation interviews. In *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE 2017)*, pages 51–60. IEEE Press, 2017.
11. D. Falessi, L. C. Briand, G. Cantone, R. Capilla, and P. Kruchten. The value of design rationale information. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(3):21, 2013.
12. A. Finkelstein and H. Fuks. Multiparty specification. In *ACM SIGSOFT Software Engineering Notes*, volume 14, pages 185–195. ACM, 1989.
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. T. F. Gordon. Visualizing carneades argument graphs. *Law, Probability & Risk*, 6(1-4):109–117, 2007.
16. C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.
17. 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.
18. D. Ionita, J.-W. Bullee, and R. J. Wieringa. Argumentation-based security requirements elicitation: The next round. In *IEEE 1st workshop on Evolving Security and Privacy Requirements Engineering (ESPRE 2014)*, pages 7–12. IEEE, 2014.
19. 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.
20. 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.
21. S. D. kenniscentrum. DEMO: The KLM case. `http://www.demo.nl/attachments/article/21/080610_Klantcase_KLM.pdf`, Last accessed on 22 August, 2012.
22. M. Lankhorst. *Enterprise architecture at work: modelling, communication, and analysis*. Springer, 2005.
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. I. Mirbel and S. Villata. Enhancing goal-based requirements consistency: An argumentation-based approach. In *Computational Logic in Multi-Agent Systems - 13th International Workshop, CLIMA XIII, Montpellier, France, August 27-28, 2012. Proceedings*, pages 110–127, 2012.
26. S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9):901–934, 2009.
27. 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.
28. G. Mussbacher and D. Amyot. Goal and scenario modeling, analysis, and transformation with jUCMNav. In *ICSE Companion*, pages 431–432, 2009.

29. B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.
30. 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.
31. M. Razavian, A. Tang, R. Capilla, and P. Lago. In two minds: how reflections influence software design thinking. *Journal of Software: Evolution and Process*, 28(6):394–426, 2016.
32. C. Reed and G. Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools*, 13(04):961–979, 2004.
33. 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.
34. 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.
35. The Open Group. *The Open Group – TOGAF Version 9*. Van Haren Publishing, Zaltbommel, The Netherlands, 2009.
36. 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.
37. UCI. Design Prompt: Traffic Signal Simulator. `http://www.ics.uci.edu/design-workshop/files/UCI_Design_Workshop_Prompt.pdf`. Accessed: 2016-12-27.
38. A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–262, 2001.
39. 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.
40. M. van Zee, F. Bex, and S. Ghanavati. Rationalization of Goal Models in GRL using Formal Argumentation. In *Proceedings of the 23rd International Requirements Engineering Conference (RE'15)*, pages 220–225. IEEE Press, August 2015.
41. M. van Zee, D. Marosin, F. Bex, and S. Ghanavati. The RationalGRL toolset for Goal Models and Argument Diagrams. In *Computational Models of Argument. Proceedings of COMMA 2016*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 479–480. IOS Press, September 2016.
42. 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)*, pages 553–560, November 2016.
43. D. Walton, C. Reed, and F. Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.
44. D. N. Walton. *Practical reasoning: goal-driven, knowledge-based, action-guiding argumentation*. Rowman & Littlefield, 1990.
45. 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*, pages 226–235, 1997.
46. Y. Yu, V. N. Franqueira, T. T. Tun, R. J. Wieringa, and B. Nuseibeh. Automated analysis of security requirements through risk-based argumentation. *Journal of systems and software*, 106:102–116, 2015.

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

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? **[22 answer to 21]** Not useful, because according to the specification all intersections have traffic lights. [DISABLE]
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	
0:15:51 (P2)	They need-	

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?
0:24:12 (P2)	So setting it per road, I think is something we want	**[33 answer to 32]** No, “Set car influx” becomes “Set car influx per road” [REPLACE]

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)