Requirements Engineering

RationalGRL: A Framework for Argumentation and Goal Modeling

--Manuscript Draft--

Manuscript Number:	REEN-D-17-00086
Full Title:	RationalGRL: A Framework for Argumentation and Goal Modeling
Article Type:	Original Research
Keywords:	Goal Modeling; Argumentation; Goal-oriented requirements engineering; Practical Reasoning
Corresponding Author:	Marc van Zee Google Research Zurich SWITZERLAND
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Google Research Zurich
Corresponding Author's Secondary Institution:	
First Author:	Marc van Zee
First Author Secondary Information:	
Order of Authors:	Marc van Zee
	Floris Bex
	Sepideh Ghanavati
Order of Authors Secondary Information:	
Funding Information:	

Noname manuscript No.
(will be inserted by the editor)

RationalGRL: A Framework for Argumentation and Goal Modeling

Marc van Zee · Floris J. Bex · Sepideh Ghanavati

the date of receipt and acceptance should be inserted later

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 back 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 modeling 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.

Marc van Zee
Google Research, Zürich, Switzerland
E-mail: marcvanzee@gmail.com

Floris Bex
Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
E-mail: f.j.bex@uu.nl

Sepideh Ghanavati
Department of Computer Science, Texas Tech University, Lubbock TX, USA
E-mail: sepideh.ghanavati@ttu.edu

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

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 "early-phase" requirements engineering activities are 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 [43]. 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 [23,37]). Several goal modeling languages have been developed in the last two decades as well. The most popular ones include $i*$ [43], Keep All Objects Satisfied (KAOS) [38], the NFR framework [7], TROPOS [17], the Business Intelligence Model (BIM) [20], and the Goal-oriented Requirements Language (GRL) [1].

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. Developing goal models for such complex and large systems is not a trivial task and can be very cumbersome. 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 the system development process.

The first challenge for the goal modeling phase, particularly in large projects, is related to its dynamic nature: goal models continuously change and evolve. Stakeholders' preferences are rarely absolute, relevant, stable, or consistent [25]. Stakeholders may change their opinions about a modeling decision in between two modeling sessions, which may require revisions of a goal model. If the rationales behind these revisions are not properly documented, alternative ideas and opposing views that could potentially lead to

Manuscript

Noname manuscript No.
(will be inserted by the editor)

RationalGRL: A Framework for Argumentation and Goal Modeling

Marc van Zee · Floris J. Bex · Sepideh Ghanavati

the date of receipt and acceptance should be inserted later

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 back 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 modeling 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.

Marc van Zee
Google Research, Zürich, Switzerland
E-mail: marcvanzee@gmail.com

Floris Bex
Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
E-mail: f.j.bex@uu.nl

Sepideh Ghanavati
Department of Computer Science, Texas Tech University, Lubbock TX, USA
E-mail: sepideh.ghanavati@ttu.edu

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

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 "early-phase" requirements engineering activities are 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 [43]. 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 [23,37]). Several goal modeling languages have been developed in the last two decades as well. The most popular ones include $i*$ [43], Keep All Objects Satisfied (KAOS) [38], the NFR framework [7], TROPOS [17], the Business Intelligence Model (BIM) [20], and the Goal-oriented Requirements Language (GRL) [1].

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. Developing goal models for such complex and large systems is not a trivial task and can be very cumbersome. 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 the system development process.

The first challenge for the goal modeling phase, particularly in large projects, is related to its dynamic nature: goal models continuously change and evolve. Stakeholders' preferences are rarely absolute, relevant, stable, or consistent [25]. Stakeholders may change their opinions about a modeling decision in between two modeling sessions, which may require revisions of a goal model. If the rationales behind these revisions are not properly documented, alternative ideas and opposing views that could potentially lead to different goal models might be 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 an input in a later RE or development phase. If the 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 other stakeholders may end up having similar discussions throughout the design and development phase as well.

Another challenge is that current goal modeling languages have limited support for reasoning about changing beliefs and opinions, and their effects on the goal model. A stakeholder may change his or her opinion, but it is not always directly clear what the effect of this is on a goal model. Similarly, with existing goal modeling languages one might change a part of a goal model without being able to reason about whether or not this new goal model is consistent with the underlying beliefs and arguments. This becomes even more challenging 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 goal models and their underlying beliefs and opinions may further lead to a poor understanding of the problem and its solution, which is an important reason of RE project failure [8].

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

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

Following from our previous work [39,40], we develop the *RationalGRL* framework, which extends an existing approach for goal modeling, the Goal-oriented Requirements Language (GRL) [1], with techniques from argumentation theory, namely *argument schemes* (or argumentation schemes [41]) and argumentation semantics [10]. Argument schemes are reusable patterns of reasoning that capture the typical ways in which humans argue and reason. Argument schemes are associated with *critical questions*, which can point to typical sources of doubt or implicit assumptions people make when arguing in a certain way. These critical questions can point to counterarguments, and the acceptability of sets of arguments can be computed using argumentation semantics. Thus computational argumentation based on schemes can guide users in systematically deriving conclusions and making assumptions explicit [5,29].

Inspired by the work on practical reasoning from Artificial Intelligence, most notably Atkinson and Bench-Capon [2], we have developed a list of argument schemes that can be used to analyze and guide stakeholders' discussions about goal models. This list of argument schemes is based on 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.

In order to specify clearly in what way RationalGRL extends GRL, we develop a metamodel, which specifies the traceability links between the arguments based on the schemes and the GRL models. In addition to this metamodel, we provide formal semantics for RationalGRL by formalizing the GRL language in propositional logic and rendering arguments about a GRL model as a formal argumentation framework to which we can apply argumentation semantics [10]. We, then, formally capture the links between argumentation and goal modeling as a set of algorithms for applying argument schemes and critical questions about goal models.

In order to support practitioners in using the RationalGRL framework, we propose a methodology, which consists of developing goal models and posing arguments based on schemes in an integrated way. We implement a web-based tool for the RationalGRL framework as well, which works on any modern browser and is developed in Javascript.[1] The tool is open-source, and designed such that new argument schemes and critical questions can be added easily.

The rest of this article is organized as follows. Section 2 introduces our running example and briefly discusses the basics of GRL and argumentation about goals using argument schemes. Section 3 contains the case study and an explanation of how we obtained an initial set of argument schemes and critical questions by coding transcripts from discussions about an information system. Section 4 provides

[1] The tool and source code can be found at `http://www.rationalgrl.com`.

an overview of the RationalGRL framework and the RationalGRL metamodel, and examples from the case study that illustrate the framework and its language. In Section 5, we provide formal semantics for GRL and RationalGRL, show how RationalGRL models can be translated to GRL models and vice versa, and we develop various algorithms that change a RationalGRL model according to an argument scheme or a critical question. Section 6 discusses the RationalGRL methodology and explains various features of the tool we developed. Finally, Section 7 covers the related work, the future work, and the conclusion.

2 Background: Goal-oriented Requirements Language and Practical Reasoning

In this section, we first introduce our running example, after which we give a brief overview of the Goal-oriented Requirements Language (GRL) [1], which is the goal modeling language we use to integrate with the argumentation framework[2]. Next, we introduce argumentation; we discuss the *practical reasoning argument scheme (PRAS)* [2], an argument scheme that is used to form arguments and counterarguments about situations involving goals, and we give informal examples of how argument and counterargument can influence the status of beliefs about goals.

2.1 Running example: Traffic Simulator

Our examples and case study are based on the data produced by a recent series of experiments by Schriek et al. [34], who in turn base their work on the so-called Irvine experiment [36]. This experiment contains a well-known design reasoning assignment in software engineering about a traffic simulator. 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 et al. asked designers (groups of students) to discuss the requirements of this traffic simulator. These discussions were recorded and transcribed. We used these transcripts for an extensive case study on the basis of which we develop our RationalGRL framework. Furthermore, we also use the traffic simulator case as running examples throughout this paper.

[2] Note that in this article, we consider a slightly simplified version of GRL.

2.2 Goal-oriented Requirements Language (GRL)

GRL is a modeling language for specifying intentions, business goals, and non-functional requirements of multiple stakeholders [1]. 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.

Figure 2 illustrates a simplified GRL diagram from the traffic simulator design exercise. An actor represents a stakeholder of a system or the system itself (*Traffic Simulator*, Figure 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. 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, there are two tasks *Create new cars* and *Keep same cars*: in order to achieve the goal *Generate cars*, the simulation can either constantly generate new ones or keep the same cars and have them reappear after they disappear off-screen. In order to be achieved or completed, softgoals, goals, and tasks may require resources to be available (e.g., *Car Objects*). Finally, the full version of GRL allows design rationale to be captured using beliefs. Since we capture the reasoning and rationales behind goal models using arguments, we do not include beliefs in our simplified version of GRL (but see the discussion at the end of this section).

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 softgoal *Realistic simulation*, and a negative contribution to the softgoal *Simple design*. Note that the full GRL specification considers different levels of positive and negative contribution values (both quantitative and qualitative), which are not directly relevant for current purposes. Dependency links are relationships between IEs, which can model dependencies

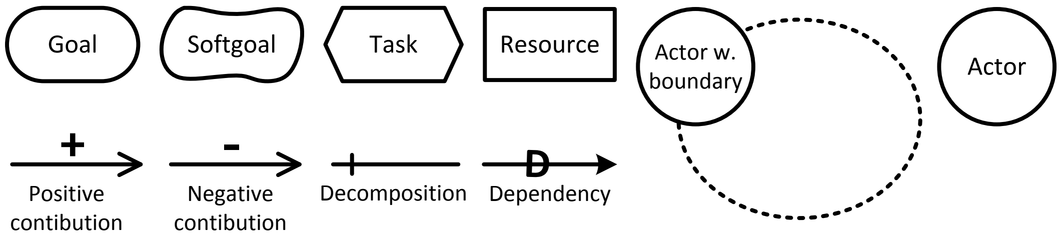


Fig. 1: Basic elements and relationships of GRL

between actors. Here, the goal *Generate cars* depends on the resource *Car objects*.

GRL is based on $i*$ [43] and the NFR Framework [7], but it is less restrictive. Intentional elements and links can be more freely combined, the notion of agents is replaced with the more general notion of actors 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 a scalable and consistent representation of multiple views/diagrams of the same goal model (see [15, Ch.2] for more details). 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. For example, GRL is linked to Use Case Maps, which provides 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.

The GRL model in Figure 2 shows the softgoals, goals, tasks and the relationship between the different intentional

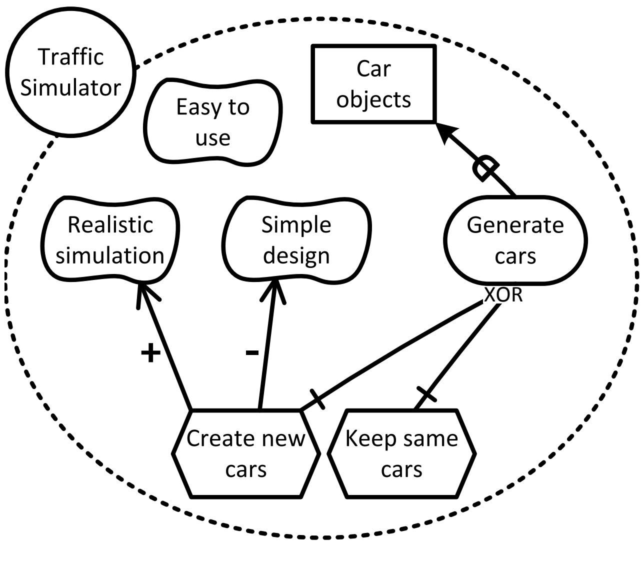


Fig. 2: Partial GRL Model of the traffic simulator example

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 the 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 types 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 model, such as 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 questions given above. Finally, the jUCMNav tool does not consider beliefs in the evaluation algorithms, and thus there is no formal connection between goal models and their underlying beliefs and opinions. For a more detailed comparison of our framework with GRL beliefs, see Section 7.1.

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 [42]. 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 counter-argument to the original argument for the action.

Atkinson and Bench-Capon [2] develop and formalize the *Practical Reasoning Argument Scheme* (PRAS). A simplified version of this argument scheme is as follows:

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

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

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

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

In argumentation, conclusions which are at one point acceptable can later be rejected because of new information. For example, we may argue that, in fact, performing action *Keep same cars* does not realize goal *Generate cars*, thus giving a counterargument to the above instantiation of PRAS. Atkinson et al. [2] define a set of so-called critical questions that point to typical ways in which an argument based on PRAS can be criticized. 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

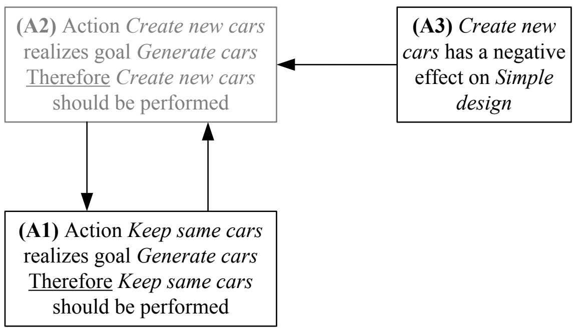

(A2) Action *Create new cars* realizes goal *Generate cars* Therefore *Create new cars* should be performed
(A3) *Create new cars* has a negative effect on *Simple design*
(A1) Action *Keep same cars* realizes goal *Generate cars* Therefore *Keep same cars* should be performed

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

between these arguments, we can compute which arguments are accepted and which are rejected using different argumentation semantics [10][3]. Figure 3 shows three arguments from the traffic simulation example, where arguments are rendered as boxes and attack relations as arrows. There are two arguments based on PRAS: argument A1 for *Keep Same Cars* and argument A2 for *Create new cars*. Argument A2 proposes an alternative way of realizing the same goal *Generate cars* with respect to argument A1 and vice versa (cf. CQ2), so A1 and A2 mutually attack each other, denoted by the arrows between A1 and A2. Argument A3 says that *Create new cars* has a negative effect on *Static Simulation*, so A3 attacks A2, as it points to a negative side-effect of *Create new cars* (CQ3). The intuition here is that an argument is acceptable if any argument that attacks it is itself rejected. In Figure 3, argument A3 is accepted because it has no attackers. This makes A2 rejected (indicated by the lighter grey color), because its attacker A3 is accepted. A1 is then also accepted, since its only attacker, A2, is rejected.

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. In fact, in some of our previous work on RationalGRL [39,40] we used PRAS-arguments such as the ones above to capture reasoning about goals, and provided a translation to goal models. However, one problem of this approach is that we cannot literally use PRAS and its critical questions, as there are elements in 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 [2] 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 typi-

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

cal discussion is much more unstructured, as is clear from the transcript excerpts in Appendix B. Thus, if we would use the version of PRAS presented in this section for our argumentation, we would violate requirement 1: The argumentation techniques must capture the actual discussions of the stakeholders or designers in the early requirements engineering phase. Our solution is 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. To get a sense of such discussions, we performed a case study to examine which types of discourse are used during discussions of system requirements, and how these discourse types can be captured as argument schemes and critical questions. We manually coded transcripts of such discussions using 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, which can be found at:

`http://www.rationalgrl.com`

In order to obtain actual requirements discussions, we turned to a recent series of experiments by Schriek et al. [34]. In these experiments, 12 groups of two or three students in a Software Architecture course at MSc level were given the traffic simulator assignment (Appendix A). 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. They 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, came up with an initial list of 11 argument schemes (AS0-AS10 in Table 1), representing *claims* about the goal model containing the requirements of the system. We used no particular method to develop this initial list besides our own intuition about which part of a goal model would be likely to be discussed.

AS0 to AS4 are schemes that concern a single element of the goal model. For example, AS0 represents the claim 'a is a relevant actor for the system', and AS3 represents the claim 'G is a goal for the system'. AS5 to AS10 are claims

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**
Gen	Generic counterargument	0	2	2	**4**
TOTAL		69	80	69	**222**

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

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 [2]. CQ2 to CQ12 (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 'We also have to be able to change the inflow of cars. How many car come out in here on the side' (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, CQ11 ('Is the element relevant/useful?') is not found directly in the transcripts, but it can be inferred from statements such as '...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, since there were many discussions on the relevance and the clarity of the names of elements, two generic critical questions CQ12 and CQ13 were added. The final results of the coding can be found in Table 1. We found a total of 159 instantiations of argument schemes AS0-AS10. 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, Gen).

Our coding further led us to identify three different operations, that represent different effects an argument or critical question can have on a goal model: an argument can introduce a new element in the goal model (INTRO); it can disable (i.e. attack) a goal model element (DISABLE); or it can replace an element in the goal model with another one (REPLACE). Consider, for example, Table 3 in Appendix B. First, an argument is posed that introduces a number of tasks. A counterargument is then given against one of these tasks (*add traffic light*), which is 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 these operations in Section 4 with various examples, and w formalize the operations in Section 5 with algorithms.

4 The RationalGRL Framework

We now give an overview of our RationalGRL framework. Through a metamodel and informal examples from our case study we will show 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 formalization of our framework using formal logic can be found in Section 5.

Figure 4 presents an overview of the RationalGRL framework. There are two activities (bottom), *Practical reasoning & argumentation* and *Goal model construction*, which give rise to two different models (top), a *RationalGRL model* and a *GRL model*. A RationalGRL model is a model in the language that we will explain in the next subsection, while a GRL model is a model in the language we described in Section 2.2. The *Goal model construction* part of the RationalGRL framework allows for the creation of goal models by analyzing the non-functional requirements and refining

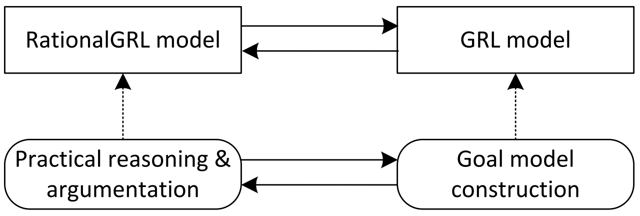


Fig. 4: The RationalGRL Framework

the high-level goals into operationalized tasks. In the *Practical reasoning & argumentation* part, arguments and counterarguments can be put forward about various parts of the 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. Thus, in the framework it is possible to trace a goal model back to the original discussions about goals, tasks and requirements. The GRL model is shown on the right-hand side of the framework while the argumentation model is on the left-hand side. The links between the two sides illustrate the impacts and relationships between two sides. Note that answer to critical questions and argument schemes that are instantiated during the analysis phase of the GRL model are documented with the GRL model and can be referred to in the future.

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 discuss the language for RationalGRL models and we provide a metamodel, linking the new concepts of our language to GRL. In Section 4.3, we provide extensive examples from our case study.

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 discussions of stakeholders (**requirement 1**). 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 the set of argument schemes and critical questions for RationalGRL into the list shown in Table 2.

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 the CQs (right column). The effects are for the CQs only, and do not apply to the argument schemes.

It is important to note that the list we provide here is not exhaustive. It is merely the result of our empirical study, but new argument schemes and questions can be added depending on the problem domain. In fact, our tool (Section 6.2) contains many more critical questions, and it is fully extensible, meaning that new argument schemes and critical questions can be added easily. See Section 7.2 for more details.

Schemes AS0-AS4 and AS11-AS12 are arguments for an element of a goal model, and AS5-AS10 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 the schemes in Table 2 can be used to form arguments about elements in a GRL 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 and in Section 5.

4.2 The RationalGRL Modeling Language

RationalGRL is an extension of GRL and includes all the 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 a GRL element is attacked by an argument that itself is not attacked, then this GRL element will be rejected. Note Figure 5 only shows one type of disabled element, a Task, but all the elements (IEs and links) can be disabled in RationalGRL.
- *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 the GRL element.

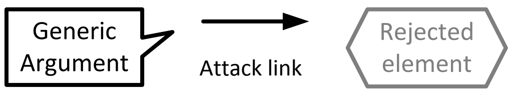


Fig. 5: The new elements and link of RationalGRL. Generic argument (left), attack link (middle), and disabled element (right). The disabled element in this figure is a task, but all IEs and links can be disabled in RationalGRL

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.

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 is simply the GRL metamodel. It 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 at most 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 for an AttackLink to 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.

The OperationTypes in the Argumentation package correspond to the operations we informally introduced in Section 4. These operations are performed by instantiating an argument scheme or answering a critical question in a certain way. An INTRO operation introduces a new RationalGRL element. A DISABLE operation creates a new argument that attacks another argument or GRL element, effectively disabling it. The REPLACE operation replaces a RationalGRL element with a new element. Instantiating an argument scheme from Table 2 always leads to an INTRO operation, that is, it always introduces a new RationalGRL element. Answering a critical question can have different effects depending on the critical question and the answer. Table 2 shows these effects for the different critical questions and answers. For example, answering CQ0 with "no" disables the argument based on AS0.

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

4.3 From Practical Reasoning to RationalGRL Models: Examples from the Case Study

We now discuss the interactions between the *practical reasoning & argumentation* (i.e. the bottom left element of the framework in Figure 4) on the one hand, and *RationalGRL models* (i.e. the top left element of the framework in Figure 4) on the other hand. We provide informal examples of the links between the practical reasoning found in our case study transcripts and RationalGRL models. The formal grounding for the connection between practical reasoning and RationalGRL models can be found in the RationalGRL metamodel (Section 4.2) and in more detail in the logical formalization in the next section. The connection between the RationalGRL models shown in this section and regular GRL models is further formally defined in Section 5.3.

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 excerpt from transcript t_3 shown in Table 5 in Appendix B. On the left side of the image, the arguments found in the transcript are shown, together with the argument scheme they are based on. The participants in the discussion argue that actor *System* has a goal and two tasks, and that the goal AND-decomposes into the two tasks. By putting forward these arguments, 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.

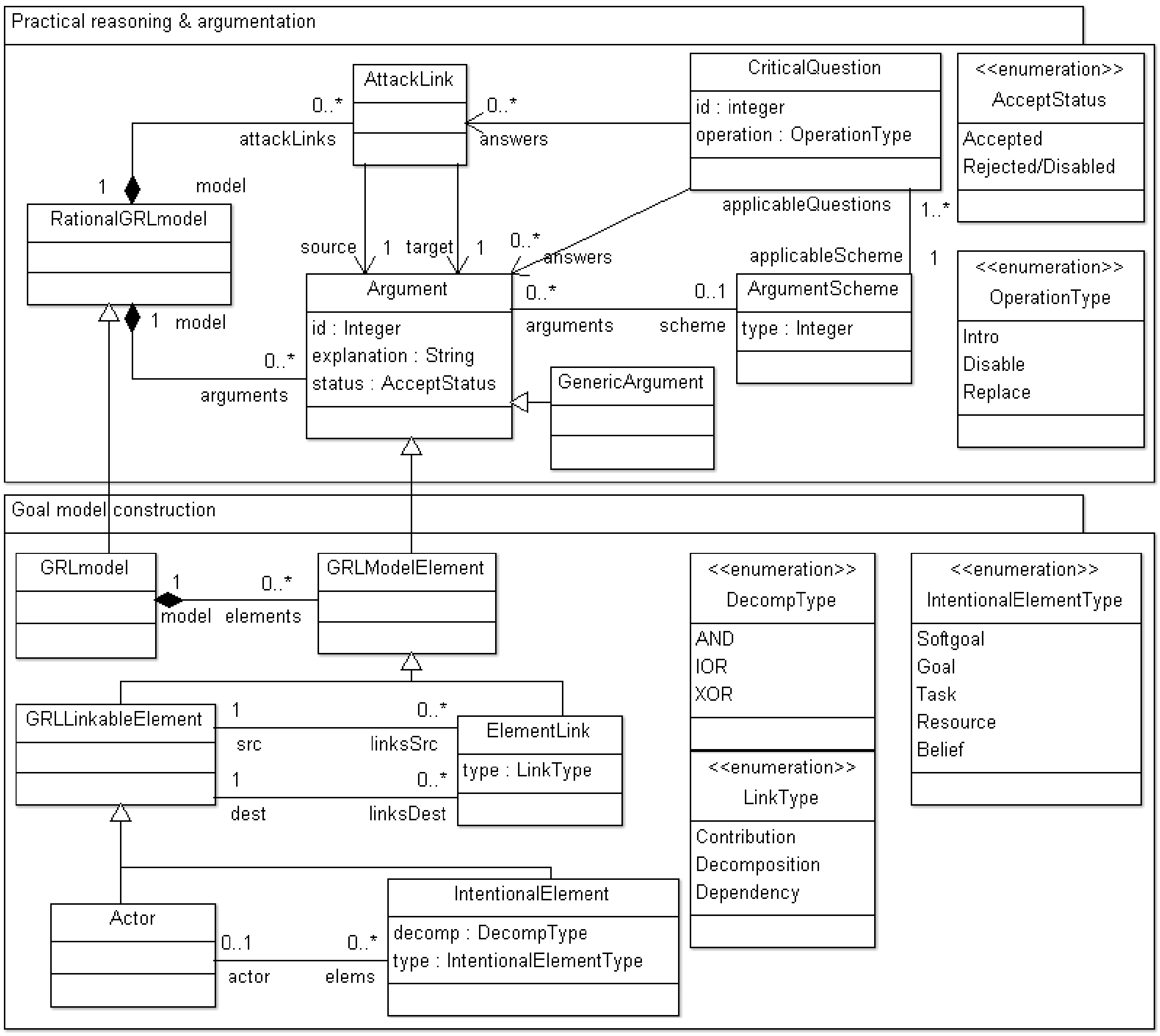


Fig. 6: The RationalGRL metamodel. The *Goal model construction* package (bottom) is the GRL metamodel. The *Practical reasoning & argumentation* package is the RationalGRL extension

Example 2: Disabling GRL elements by answering critical questions (DISABLE) The excerpt from transcript t_1 used in this example is shown in Table 3 in Appendix B. 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 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 argument scheme 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 5.2), indicated by 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 excerpt of transcript t_3 used for this example is shown in Table 5 in Appendix B. It consists of a discussion about the type of decomposition

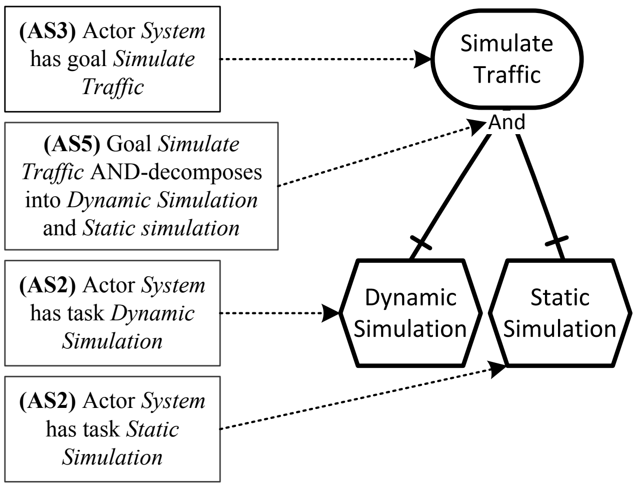

(AS3) Actor System has goal Simulate Traffic
Simulate Traffic
(AS5) Goal Simulate Traffic AND-decomposes into Dynamic Simulation and Static simulation
And
(AS2) Actor System has task Dynamic Simulation
Dynamic Simulation
Static Simulation
(AS2) Actor System has task Static Simulation

Fig. 7: RationalGRL model (right) with instantiated argument schemes (left): Introducing new elements (operation INTRO)

relationship for the goal *Simulate Traffic* (Figure 9). Recall that in Example 1, an AND-decomposition was introduced for this goal with AS5. In the discussion of the current example, CQ5c – "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.

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*" (Figure 10). 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*.

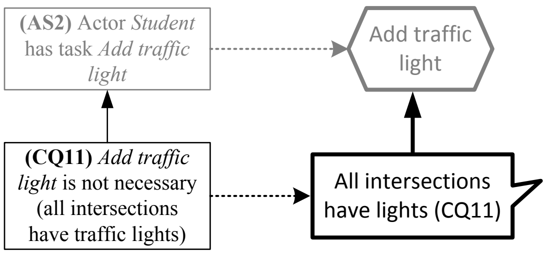

(AS2) Actor Student has task Add traffic light
Add traffic light
(CQ11) Add traffic light is not necessary (all intersections have traffic lights)
All intersections have lights (CQ11)

Fig. 8: Disabling elements (operation DISABLE).

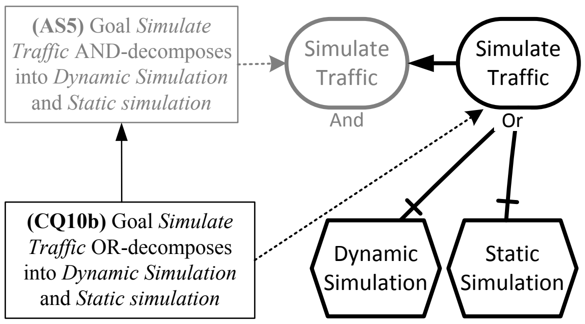

(AS5) Goal Simulate Traffic AND-decomposes into Dynamic Simulation and Static simulation
Simulate Traffic
And
Simulate Traffic
Or
(CQ10b) Goal Simulate Traffic OR-decomposes into Dynamic Simulation and Static simulation
Dynamic Simulation
Static Simulation

Fig. 9: Replacing an element (operation REPLACE).

Example 5: Defending the addition of an actor (DISABLE) The excerpt from transcript t_3 used in this example is shown in Table 6 in Appendix B. Actor *Development Team* is introduced with an argument based on AS0 (Figure 11). This is then attacked by arguing that the professor will develop the software, so there will not be any development team (CQ0).

Further in the discussion, it is then argued that the development team should be considered, since the professor does not develop the software. This is captured using a generic counterargument *Att*, 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 5.2), which causes the actor in the RationalGRL model to be enabled again.

5 RationalGRL: Logical Framework

In Section 4, we have shown 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 formal rendition of this traceability will be presented in this section, in which we present a logical formalization of

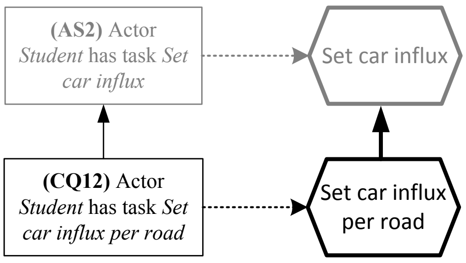

(AS2) Actor Student has task Set car influx
Set car influx
(CQ12) Actor Student has task Set car influx per road
Set car influx per road

Fig. 10: Renaming an element name (operation REPLACE).

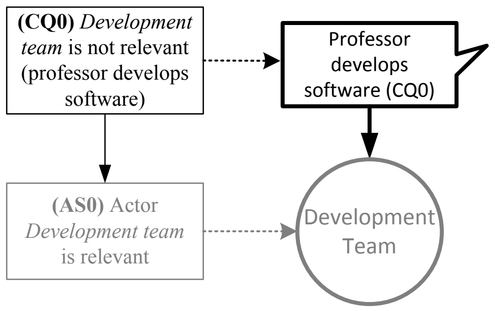

(CQ0) Development team is not relevant (professor develops software)
Professor develops software (CQ0)
(AS0) Actor Development team is relevant
Development Team

Fig. 11: Disabling an element (operation DISABLE).

RationalGRL. This is done for multiple reasons: (i) Most approaches in formal argumentation (cf. Section 2.3) 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 to valid GRL models and vice versa in a precise way, and (iv) the formal approach is a basis for automating the framework in terms of tool support, which we present in the next section.

In Sections 5.1 and 5.2 we formalize a static representation of our framework based on the metamodel (Figure 6). We first provide a formal specification of a GRL model in Section 5.1, and we extend this with arguments and attack links in the Section 5.2, hereby obtaining a formal specification of a RationalGRL model. In Section 5.3 we then present algorithms in order to translate a GRL model into a RationalGRL model, and vice versa. Finally, in Section 5.4 we develop algorithms for instantiating argument schemes and answering critical questions and thus formally capture the INTRO, REPLACE and DISABLE operations of which examples were previously given in Section 4.3.

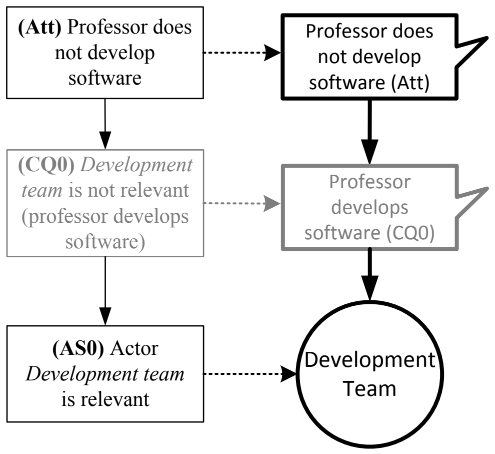

(Att) Professor does not develop software
Professor does not develop software (Att)
(CQ0) Development team is not relevant (professor develops software)
Professor develops software (CQ0)
(AS0) Actor Development team is relevant
Development Team

Fig. 12: Defending an element by attacking its disabling attacker (operation DISABLE).

5.1 Formal Specification of GRL

We formalize a GRL model based on the metamodel (Figure 6), starting with intentional elements and actors.

Throughout this section, we adopt the convention that variables start with a lowercase letter (e.g, id, i, j, $name$, $goal$), and sets and constants start with an uppercase letter (e.g., $Type$, AND, $Goal$). We begin by defining general elements of our language that we use in subsequent definitions.

Definition 1 (General definitions) We define the following sets:

- $IETypes = \{Softgoal, Goal, Task, Resource\}$,
- $LinkTypes = \{PosContr, NegContr, Dep, Decomp\}$[4],
- $Types = IETypes \cup LinkTypes \cup \{Actor, ActIE, GenArg\}$,
- $Names$ is a finite set of strings,
- $DecompTypes = \{AND, OR, XOR\}$.

Next, we define an intentional element.

Definition 2 (Intentional Element) An intentional element is a tuple $ie = (id, type, name, decomptype)$, where:

- $id \in \mathbb{N}$ 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,
- $decomptype \in DecompTypes$ refers to the type of decomposition.

A set of intentional elements is denoted by IE.

The definition above is sufficient to capture all intentional elements (IEs) used in GRL. Note that according to the definition above, all IEs have a decomposition type, even when the IE is not decomposed into other IEs. This is in line with the GRL metamodel. However, in the following we sometimes leave out the $decomptype$ if the IE is not decomposed to simplify notation and improve readability.

Throughout this section we use the example in Figure 2. This example contains various IEs which can be formalized using Definition 2, for example, $(1, Resource,$ Car objects$)$, $(2, Softgoal,$ Realistic simulation$)$ and $(5, Goal,$ Generate cars$, XOR)$.

We now define actors.

Definition 3 (Actor) An actor is a tuple $act = (id, type, name)$, where:

- $id \in \mathbb{N}$ is the identifier of the actor,
- $type = Actor$ states that this tuple is an actor.

[4] Recall from Section 2 that for contribution links we only distinguish between positive and negative contributions. Extending the formalization model to include all the GRL values is straightforward but has been omitted here for conciseness.

– $name \in Names$ is a string description of its name.

A set of actors is denoted by Act.

We can formalize the actor in our example (Figure 2) as $(0, Actor,$ Traffic Simulator$)$.

The relation between actors and their intentional element is formalized as follows.

Definition 4 (Actor-IE Relations) An *Actor-IE relation* is a tuple $(type, i, j)$, where:

– $type = ActIE$ states that this tuple is an Actor-IE relation,
– $i \in \mathbb{N}$ is an id of an actor,
– $j \in \mathbb{N}$ is an id of an IE.

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

Note that Actor-IE relations do not have an id: they are themselves not elements of a GRL model but rather indicate relations between elements of a GRL model (i.e., intentional elements and actors). The relation between the *Traffic Simulator* actor ($id = 0$) and the *Car objects* resource ($id = 1$), for example, can be formalized as $(ActIE, 0, 1)$.

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

Definition 5 (GRL Link) A *GRL link* is a tuple $link = (id, type, src, dest)$, where:

– $id \in \mathbb{N}$ is the unique identifier of the link,
– $type \in LinkTypes$ specifies the type of the link,
– $src \in \mathbb{N}$ is the identifier of the source IE,
– $dest \in \mathbb{N}$ is the identifier of the destination IE.

A set of links is denoted by $Link$.

Similar to IEs, links have identifiers as well. Some of the links in our example (Figure 2) are $(8, Dep, 4, 1)$, $(9, PosContr, 5, 2)$ and $(11, Decomp, 5, 4)$.

We can now form GRL models as follows.

Definition 6 (GRL Model) A *GRL model* $GRL = (IE, Act, R_{ActIE}, Link)$ consists of:

– A set IE of intentional elements (Def. 2),
– A set Act of actors (Def. 3),
– A set R_{ActIE} of Actor-IE relations (Def. 4),
– A set $Link$ of GRL links (Def. 5).

The full specification of Figure 2 is as follows.

$$
\begin{aligned}
IE = \quad & \{(1, Resource, \text{Car objects}),\\
& (2, Softgoal, \text{Realistic simulation}),\\
& (3, Softgoal, \text{Simple design}),\\
& (4, Softgoal, \text{Easy to use}),\\
& (5, Goal, \text{Generate cars}, XOR),\\
& (6, Task, \text{Create new cars}),\\
& (7, Task, \text{Keep same cars})\}\\
Act = \quad & \{(0, Actor, \text{Traffic Simulator})\}\\
R_{ActIE} = \quad & \{(ActIE, 0, 1), \ldots, (ActIE, 0, 7)\}\\
Link = \quad & \{(8, Dep, 4, 1)\\
& (9, PosContr, 6, 2)\\
& (10, NegContr, 6, 3)\\
& (11, Decomp, 6, 5)\\
& (12, Decomp, 7, 5)\}
\end{aligned}
$$

Definition 6 only sums up the elements of the model and not the constraints that make a *valid* GRL model. We will do so in the next definition. Note that in this definition, we use a subscript notation to refer to an element with a specific id. That is, IE_i, $Link_j$, and Act_k refer respectively to the IE with id i, the link with id j, and the actor with id k.

Definition 7 (Valid GRL Model) A GRL model $GRL = (IE, Act, R_{ActIE}, Link)$ (Def. 6) is a *valid GRL model* iff the following conditions are satisfied:

1. ids are globally unique across IEs, Links, and Actors, i.e., let $X, Y \in \{IE, Act, Link\}$. For all X_i and Y_j: if $i = j$ then $X = Y$ and $X_i = Y_j$.
2. Intentional elements of actors exist: $\forall (ActIE, i, j) \in R_{ActIE} : \exists act_i \in Act \wedge \exists ie_j \in IE$.
3. An intentional element belongs at most to one actor: $\forall ie_i \in IE : |\{(ActIE, j, i) \in R_{ActIE}\}| \leq 1$.
4. Links connect intentional elements: $\forall (i, type, j, k) \in Link : \{ie_j, ie_k\} \subseteq IE$.

Let us briefly verify that our previous formalization of Figure 2 satisfies all the constraints of Definition 7:

1. All elements in the formalization have different ids, so this constraint is satisfied.
2. R_{ActIE} contains one element for each IE with id i, so this constraint is satisfied as well. Note that, in line with the GRL metamodel, R_{ActIE} does not relate links with actors.
3. There is one actor ($id = 0$) and this is the only actor that appears in R_{ActIE} so this constraint is satisfied.
4. All links connect IEs: the contribution links connect elements with ids 2, 3, and 6, which are all IEs; the decomposition links connect elements with ids 5, 6, and 7,

which are all IEs; and the dependency link connects id 1 with 4, which are both IEs.

5.2 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), namely *generic arguments* and *attack links*. We illustrate the new elements using the RationalGRL model in Figure 13, which is an extension of Figure 2.

Definition 8 (Generic Argument) A generic argument is a tuple $ga = (id, type, name)$, where:

- $id \in \mathbb{N}$ is the identifier of the generic argument,
- $type = GenArg$ states that the tuple is a generic argument,
- $name \in Names$ is a string description of its name.

A set of generic arguments is denoted by GA.

Thus, a generic argument is any element in the RationalGRL model that is not an intentional element or an actor. In the example, some of the generic arguments are $(13, GenArg,$ Redundant$)$ and $(14,$ $GenArg$, Necessary$)$. Note that a constraint of a GRL model (Definition 6) is that

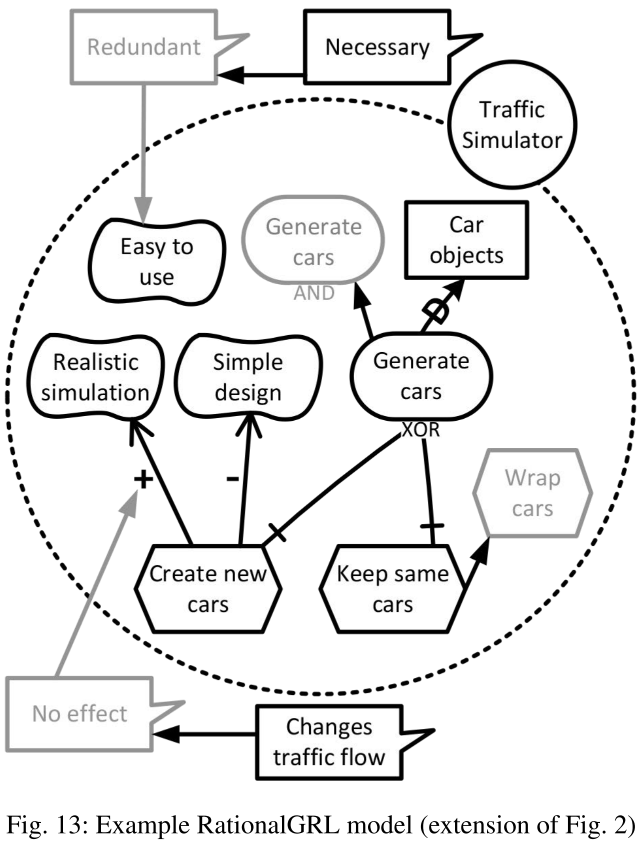


Fig. 13: Example RationalGRL model (extension of Fig. 2)

GRL links should connect IEs, which means that in generic arguments cannot be connected with GRL links.

We can now define *arguments*.

Definition 9 (Argument) An argument A is on of the following tuples:

- An intentional element ie (Def. 2),
- An actor act (Def. 3),
- An Actor-IE relation r_{ActIE} (Def. 4),
- A link $link$ (Def. 5),
- A generic argument ga (Def. 8).

This definition captures the specification in the RationalGRL metamodel (Figure 6) in which the class Argument is a superclass of GenericArgument and GRLModelElement. In sum, we define an argument simply as any one of the GRL IEs or links, or a generic argument.

Examples of arguments in Figure 13 are $A_5 = (5,$ $Goal$, Generate cars$, AND)$, $A_5' = (5, Goal,$ Generate cars$, XOR)$, $A_4 = (4, Softgoal,$ Easy to use$)$ and $A_{13} = (13,$ $ArgElem$, Redundant$)$. Note that the two arguments A_5 and A_5' show an important difference between RationalGRL models and valid GRL models: While a valid GRL model disallows multiple elements with the same identifier (Definition 7, condition 1), RationalGRL models do not enforce this restriction. This is because it is possible to create multiple arguments for the same element, where each argument contains different content for the same element. However, the set of *accepted* elements in a RationalGRL should all have unique identifier (see Definition 15).

Definition 10 (Attack Link) Given a set of arguments Arg, an attack link $att = (A_i, A_j)$ means:

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

A set of attack links is denoted by Att.

As an example, take the arguments A_5 for the goal 'Generate cars' (AND decomposition), A_5' for the goal 'Generate cars' (XOR decomposition), A_4 for the softgoal 'Easy to use' and the generic argument A_{13} ('Redundant') against the softgoal ('Easy to use'). Given these arguments there are two attack links (Figure 13), namely (A_5', A_5) and (A_{13}, A_4).

We now define a RationalGRL model.

Definition 11 (RationalGRL Model) A *RationalGRL model* $RatGRL = (Arg, Att)$ consists of a set of arguments $Args$ (Def. 9) and a set of attack links Att (Def. 10).

The definition of a RationalGRL model collects all the previously defined GRL and RationalGRL elements in a single definition. For completeness, we now provide the full specification of Figure 13. Let us first enumerate all the arguments used in this example:

$$\begin{aligned}
A_0 =& (0, Actor, \text{Traffic simulator})\\
A_1 =& (1, Task, \text{Car objects}),\\
A_2 =& (2, Softgoal, \text{Realistic simulation}),\\
A_3 =& (3, Softgoal, \text{Simple design}),\\
A_4 =& (4, Softgoal, \text{Easy to use}),\\
A_5 =& (5, Goal, \text{Generate cars}, AND),\\
A'_5 =& (5, Goal, \text{Generate cars}, XOR),\\
A_6 =& (6, Task, \text{Create new cars}),\\
A_7 =& (7, Task, \text{Wrap cars}),\\
A'_7 =& (7, Task, \text{Keep same cars}),\\
A_8 =& (8, Dep, 4, 1),\\
A_9 =& (9, PosContr, 6, 2),\\
A_{10} =& (10, NegContr, 6, 3),\\
A_{11} =& (11, Decomp, 6, 5),\\
A_{12} =& (12, Decomp, 7, 5),\\
A_{13} =& (13, GenArg, \text{Redundant}),\\
A_{14} =& (14, GenArg, \text{Necessary}),\\
A_{15} =& (15, GenArg, \text{No effect}),\\
A_{16} =& (16, GenArg, \text{Changes traffic flow}),\\
A_{17} =& (ActIE, 0, 1), \ldots, A_{23} = (ActIE, 0, 7)
\end{aligned}$$

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

$$\begin{aligned}
Arg =& \{A_0, \ldots, A_5, A'_5, A_6, A_7, A'_7 \ldots, A_{23}\}\\
Att =& \{(A'_5, A_5), (A'_7, A_7), (A_{13}, A_4), (A_{14}, A_{13}),\\
& (A_{15}, A_9), (A_{16}, A_{15})\}
\end{aligned}$$

In Figure 13, it can be read from that arguments A_5, A_7, A_{13} and A_{15} are currently disabled (rejected). However, we have not yet make it clear how exactly this is computed. We will do so in the following definitions. In order to compute when an argument is accepted and when it is not we use argumentation semantics. We use the standard approach by Dung [10].

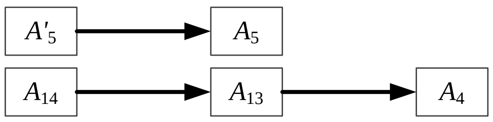


Fig. 14: Example argumentation framework, subset of the RationalGRL model from Figure 13.

Definition 12 (Argumentation Framework) An *argumentation framework* $AF = (Arg, Att)$ consists of a set of arguments Arg and a set of attack relations $Att \subseteq Arg \times Arg$.

Note that the definition of a RationalGRL model (Definition 11) is exactly the same as the definition for an argumentation framework. This allows us to use the following definitions directly.

Definition 13 (Attack, conflict-freeness, defense, admissibility, preferred extension) Suppose an argumentation framework $AF = (Arg, Att)$, two sets of arguments $S \cup S' \subseteq Arg$, and some argument $A \in Arg$. We say that

- S *attacks* A iff some argument in S attacks A,
- S *attacks* S' iff some argument in S attacks some argument in S',
- S is *conflict-free* iff it does not attack itself,
- S *defends* A iff for each B such that B attacks A, S attacks B,
- S is *admissible* iff S is conflict-free and defends each argument in it.
- S is a *preferred extension* iff it is a maximal (w.r.t. set inclusion) admissible set.

Let us explain these definitions using the example argumentation framework in Figure 14, which is a subset of our RationalGRL model from Figure 13, containing only arguments A_4, A_5, A'_5, A_{13}, and A_{14}. In this example, there are five admissible sets: $\{A'_5\}$, $\{A_{14}\}$, $\{A'_5, A_{14}\}$, $\{A_4, A_{14}\}$ and $\{A_4, A'_5, A_{14}\}$. In the admissible sets that contain both A_4 and A_{14}, we say that A_{14} defends A_4 against its attacker A_{13}. Sets containing both A_5 and A'_5, or both A_{13} and either A_4 or A_{14} are not conflict free. Sets containing A_5 are not admissible, as they do not defend A_5. Similarly, sets containing A_4 but not A_{14} are not admissible as they do not defend themselves against A_{13}.

The notion of admissible sets gives rise to various possible extensions of an argumentation framework; in this article, we use the preferred extension. In Figure 14, there is one preferred extension, namely $\{A_4, A'_5, A_{14}\}$. It is possible to have multiple preferred extensions in cases where the argumentation framework contains cycles. A simple example of such an argumentation framework is shown in Figure 15, where arguments A_x and A_y attack each other. There are two preferred extensions, namely $\{A_x\}$ and $\{A_y\}$. Although the algorithms in Section 5.4 do not generate mutually attacking arguments such as in Figure 15, our formal framework does not explicitly forbid attack cycles in RationalGRL models. We discuss this in more detail in Section 7.2.

The status of individual arguments can now be determined on the basis of the preferred extensions. Recall from the metamodel (Figure 6) that an argument can be *acceptable*, *undecided* or *rejected*.

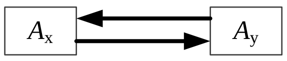


Fig. 15: Example of a cyclic argumentation framework.

Definition 14 (Argument Acceptability)

- An argument is *acceptable* w.r.t. an argumentation framework AF if it is in every preferred extension of AF.
- An argument is *undecided* w.r.t. an argumentation framework AF if it is in at least one but not all preferred extensions of AF.
- An argument is *rejected* w.r.t. an argumentation framework AF if it is in no preferred extension of AF.

Take our two simple examples. In Figure 14, we have that arguments A_4, A_5' and A_{14} are acceptable and arguments A_5 and A_{13} are rejected. In Figure 15, both arguments A_x and A_y are undecided. Returning to our larger example from Figure 13, we can see that arguments A_5, A_7, A_{13} and A_{15} are rejected and all the other arguments are acceptable.

5.3 Translating between RationalGRL and GRL

Now that we have formalized both a GRL model and a RationalGRL model, we present algorithms to translate between these two models. Both of these two translation algorithms are straightforward, which is result of the fact that the two models are formalized in a very similar way.

GRL to RationalGRL Translation We start with the translation algorithm from GRL to RationalGRL, which is shown in Algorithm 1. The translation algorithm takes a GRL model $GR = (IE, Act, R_{ActIE}, Link)$ (Definition 6) as input. On line 2, it collects all the elements of the GRL model into a set Arg, which represents the set of arguments of the RationalGRL model. On line 3, the set of attack links is initialized with an empty set, and the new RationalGRL model (Arg, Att) is returned on line 4.

Algorithm 1 GRL to RationalGRL Translation

```
1: procedure $ToRationalGRL(GRL)$
2:     $Arg \leftarrow IE \cup Act \cup R_{ActIE} \cup Link$
3:     $Att \leftarrow \emptyset$
4:     return $(Arg, Att)$
5: end procedure
```

RationalGRL to GRL translation The translation from a RationalGRL model to a GRL model is given in Algorithm 2. First. the arguments are each put in the corresponding GRL component sets. The procedure $ComputeExtensions(Arg, Att)$ (line 2) computes the (preferred) extensions of the RationalGRL model (Definition 13). A GRL model is then generated from each of the preferred extensions. 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 – here, $Arg.type$ refers to the $type$ element in an argument tuple Arg – and each case ensures the argument is put in the right GRL component. Finally, the algorithm returns a GRL model on line 16. As an example, compare the RationalGRL model in Figure 13 to the GRL model in Figure 2, where the latter is a translation of the former.

Valid RationalGRL model While we have defined a notion of a *valid GRL model* (Definition 7), we have not done so for a RationalGRL model yet. We define a RationalGRL model as valid if and only if the RationalGRL to GRL translation results in a valid GRL model. Thus, we do not have to reiterate all conditions on a GRL model, but use the translation algorithm.

Definition 15 (Valid RationalGRL Model) A RationalGRL model $RatGRL = (Arg, Att)$ (Def. 11) is a *valid RationalGRL model* iff $ToGRL(RatGRL)$ (Algorithm 2) is a valid GRL model (Def. 7).

5.4 Algorithms for argument schemes and critical questions

We have now formalized a *static* representation of the RationalGRL framework. In this section we formalize algorithms for applying argument schemes and critical questions. These algorithms are applied to RationalGRL models (Definition 11) and produce new arguments and attack relations.

Algorithm 2 RationalGRL to GRL Translation

```
1: procedure $ToGRL(RatGRL)$
2:     $Ext \leftarrow ComputeExtensions(Arg, Att)$
3:     for $E \in Ext$ do
4:         $IE \leftarrow \emptyset$, $Act \leftarrow \emptyset$, $R_{ActIE} \leftarrow \emptyset$, $Link \leftarrow \emptyset$
5:         for $A \in E$ do
6:             switch $A.type$ do
7:                 case $IETypes$
8:                     $IE \leftarrow IE \cup \{A\}$
9:                 case $Actor$
10:                    $Actor \leftarrow Actor \cup \{A\}$
11:                case $ActIE$
12:                    $ActIE \leftarrow ActIE \cup \{A\}$
13:                case $LinkTypes$
14:                    $Link \leftarrow Link \cup \{A\}$
15:        end for
16:        return $(IE, Act, R_{ActIE}, Link)$
17:    end for
18: end procedure
```

As discussed in Section 4, the argument schemes and critical questions of Table 2 all lead to one of three operations: INTRO introduces a new RationalGRL element and DISABLE creates a new argument that attacks another argument. The REPLACE operation can be seen as a combination of INTRO and DISABLE: a new argument corresponding to a GRL element is added and this new argument attacks a previous version of the element or link. We will discuss all three operations in separate sections below.

In all of the following algorithms, we assume that:

- The algorithms are applied to some valid RationalGRL model $RatGRL$ (Definition 15),
- The procedure $mintId()$ generates a new unique id.

5.4.1 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.

Algorithm 3 AS0: a is an actor

```
1: procedure $AS_0(n)$
2:     $A \leftarrow (mintId(), Actor, a)$
3:     $Arg \leftarrow Arg \cup \{A\}$
4: end procedure
```

Algorithm 3 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 actor is added as a new argument. As an example, we can formalize the addition of actor “Traffic Simulator” to an empty RationalGRL model (i.e. $Arg = \emptyset$ and $Att = \emptyset$) as executing the algorithm AS_0(Traffic simulator), which results in a RationalGRL model with argument $A_0 = (0, Actor, \text{Traffic simulator}) \in Arg$.

Algorithm 4 AS1: Actor with id i has resource R

```
1: procedure $AS_1(i, R)$
2:     $res\_id \leftarrow mintId()$
3:     $A_1 \leftarrow (res\_id, Resource, R)$
4:     $A_2 \leftarrow (ActIE, i, res\_id)$
5:     $Arg \leftarrow Arg \cup \{A_1, A_2\}$
6: end procedure
```

In Algorithm 4, we have slightly reworded the original AS1 from Table 2 to “Actor with id i has resource R”. Furthermore, we assume actor a already exists when Algorithm 3 is run. The algorithm itself runs as follows: on line 2, a unique id is assigned to variable res_id (“resource identifier”). On line 3, an argument for a resource with name R and identifier res_id is assigned to A_1. On line 4, a second argument A_2 is created, which is an argument for an Actor-IE relation (Def. 4) between the input actor i and the newly created element res_id. Finally, on line 5, the arguments are added to the set of arguments Arg.

As an example, consider the addition of resource “Car objects” to the RationalGRL model we just discussed (where $A_0 = (0, Actor, \text{Traffic simulator})$). Adding the resource can then be formalized as executing $AS1(0, \text{Car objects})$, resulting in the addition of two arguments $(1, Resource, \text{Car object})$ and $(ActIE, 0, 1)$ which are added to the set of arguments Arg.

Since arguments schemes AS2-AS4 are very similar to AS1, we don’t show these algorithms here.

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

```
1: procedure $AS_5(i, T)$
2:     $task\_id \leftarrow mintId()$
3:     $A_1 \leftarrow (task\_id, Task, T)$
4:     $A_2 \leftarrow (mintId(), Decomp, i, task\_id)$
5:     $Arg \leftarrow Arg \cup \{A_1, A_2\}$
6: end procedure
```

Similar to the previous algorithm, we have slightly reworded critical question AS5 in Algoritm 5. 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 the algorithm, on line 2 a unique identifier is created for the task, which is created on line 3. On line 4 an argument is created for the decomposition link $(mintId(), Decomp, i, task\_id)$ (Def. 5, going from the existing goal with identifier i to the new task with identifier $task\_id$. On line 5 the two arguments are added to the set of arguments Arg.

As an example, suppose we are adding the decomposition of goal “Generate cars”, expressed as argument $A_5 = (5, Goal, \text{Generate cars}, AND)$, into task “Keep same cars”, and suppose we have the following argument for the goal: . Adding the decomposition can be formalized as executing $AS_5(5, \text{Keep same cars})$, and results in two new arguments: $A_7 = (7, Task, \text{Keep same cars})$ and $A_{12} = (12, Decomp, 7, 5)$, which are both added to the set of arguments Arg.

The remaining argument schemes AS6-AS12 are all similar to algorithms 3-5 and are therefore not shown here.

The critical questions of type *INTRO* are very similar as well, with one exception: they 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*. Therefore, we are not shown here as well.

5.4.2 DISABLE algorithms

As discussed before, algorithms of type *DISABLE* consist of adding a new argument attacking an existing argument. The argument that is added is not an argument for a GRL element or link, but it is rather a generic argument (Definition 8).

In all of these algorithms, we assume the critical question is answered affirmatively, as indicated in the right-most column of Table 2. For instance, for critical question CQ0 “Is the actor relevant?”, we assume it is answered with “No”. In this case, an action is required. In contrary, if the answer to the question is “Yes”, no action is required.

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

```
1: procedure $CQ_0(i)$
2:     $A \leftarrow (mintId(), GenArg, CQ0)$
3:     $Arg \leftarrow Arg \cup \{A\}$
4:     for $A_j \in \{A = (i, Actor, n) \mid A \in Arg\}$ do
5:         $Att \leftarrow Att \cup \{(A, A_j)\}$
6:     end for
7: end procedure
```

Algorithm 6 is executed when critical question CQ0 is answered 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* $(mintId(), GenArg, CQ0)$ (Def. 8). The for loop starting at line 4 then iterates over all arguments for actors, where i is the id of the actor that is no longer relevant. 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 to the argument for the actor A_j. After executing the algorithm, all existing arguments for the actor with identifier i are attacked by a newly created argument A.

Consider for example the RationalGRL model in Figures 11, which consists of an actor and a generic counterargument. Let us reconstruct this model using the application of argument schemes and critical questions to an initially empty RationalGRL model ($Arg = \emptyset$ and $Att = \emptyset$). The algorithm $AS_0(\text{Development team})$ is executed, which results in $A_1 = \{(0, Actor, \text{Development team}\})$. After this, critical question CQ0: “Is the actor Development team relevant?” is answered with “No, because the professor develops the software”. We can formalize this as executing algorithm $CQ_0(0)$ (Algorithm 6), which results in adding a generic counter argument $A_2 = (1, GenArg, \text{CQ0: Professor develops software})$ and an attack link (A_2, A_1). The RationalGRL model now consists of two arguments, and that the preferred extension is $\{A_2\}$, which means that the resulting GRL model is empty, because generic arguments are not translated to GRL.

Critical questions C1-CQ3 are all very similar to CQ0 and are therefore not shown 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:     $A \leftarrow (mintId(), GenArg, CQ5a)$
3:     $Arg \leftarrow Arg \cup \{A\}$
4:     for $A_j \in \{(k, Decomp, g\_id, t\_id) \in Arg\}$ do
5:         $Att \leftarrow Att \cup \{(A, A_j)\}$
6:     end for
7: end procedure
```

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 4, where we iterate over all decomposition links with source id g_id and target id t_id (Definition 5). This means we iterate over all decomposition links from the goal to the task, and we attack all arguments for these links on line 5 with the newly created argument.

Almost all of the remaining critical questions of type DISABLE 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 is therefore now shown here. The only other algorithm of type *DISABLE* that we discuss is Att.

Algorithm 8 can be regarded as the most general way of providing counterarguments to arguments. In all of the previous *DISABLE* algorithms, 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.

Let us reconsider Figure 11, which we previously formalized as a RationalGRL model where $Arg = \{A_1, A_2\}$, $A_1 = \{(0, Actor, \text{Development team}\})$, $A_2 = (1, GenArg,$

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

```
1: procedure $Att(A_1, \ldots, A_n, N)$
2:     $A \leftarrow (mintId(), GenArg, N)$
3:     $Arg \leftarrow Arg \cup \{A\}$
4:     for $A_j \in \{A_1, \ldots, A_n\}$ do
5:         $Att \leftarrow Att \cup \{(A, A_j)\}$
6:     end for
7: end procedure
```

CQ0: Professor develops software) and $Att = (A_2, A_1)$. Suppose we execute algorithm $Att(\{A_2\}$, Professor does not develop software). This results in $A_3 = (2, GenArg$, CQ0: Professor does not develop software) and a new attack link (A_3, A_2) (Figure 12). Since the argument for the goal is now part of the preferred extension, it does show up in the resulting GRL model after the translation of Algorithm 2.

5.4.3 REPLACE algorithms

Recall that *REPLACE* algorithms create a new argument that attacks all arguments for an existing element.

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

```
1: procedure $CQ_{Decomp}(ie_i, X)$
2:     $A \leftarrow ie_i$
3:     $A.decomptype \leftarrow X$
4:     $IEArgs \leftarrow IE_i \subseteq Arg$
5:     for $\{(A_i, A_j) \in Att \mid A_j \in IEArgs\}$ do
6:         $Att \leftarrow (A_i, A)$
7:     end for
8:     for $\{(A_i, A_j) \in Att \mid A_i \in IEArgs\}$ do
9:         $Att \leftarrow (A, A_j)$
10:        for $A_i \in IEArgs$ do
11:            $Att \leftarrow Att \cup \{(A, A_i)\}$
12:        end for
13:    end for
14:    $Arg \leftarrow Arg \cup \{A\}$
15: end procedure
```

While the original critical question CQ5c is specific to the decomposition between a goal and a task, Algorithm 9 is more generally applicable to the decomposition type of any IE, since all IEs have a decomposition type (Definition 2).

Let us go through this algorithm step by step. On line 2, a new argument A is created which is identical to original IE. On line 3 the decomposition type of the argument is changed to X – here, $Arg.decomptype$ refers to the $decomptype$ element in an argument tuple Arg. On line 4, the set $IEArgs$ is assigned with all existing arguments for the input IE – here IE_i is the set of all IEs with id i. The first two for loops on respectively lines 5 and 8 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. Then, in the third for loop on line 10, we add attack links from the argument that has just been created to all existing arguments for the IE. This ensures that all previous version of the IE are not part of the preferred extension, and as a result do not show up in the resulting GRL model. Finally, one line 14, the new argument is added to the set of arguments.

As an example, take Figure **??**. The initial RationalGRL model (before applying CQ10b), can be formalized as follows: $RatGRL = (Arg, Att)$, with $Arg = \{A_1, A_2, A_3, A_4, A_5\}$ and $Att = \emptyset$, where:

- $A_1 = (0, Goal,$ Simulate traffic$, AND)$
- $A_2 = (1, Task,$ Dynamic simulation$, AND)$
- $A_3 = (2, Task,$ Static simulation$, AND)$
- $A_4 = (3, Decomp, 1, 0)$
- $A_5 = (4, Decomp, 2, 0)$

Suppose algorithm $CQ_{Decomp}(0, XOR)$ is executed for this RationalGRL model. On lines 2 and 3 a new argument $A_6 = (0, Goal,$ Simulate traffic$, OR)$ is created. On line 4, $IEArgs$ contains all arguments for element 0, which is $\{A_1\}$ (note this does not yet include argument A_6, since it is not yet added to the RationalGRL model). The first two for loops on lines 5 and 8 do not do anything in the case of our example, since A_1 is itself not attack or does not attack any other argument. The for loop on line 10 ensures all previous versions of the element with id 0 are now attacked, which means in our case the (A_6, A_1) is added to Att. Finally, on line 14 the new argument A_6 is added to Arg.

The other *REPLACE* algorithms are similar to Algorithm 9, which can be used directly for CQ10c. For CQ12 we should make a small modification: instead of replacing the decomposition type of the IE, we replace its name. Since this is a very minor modification we don't show it here.

6 The RationalGRL Methodology and Tool

In previous sections, we have shown how the RationalGRL framework can capture stakeholders discussions, and how interactions between two types of reasoning, practical reasoning and goal modeling, leads to two interlinked models, RationalGRL and GRL models. The previous section contained a formalization of RationalGRL, which forms the starting point of this section. In this section we clarify how practitioners can actually use the RationalGRL framework by proposing a methodology (**requirement 4**) and discussing a prototype RationalGRL tool (**requirement 5**).

6.1 RationalGRL Methodology

We propose the methodology shown in Figure 16 to develop a RationalGRL model, a version of which was presented at the 2017 iStar workshop [16]. Here we assume that the initial GRL models have been created based on the requirements specification documents and the discussions of the stakeholders. The rest of the steps are as follows:

(1) Instantiate Argument Schemes (AS) – In this step, we start from the list of arguments schemes (Table 2). Whilst discussing the requirements, we select schemes from the list and instantiate them to form arguments for GRL IEs or links. In this way we build the GRL model by introducing new GRL elements (INTRO). For example, an argument scheme can be ”Goal *G* contributes to softgoal *S*”. When an argument scheme is instantiated, it corresponds to an argument for or against part of a goal model.

(2) Answer Critical Questions (CQs) – After building or modifying the initial GRL model, we ask the relevant critical questions. Since 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. For example, for the argument scheme, ”Goal *G* contributes to softgoal *S*”, there are two critical questions as follows: *Does the goal contributes to the softgoal?* and *Does the goal contributes to some other softgoals?*. When the analyst answers a critical question, a new argument scheme may be instantiated. This is done in step (3) when the operation INTRO is executed.

(3) Decide on Intentional Elements and their Relationships – By answering a critical question, one of the three operations (INTRO, DISABLE or REPLACE) is applied to the GRL model. Any of these operations impact the arguments and corresponding GRL intentional elements, modifying the initial GRL model into a RationalGRL model. Recall that INTRO means that a new argument scheme is created. That means, the current argument related to the critical question does not get attacked. In the case of DISABLE, the intentional element or its related links are disabled in the model. REPLACE introduces a new argument and attacks the original argument at the same time. This means that the original element of the argument scheme is replaced with a new one.

(4) Modify GRL Models – Based on the RationalGRL model of step (3), the GRL model can be modified: 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.

6.2 The RationalGRL Tool

The final requirement of our framework is that it has tool support (**requirement 5**). This is important for various reasons: i) although there are various approaches attempting to combine goal modeling with argumentation, we are not aware of existing tool support (see Section 7), ii) it allows us to do user tests in future research, exploring the difference between GRL and RationalGRL, iii) tool support is an excellent verification to ensure our formal framework and algorithms actually work. In this section we briefly highlight some features of the tool, and we explain some of its current limitations. The tool can be accessed from:

```
http://www.rationalgrl.com
```

Relation to jUCMNav GRL is implemented in the open-source tool jUCMNav [30], which is actively maintained[5]. jUCMNav currently has over 2,000 commits made by over 40 contributors, representing over 200,000 lines of code. jUCMNav is used actively in research, and the tool has been extended in many directions over the past few years, such as for modeling and analyzing security misuse cases [9], supporting activity theory [14], combined goal and feature model reasoning [24], and enterprise architecture modeling [26]. Furthermore, jUCMNav includes Use Case modeling and contains various GRL evaluation algorithms with which the satisfiability of goals can be determined.

The jUCMNav tool is implemented as an Eclipse plugin. While we aim to integrate our framework into jUCMNav in the future, a web-based version of RationalGRL (and GRL) is more suited for current purposes. Since RationalGRL does not make use of the many features of jUCMNav, we believe a light-weight version is sufficient for our current aims, and may benefit the community in general. Our tool is thus not meant to replace jUCMNav. For simplicity we have a web-based version, and it has the functionality to export to jUCMNav.

Tool overview The RationalGRL tool is an open-source web-based Javascript application, which runs on all modern browsers. It is based on our formalization in Section 5 and

[5] `http://jucmnav.softwareengineering.ca/foswiki/ProjetSEG`

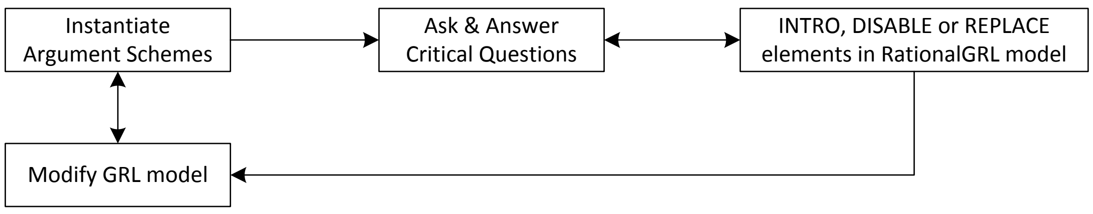
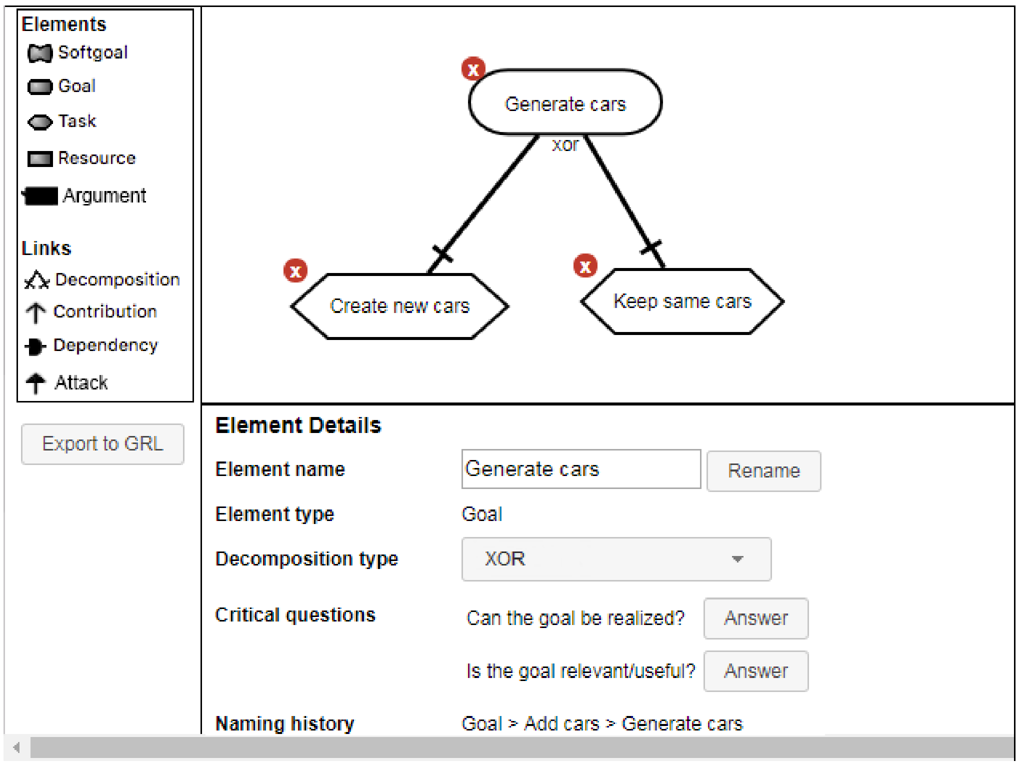

Instantiate Argument Schemes
Ask & Answer Critical Questions
INTRO, DISABLE or REPLACE elements in RationalGRL model
Modify GRL model

Fig. 16: The RationalGRL Methodology



Element Details

Element name	Generate cars
Element type	Goal
Decomposition type	XOR
Critical questions	Can the goal be realized?
	Is the goal relevant/useful?
Naming history	Goal > Add cars > Generate cars

Fig. 17: Overview of the RationalGRL tool

provides export functionality to jUCMNav, using the translation algorithm (Algorithm 2). It contains all of the GRL elements and links, except for beliefs and actors. We have argued in Section 2.2 that arguments can be seen as an extension to beliefs, which is the reason why we did not implement them. Actors are also missing, but they will be added in the future.

When the tool starts, the user is presented with a screen as in Figure 17. This screen shows the palette of elements and links (top left pane), a canvas on which RationalGRL models can be built (top right pane), an 'Export to GRL' button (bottom left pane), and a details pane of the currently selected elements (bottom right pane). Elements and links can be added to the canvas by selecting them on the left and clicking on the canvas, thus capturing the INTRO operation (Section 5.4.1). This makes it possible to build and modify GRL models using the tool.

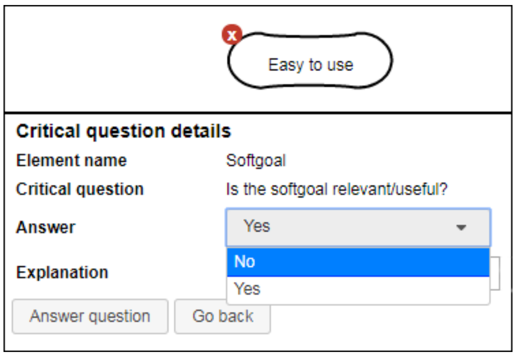
In total there are four different types of details panes, which we now explain in turn.

IE and links details pane Figure 17 shows the details pane for the goal 'Generate cars'. The user can change the name of the IE by changing the name and clicking 'Rename', which will update the naming history on the bottom of the pane. The naming history is (simplified) implementation of the REPLACE operation on IEs, since it is a very simple form of *clarification*. The main difference with jUCMNav is that we have a renaming history, so the user can see which names the IE had prior to the current one. If the IE has decomposition links, the user can change the decomposition type as well. Each IE and link has a set of associated critical questions, which can be answered in the details pane by click on the button next to the question.

Critical question details pane The details panel can be used to answer the critical questions of the RationalGRL framework (Table 2): Figure 17, for example, shows two critical questions for the goal IE "Generate cars", namely "Can the goal be realized?" (CQ3) and "Is the goal relevant/useful?" (CQ11). As an example, consider Figure 18, where the softgoal "Easy to use" is questioned with the relevancy question (CQ11). It is possible to select the answer and provide an explanation for the answer.

Assume that in the example of Figure 18 the user selects "No" and clicks the "Answer question" button. A new argument is then automatically created that attacks the softgoal and the details pane shows the critical question as answered (Figure 19). This is an implementation of a DISABLE algorithm (Section 5.4.2) similar to Algorithm 6: a new argument is added that attacks the existing argument.

Argument details pane It is also possible to attack arguments by adding an "Argument" element and an "Attack" link manually. Consider, for example, Figure 20. Here, a



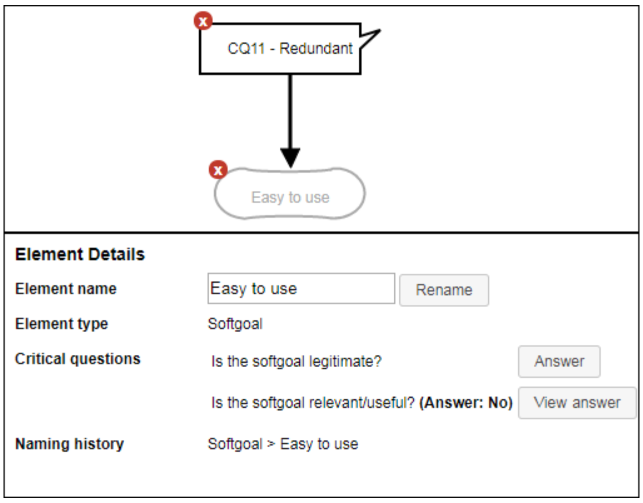
Critical question details

Element name	Softgoal
Critical question	Is the softgoal relevant/useful?
Answer	Yes
Explanation	

Fig. 18: Critical question details pane for "Is the softgoal relevant/useful?"

new argument "Necessary", which attacks the previously generated argument based on CQ11, has been added by the user. As the details pane shows, this new argument is not based on a CQ. It is further worth noting that it is possible to provide further a explanation in argument elements, allowing for more fine-grained rationalizations.

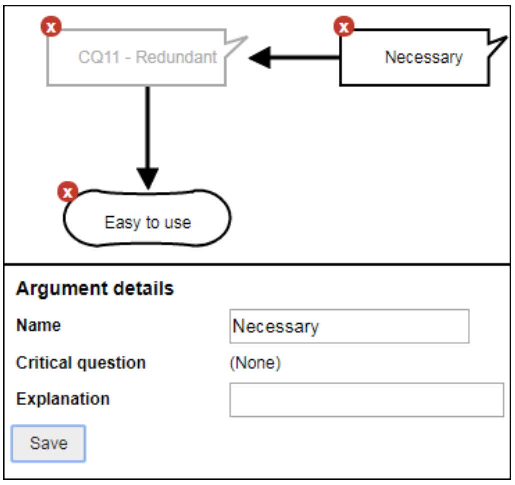
Argumentation semantics The RationalGRL tool computes the acceptability of arguments on the fly. In the example of Figure 19, the original (argument for) softgoal "Easy to use" is rejected because its only attacker "CQ11 - Redundant" is accepted. However, if we then attack "CQ11 - Redundant" with a new argument "Necessary", the original argument for "Easy to use" is again accepted because its only attacker is rejected (Figure 20). Note that when computing the acceptability of arguments, the RationalGRL tool makes the assumption that there are no attack cycles in the model and that hence there is one unique preferred extension (cf. Section 5.2) – if the user creates an attack cycle, an error message is shown.



Element Details

Element name	Easy to use	Rename
Element type	Softgoal	
Critical questions	Is the softgoal legitimate?	Answer
	Is the softgoal relevant/useful? **(Answer: No)**	View answer
Naming history	Softgoal > Easy to use	

Fig. 19: Effect of answering CQ11: "Is the softgoal relevant/useful?" with "No"



Argument details

Name	Necessary
Critical question	(None)
Explanation	

Fig. 20: Attacking an argument with a generic argument "Necessary"

Argument schemes and critical questions The tool contains more argument schemes and critical questions than those that we initial collected in Table 2. As we mentioned before, our table is not meant to be exhaustive, and it is straightforward to add more argument schemes and critical questions if necessary. For instance, the table does not contain critical questions for links between all IEs (e.g., a contribution link from a task to a goal does not occur in the table). In the tool, we have added critical questions to all links and IEs. We implemented an *argument schemes database*, which is designed to be extended easily.

Export to jUCMNav The tool implements the RationalGRL to GRL translation (Algorithm 2) through an export function. RationalGRL models built in the RationalGRL tool can be exported to the `.grl` file format, which can be imported in jUCMNav. Note that in order for the GRL models to be rendered correctly, we recommend using the Graphviz dot[6] as the auto-layout tool (this can be selected under "Auto layout preferences" when importing the file in jUCMNav).

Two examples of this export are provided in Appendix C. Figure 21 shows the model that was discussed earlier in this paper (Figure 13) in the RationalGRL tool. Recall that translating this model to GRL provided us with the model in Figure 2. This is also what follows from our export: if we export the RationalGRL model and then import the resulting GRL model in jUCMNav, we get the model in Figure 22, which is the same as Figure 2.

Our translation and export function uses the argument acceptability as a way of determining the GRL model. Take

[6] `http://www.graphviz.org/`

for example, the RationalGRL model in Figure 23. A positive contribution is added from "Keep same cars" to "Simple design". More inportantly, an argument "not enough CPU" has been added to the model. This argument attacks the "Realistic simulation" softgoal and the "Create new cars" task, arguing that there is not enough processing power for either of these to be feasible in the traffic simulator. Now, if we export this to GRL, the GRL elements that are rejected or disabled (greyed out in Figure 23) are not included. This can be seen in Figure 24, where the jUCMNav GRL model that was exported from the RationalGRL tool is shown. The pair of figures 23 and 24 also nicely shows the added value of RationalGRL: Figure 23 shows that there can be a larger discussion and rationalization underlying even a fairly simple goal model such as the one in Figure 24.

Limitations For usability purposes, the REPLACE operation has been implemented differently than in our formal framework (Section 5.4.2). In the formal framework, a REPLACE introduces a new argument that replaces (and therefore attacks) all previous arguments with the same identifier. Including REPLACE like this in the tool would mean that, when the name of an element is changed, we would need to render all the previous versions of the element on the canvas of the tool. Since this would quickly become very cumbersome, we decided to implement a naming history for each element. For example, in Figure 17, the goal "Generate cars" was previously named "Add cars". In this way, a history is kept of how an element name was clarified without cluttering the model canvas. This feature is currently very limited and can be extended in various was, for instance by changing the IE type. Similarly, the decomposition type can be changed in the details pane (Figure 17) without introducing new elements that attack the original ones (cf. Figure 13, where the old decomposition type AND of "Generate traffic" is shown as an explicit argument).

7 Discussion

7.1 Related work

Design Rationale Argumentation in software design has been the subject of the research on so-called *design rationale* (DR) [35], 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 [33, 18] has been made, where argument diagrams represent reasoning from premises to conclusions. More recent research 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 [32,34], or which parts of the design process are best explicitly documented [12].

Software design and requirements engineering are very closely related [31] and hence the insights from the DR literature are directly applicable to RE. The RationalGRL framework essentially incorporates the core ideas from DR into goal-oriented requirements engineering by explicitly including arguments for and against 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.* [19], 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 they 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.* [44] further extend the framework by Haley *et al.*, including algorithms for Dung-style argumentation semantics [10] and a database of specific ways in which to attack (or mitigate) risks. This extension can be likened to a set of critical questions for risks and security requirements (cf. Yu *et al.* [44] Section 3.1).

Another recent example of the use of arguments in goal-oriented requirements engineering is the work by Murukannaiah *et al.* [29], who propose Arg-ACH, where 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 [18]. 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.* [21] propose a 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 [10] are then used to determine the risks that are still valid, and those that have been successfully rejected. Elrakaiby *et al.* [11] 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 or an insufficiency. An inconsistency can be either with the client's previous statements or with the requirement engineer's beliefs. An insufficiency occurs 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. These counterarguments say that, for example, the functionality expressed in the statement cannot be realized or is irrelevant. Elrakaiby *et al.* coded the data from 34 requirement elicitation interviews. They identified 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.* [29] 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.* [11] provide a case study similar to our research, identifying, many counterarguments specifically with respect to realisability, relevance and clarity (cf. CQ2a, CQ3, CQ11 and CQ12 in Table 2). Similar to RationalGRL, the use of Dung-style argumentation semantics to compute the acceptable claims in RE is also advocated by the literature [44, 21, 11].

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 [19, 44, 21] or the reasoning is about single goals or tasks instead of about the wider context as represented in a goal model [19, 44, 29, 11]. 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 5 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 [1] is meant to capture the rationales 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 [37] 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 subgoal or task in a goal model is then an explicit argumentative move in the discussion. 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 [22]), argumentation produces different, richer and complementary information to 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 Bagheri and Ensan [3] and Mirbel and Villata [27] use abstract argumentation frameworks (see Definition 12) 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 [10]. 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 as such, 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 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 [1], which take into account conflicts, decompositions and dependencies. RationalGRL focuses on what is not captured in the work discussed above [3,27], 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 [3,27] 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.* [22]. 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.* use a well-known fictitious 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).

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 16 can be seen as a more specific version of GAM explicitly meant for goal modeling. The argument schemes and critical questions in RationalGRL provide clearer handles for goal modeling (cf. requirement 4 of our framework, Section 1). Furthermore, the RationalGRL framework adds full tool support for the goal modeling and reasoning process, which GAM lacks. GAM does contain more specific ways of dealing with different types of clarity problems; in RationalGRL clarification is captured as a single critical question (CQ12, Table 2). However, as argued above further critical questions can easily be added to RationalGRL if needed.

Jureta *et al.*'s framework also includes an argumentation part, where reasons (justifications) for conclusions are given as formal structured arguments[7]. 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.

RationalGRL allows for two-way translation between arguments about goals and standard goal models (cf. Section 5.3). In contrast, Jureta *et al.* only provide a mapping from goal models to structured arguments, and the step from structured arguments or argument diagrams to goal models is never formally defined. In previous work on the RationalGRL framework [39,40], we extended Jureta *et al.*'s work and translated argument diagrams to GRL models (an automatic translation tool is discussed in [40]). This gives us 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. This means that any stakeholder trying to understand the discussion has to parse two complex diagrams containing goals, alternative solutions, tasks, and so forth. RationalGRL overcomes this problem by integrating argumentation and goal modeling in a single modeling language.

7.2 Future work

RationalGRL lays down a basic framework for argumentation in requirements engineering, and all aspects of this framework are intended to be fully extensible. We envision a large number of open issues to be explored in future research.

Specific Domains The argumentation schemes and critical questions presented in this paper are focused only on the core goal modeling process, that is, the discussion about the goals and functionality of an information system. In the RE process, there are many more domain specific discussions that also take place, such as discussions involving expert opinions [29], discussions about security requirements [19,

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

44,21], architectural principles [26], legal compliance [15], and so forth. From a knowledge engineering perspective, including the argument schemes and critical questions associated with these domains in the RationalGRL framework is a time-consuming task. However, from a formal perspective adding new schemes and questions is easy: as already discussed by Bex *et al.* [5], critical questions for argument schemes will always lead to either new information being introduced, new information replacing old information, or new information attacking old information. This corresponds to the INTRO, REPLACE and DISABLE operations in the RationalGRL framework, which have been formalized in Algorithms 3-9 in Section 5.4, hence it would be suitable for other types of schemes and questions as well.

Formal argumentation The amount of theory from computational argumentation used in this article has been relatively small. Our intention is to create a bridge between the formal theories in argumentation and the practical tools in requirements engineering. Now that the initial framework has been developed, however, it is worth exploring what additional techniques computational argumentation has to offer in more detail. For instance, in our framework we did not include the possibility of expressing preferences between mutually inconsistent arguments (e.g. such as in Figure 15). Using the work by Modgil [28], it is possible to provide explicit arguments for preferences, thus allowing us to make a reasoned choice between two options.

Tool support The current tool is a prototype that implements the RationalGRL framework in a fairly simple way. That is, by offering the possibility to argue and ask critical questions about a goal model, and then export this goal model for further analysis in jUCMNav. Integration of RationalGRL elements into jUCMNav, or at least creating an import function to import jUCMNav goal models into the tool would allow us to tap into the large amount of research and development that is being carried out with jUCMNav. Furthermore, the tool could be expanded to include the new features described above, such as new argument schemes and reasoning with preferences. In addition to such extensions, it is interesting to think about new possibilities. One idea is to use the tool for collaborative on-line goal modeling, similar to Github: since the renaming (i.e. replacement) and deletion (i.e. disabling) of elements are all logged, it is easy for a stakeholder to continue working on a model that was made by another stakeholder. Furthermore, critical questions can be included for other stakeholders to answer at a later date. As a formal underpinning of this asynchronous communication between users, it would make sense to capture requirements engineering and software design processes as dialogs between parties [13,6], which are a natural fit with the question-answer format employed in the RationalGRL framework.

Empirical validation We have performed a case study and found more than 200 incidences of arguments and questions. However, what is still lacking is further empirical investigation of whether, and how, the tools and techniques provided by the RationalGRL framework really improve early phase requirements engineering. The complexity of the domain makes focused experiments difficult, but not impossible. For example, Murakannaiah *et al.* [29] provided a realistic but bounded problem for their subjects to solve, and similar experiments are imaginable to test RationalGRL: have two groups solve a goal modeling problem, one using only GRL and one using RationalGRL, and examine how they perform. Naturally, the type of problem will greatly influence the outcome. For a simple problem which two people have to solve in an hour, RationalGRL will most likely be of little benefit. However, a more complex problem on which multiple persons have to work asynchronously might benefit from the extra tools offered by RationalGRL. Furthermore, the outcome also depends on which part of the set of tools and techniques offered by RationalGRL is tested: the methodology, the tool, or a list of critical questions to serve as reminders during the goal modeling process (cf. [34]).

7.3 Conclusion

The process of constructing a goal model 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 back in the resulting model. This makes it not only more difficult to understand the model, but other stakeholders may end up having similar discussions throughout the design and development phase as well. Furthermore, the disconnect between goal models and their underlying beliefs and opinions may lead to a poor understanding of the problem and solution domain.

In order to solve these problems, we present RationalGRL: a framework for integrated goal modeling and argumentation. We extend the well-known goal modeling language GRL with argument schemes and critical questions that can be used to analyze and guide stakeholders' discussions about goal models. Furthermore, we provide formal argumentation semantics for the new RationalGRL language. Our approach, thus, provides a rationalization for the elements of the goal model in terms of underlying arguments, and helps in understanding why parts of the model have been accepted while others have been rejected. In the introduction, we identified five important requirements for our framework. Below, we discuss how the RationalGRL framework meets these requirements.

Our list of argument schemes and critical questions in Table 2 was constructed by performing an extensive case study in which we analyzed 153 pages of transcripts of discussions among designers of a traffic simulator information system. In this way, we ensure that he argumentation techniques capture the actual discussions of the stakeholders or designers in the early requirements engineering phase (**requirement 1**).

The metamodel of the RationalGRL framework in Figure 6 clearly specifies the formal traceability links elements of the GRL goal model and the underlying arguments. (**requirement 2**). In addition to this metamodel, we provide formal semantics for RationalGRL by formalizing the GRL language in propositional logic and rendering arguments about a GRL model as a formal argumentation framework [10]. We, then, formally capture the link between argumentation and goal modeling 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* [1]).

One of our main goals is to provide means for RE practitioners to capture the underlying arguments of goal models by using the RationalGRL framework (**requirement 4**). To this end, we propose a methodology, which consists of developing goal models and posing arguments based on schemes in an integrated way. Finally, we have implemented the RationalGRL tool, a web-based prototype[8], for modeling argument schemes and critical questions and for reasoning about goal models with respect to them (**requirement 5**).

[8] http://www.rationalgrl.com

References

1. 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.
2. K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171(10):855–874, 2007.
3. 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.
4. T. J. Bench-Capon and P. E. Dunne. Argumentation in Artificial Intelligence. *Artificial intelligence*, 171(10-15):619–641, 2007.
5. F. Bex, H. Prakken, C. Reed, and D. Walton. Towards a Formal Account of Reasoning about Evidence: Argumentation Schemes and Generalisations. *Artificial Intelligence and Law*, 11(2/3):125–165, 2003.
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. O. Daramola, G. Sindre, and T. Moser. Ontology-based support for security requirements specification process. *Lecture Notes in Computer Science (LNCS)*, 7567:194–206, 2012.
10. 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.
11. 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.
12. 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.
13. A. Finkelstein and H. Fuks. Multiparty specification. In *ACM SIGSOFT Software Engineering Notes*, volume 14, pages 185–195. ACM, 1989.
14. G. Georg, G. Mussbacher, D. Amyot, D. Petriu, L. Troup, S. Lozano-Fuentes, and R. France. Synergy between activity theory and goal/scenario modeling for requirements elicitation, analysis, and evolution. *Information and Software Technology*, 59:109–135, 2015.
15. S. Ghanavati. *Legal-urn framework for legal compliance of business processes*. PhD thesis, University of Ottawa, 2013.
16. S. Ghanavati, M. van Zee, and F. Bex. Argumentation-based methodology for goal-oriented requirements language (grl). In *Proceedings of the 10th International iStar Workshop (iStar'17)*, volume 1829 of *CEUR Workshop Proceedings*, pages 97–102, 2017.
17. 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.
18. T. F. Gordon. Visualizing carneades argument graphs. *Law, Probability & Risk*, 6(1-4):109–117, 2007.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.

24. Y. Liu, Y. Su, X. Yin, and G. Mussbacher. Combined goal and feature model reasoning with the user requirements notation and jucmnav. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 321–322. IEEE, 2014.
25. J. G. March. Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, pages 587–608, 1978.
26. 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.
27. 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.
28. S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9):901–934, 2009.
29. 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.
30. G. Mussbacher and D. Amyot. Goal and scenario modeling, analysis, and transformation with jUCMNav. In *ICSE Companion*, pages 431–432, 2009.
31. B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.
32. 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.
33. C. Reed and G. Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools*, 13(04):961–979, 2004.
34. 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.
35. 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.
36. UCI. Design Prompt: Traffic Signal Simulator. `http://www.ics.uci.edu/design-workshop/files/UCI_Design_Workshop_Prompt.pdf`. Accessed: 2016-12-27.
37. 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.
38. 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.
39. 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.
40. 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.
41. D. Walton, C. Reed, and F. Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.
42. D. N. Walton. *Practical reasoning: goal-driven, knowledge-based, action-guiding argumentation*. Rowman & Littlefield, 1990.
43. 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.
44. 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)

C From RationalGRL to GRL - Examples from the Tool

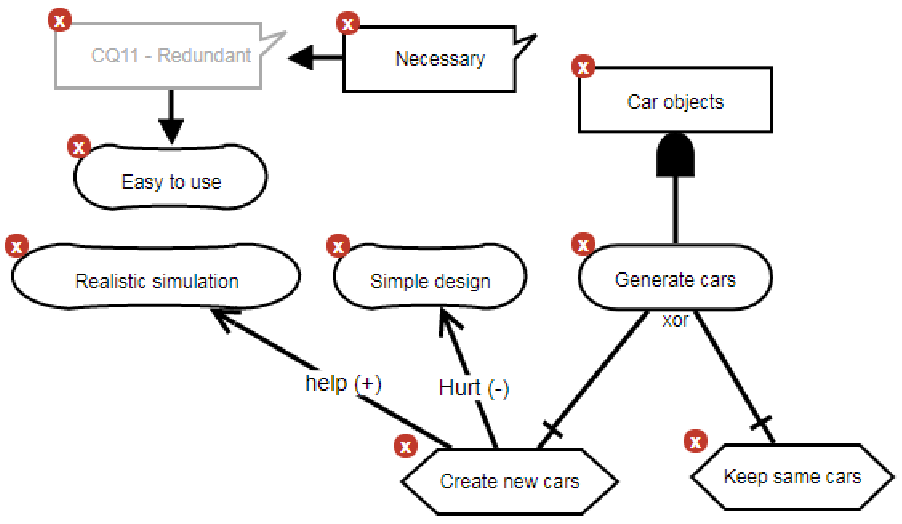


Fig. 21: RationalGRL model of Figure 13 modeled in the RationalGRL tool

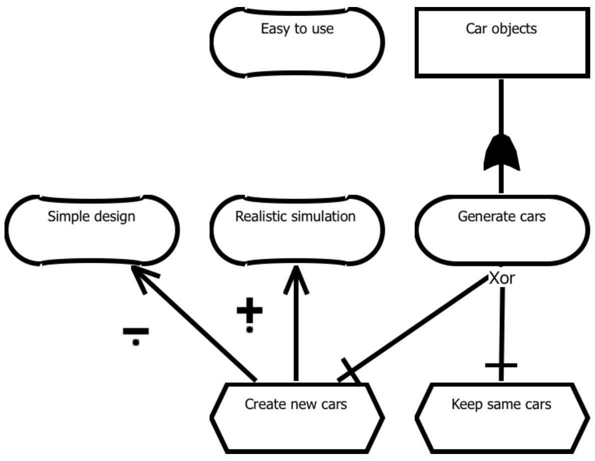


Fig. 22: Exporting the model Figure 21 to GRL gives the the above model in jUCMNAv

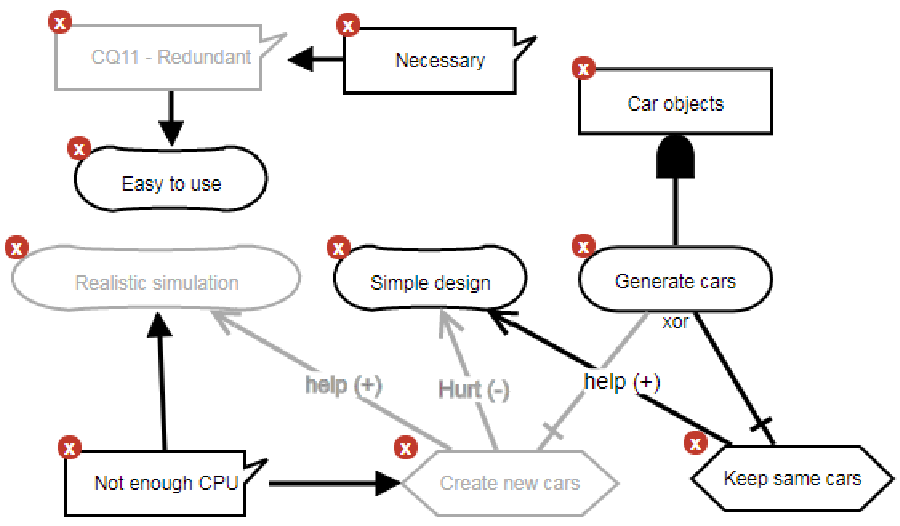


Fig. 23: Adding an argument that attacks multiple GRL elements to Figure 21

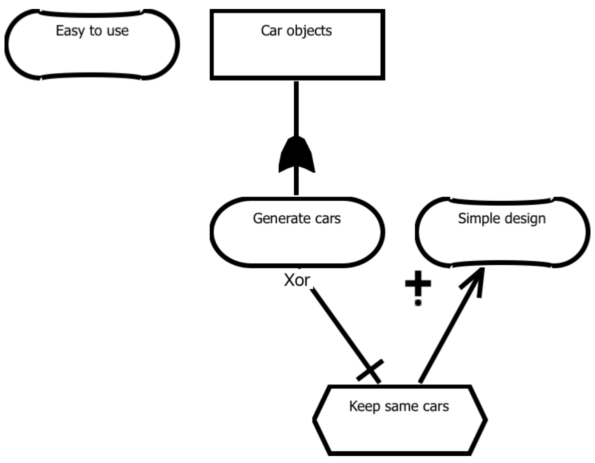


Fig. 24: Exporting the model Figure 23 to GRL gives the the above model in jUCMNAv