RationalGRL: A framework for argumentation and goal modeling

Marc van Zee [a,*], Floris Bex [b,c] and Sepideh Ghanavati [d]

[a] *Brain Team, Google Research, The Netherlands*
E-mail: marcvanzee@google.com

[b] *Department of Information and Computing Sciences, Utrecht University, The Netherlands*

[c] *Department of Law, Technology, Markets and Society, Tilburg University, The Netherlands*
E-mail: F.J.Bex@uu.nl

[d] *School of Computing and Information Science (SCIS), University of Maine, ME, USA*
E-mail: sepideh.ghanavati@maine.edu

Abstract. Goal-oriented requirements modeling approaches aim to capture the intentions of the stakeholders involved in the development of an information system as goals and tasks. The process of constructing such goal models usually involves discussions between a requirements engineer and a group of stakeholders. Not all the arguments in such discussions can be captured as goals or tasks: e.g., the discussion whether to accept or reject a certain goal and the rationale for acceptance or rejection cannot be captured in goal models. In this paper, we apply techniques from computational argumentation to a goal modeling approach by using a coding analysis in which stakeholders discuss requirements for a Traffic Simulator. We combine a simplified version of a traditional goal model, the Goal-oriented Requirements Language (GRL), with ideas from argumentation on schemes for practical reasoning into a new framework (RationalGRL). RationalGRL provides a formal semantics and tool support to capture the discussions and outcomes of the argumentation process that leads to a goal model. We also define the RationalGRL development process to create a RationalGRL model.

Keywords: Goal modeling, argumentation, goal-oriented requirements engineering, practical reasoning

1. Introduction

Requirements Engineering (RE) is about producing a consistent and comprehensive set of requirements of an information system, such as general functional requirements, operational environment constraints, and 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 [51]. These activities fall under the umbrella of goal modeling approaches. There are a large number of established RE methods using goal models in the early stage of requirements analysis [28], and a number of goal modeling languages have been developed in the last two decades, with $i*$ [51] being a very influential example and the Goal-oriented Requirements Language (GRL) [2] being a fairly recent extension of $i*$.

---

*Corresponding author. E-mail: marcvanzee@google.com.

A goal model is the result of a discussion process. Goal models for complex information systems – e.g., air-traffic management systems, systems that support industrial production processes, government or healthcare services – are constructed over the course of several workshops with stakeholders and requirements engineers. Failing to record the discussions underlying these complex goal models in a structured manner may harm the success of the RE phase of the system development process. The challenge here is that goal modeling is a *dynamic process*: models continuously change and evolve, and stakeholders change their opinions in between sessions. If the rationales behind goal model revisions are not properly documented, alternative ideas and opposing views that could potentially lead to different goal models will be lost. The problem is further exacerbated if the stakeholders constructing the goal model change, since modeling decisions made by one group of stakeholders may conflict with the opinions of another group of stakeholders. The disconnect between goal models and their underlying beliefs and opinions may thus lead to a poor understanding of the problem and its solution, which is an important reason for an RE project to fail [12].

Current goal modeling languages have limited support for capturing and reasoning about changing beliefs and opinions: the resulting goal model only shows the end product of a long process and not the discussions of the stakeholders during the modeling process, and with existing goal modeling languages one might change 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. 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 to clearly link the rationales to the elements of the resulting goal model. In order to do this, we propose a framework with its tool support which combines traditional goal modeling approaches with techniques from *computational argumentation* [7]. We have identified **five important requirements** for our framework:

(1) The argumentation must capture the actual discussions of the stakeholders 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 on the goal model, and vice versa.
(4) A process explaining how practitioners can develop models in the framework must be identified.
(5) The framework must have accessible tool support.

Based on our previous work [48,49], we develop the *RationalGRL* framework, which extends an existing goal modeling approach, the Goal-oriented Requirements Language (GRL) [2], with ideas from (computational) argumentation, namely *critical questions* and *argumentation semantics*. Critical questions find their origins in the literature on argument schemes [8,50]), which represent (stereotypical) patterns of non-monotonic reasoning. Part of each scheme are critical questions, which point to typical sources of doubt or (implicit) assumptions in arguments based on schemes. These questions can then be used to make explicit the counterarguments to the original argument. Such counterarguments can be attacked in turn by their own counterarguments and thus an argumentation framework [13] of mutually attacking arguments can be constructed. Given such an argumentation framework, the acceptability of sets of arguments can be computed using argumentation semantics [13]. Thus, computational argumentation based on schemes and critical questions can guide users in systematically deriving conclusions and making assumptions explicit [8,35].

Documenting choices and considerations in software or systems design using argumentation has a long tradition. Already in 1970, the first article on Issue-based Information Systems [29] talked about "argumentative processes" in information system design. IBIS was further developed in the 80's and 90's in the influential work on issue-driven design methods such as gIBIS [11] (see also [34,42] for further work on arguments or rationales in software design). For reasons unknown, this work was largely ignored by the goal-modeling community, which is why we propose our RationalGRL framework that incorporates the core ideas from issue-driven design methods into goal-oriented requirements engineering by explicitly including arguments for and against the various options into a goal model.

We first extend existing work on modelling goals and arguments by developing a list of critical questions that can be used to analyze and guide stakeholders' discussions about goal models. Inspired by the work on practical reasoning from computational argumentation, most notably Atkinson and Bench-Capon [4], we performed a coding analysis in which we analyzed a set of transcripts containing more than 4 hours of discussions among designers of a traffic simulator information system, finding 222 instances of arguments and critical questions about goals and tasks (Requirement 1).

We then specify clearly in what way RationalGRL extends GRL by developing a metamodel, which specifies the traceability links between the arguments based on the schemes and critical questions and the GRL models (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 to which we can apply argumentation semantics [13]. We, then, formally capture the links between argumentation and goal models as a set of algorithms for applying argument schemes and critical questions about goal models (Requirement 3).

In order to support practitioners in using the RationalGRL framework, we propose the RationalGRL development process, which explains steps for developing goal models and posing arguments based on critical; questions in an integrated way (Requirement 4). We implement and briefly evaluate a web-based tool for the RationalGRL framework, which works on any modern browser and is developed in Javascript (Requirement 5).[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 coding analysis 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 an overview of the RationalGRL framework and the RationalGRL metamodel, and examples from the coding analysis 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 development process and explains various features of our tool. Finally, Section 8 covers related work, future work, and the conclusion.

2. Background: Goal-oriented requirements language and practical reasoning

In this section, we introduce our running example, after which we give a brief overview of the Goal-oriented Requirements Language (GRL) [2], which is the goal modeling language we use to integrate with the argumentation framework. We consider a slightly simplified version of GRL, which includes

---

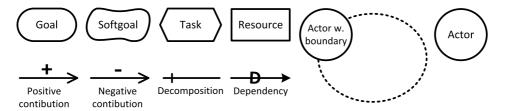[1]The tool and source code can be found at http://www.rationalgrl.com.

Fig. 1. Basic elements and relationships of GRL.

all the core elements of GRL but, for example, has only one level of contribution strength, whereas GRL contains different levels of positive and negative contribution. This simplification does not in any way influence the coding analysis. Furthermore, extending the GRL part of RationalGRL to include the full set of GRL elements is straightforward. Next, we discuss the ideas behind computational argumentation. In particular, we focus on the *practical reasoning argument scheme (PRAS)* [4], an argument scheme that is used to form arguments and counter-arguments about situations involving goals, and we give informal examples of how argument and counterargument can influence the status of beliefs about goals.

2.1. Running example: Traffic simulator

Our examples and coding analysis are based on the data produced by a series of experiments by Tang et al. [45], who in turn base their work on the so-called Irvine experiment [46]. This experiment contains a 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 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. Tang et al. asked designers (groups of students) to discuss the requirements of this traffic simulator. These discussions were recorded and transcribed. We performed a coding analysis on these transcripts, which we used to develop our RationalGRL framework. Furthermore, we also use the traffic simulator case as a running example throughout this paper.

2.2. Goal-oriented requirements language (GRL)

GRL is a modeling language for specifying intentions, business goals, and non-functional require-ments of multiple stakeholders [2], based on $i*$ [51] and the NFR Framework [10]. 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 alterna-tives 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 intentional elements (IEs) and relationships used in this paper are shown in Fig. 1.

Figure 2 illustrates a simplified GRL diagram from the traffic simulator design exercise. An actor rep-resents a stakeholder of a system or the system itself (*Traffic Simulator*, Fig. 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
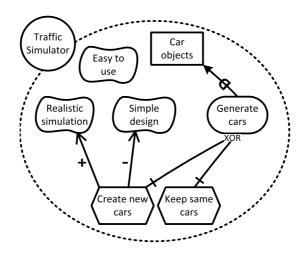
Fig. 2. Partial GRL model of the traffic simulator example.

a goal is quantifiable. Softgoals (e.g. *Realistic simulation*) are often related to non-functional require-ments, whereas goals (such as *Generate Cars*) are related to functional requirements. Tasks represent solutions to (or operationalizations of) goals and softgoals. In Fig. 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 or-der 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 (eXclu-sive OR) decomposition links allow an element to be decomposed into sub-elements. In Fig. 2, the goal *Generate cars* is XOR-decomposed to the tasks *Create new cars* and *Keep same cars*, as they are al-ternative 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 our current purpose. Dependency links are relationships between intentional elements, which can model dependencies between actors. Here, the goal *Generate cars* depends on the resource *Car objects*.

The GRL model in Fig. 2 shows the softgoals, goals, tasks and the relationship between the different intentional elements in the model. However, the rationales and arguments behind certain intentional elements are not shown in the GRL model. Some of the questions that might be interesting to reason 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 Fig. 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 so-called *belief* elements can be used to capture such design rationales that make later justification and review of a model easier. In the original GRL specification, however, the exact semantics of these belief elements and the ways in which they should be used remain unclear. Furthermore, belief elements cannot be attached to links between elements, which makes answering the fourth and fifth question above impossible. We further discuss belief elements in light of our proposed argumentation-based framework in Section 8.1.

2.3. Argumentation about goals and actions

An important characteristic of human reasoning is its non-monotonic, or defeasible, nature: conclusions that are accepted at one moment may be retracted later in view of new information; on the other hand, new information may cause conclusions that were rejected earlier to be reinstated. *Computational argumentation* is an approach to non-monotonic reasoning that focuses on interactions between arguments and counterarguments. With applications in multi-agent systems, decision-making tools, medical and legal-reasoning, argumentation has become a major subfield of artificial intelligence [7].

One central concept in argumentation is that of argument schemes [8,50]), stereotypical patterns of non-monotonic reasoning – for example, "if an expert says something, we can usually accept his claims". Part of argument schemes are the critical questions, which point to typical sources of doubt or implicit assumptions in arguments based on schemes – for example, "is the expert biased?" or "is the expert a recognized expert in the field that he claims?". Critical questions can be used to make explicit the counterarguments to the original argument – e.g. "we should not accept the expert's claim since he is biased". Such counterarguments can be attacked in turn by their own counterarguments – "no he's not biased since he has no stake in the case" – and thus an argumentation framework [13] of mutually attacking arguments can be constructed.

In early-phase requirements engineering, we can imagine many different argument schemes, such as for example the Appeal to Expert Opinion Scheme illustrated above. Central, however, in discussions surrounding goal models is obviously the reasoning about which goals to pursue and which actions to take, often referred to as *practical reasoning*. Practical reasoning has been studied extensively in computational argumentation, mainly by Atkinson and Bench-Capon [4], who have developed and formalized the Practical Reasoning Argument Scheme. A simplified version of this argument scheme, presented in syllogistic form, is as follows:

> *G* is a goal,
> Performing action *A* realizes goal *G*,
> <u>Therefore</u>
> 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*,
> <u>Therefore</u>
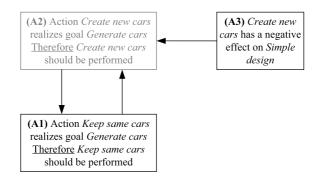> Action *Keep same cars* should be performed

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

Like in all argumentation, practical reasoning is also about counterarguments. 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 argument. Atkinson et al. [4] define a set of critical questions that point to typical ways in which practical reasoning arguments 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?

Answers to these critical questions are counterarguments to the original 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. Note that argument schemes and the arguments based on them can thus be of a more syllogistic form (i.e., "*G*, *A* realizes *G*, *Therefore A*") but also of a simpler, more propositional form (i.e., "*A* does not realize *G*").

Counterarguments *attack* the original arguments. Given a set of arguments and attacks between these arguments, we can compute which arguments are accepted and which are rejected using different argumentation semantics [13].[2] Figure 3 shows three arguments from the traffic simulation example, where arguments are rendered as boxes and attack relations as arrows. There are two arguments for a goal: 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). Thus, 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*, therefore, 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 Fig. 3, argument A3 is accepted because

---

[2]Formal definitions of argumentation frameworks and semantics will be given in Section 3.

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.

The practical reasoning scheme and its critical questions as given above could be used to argue about goal models. In fact, in some of our early work on RationalGRL [48,49] we used practical reasoning arguments such as A1 and A2 to capture reasoning about goals, combined with arguments based on other schemes such as the scheme from Expert Opinion. We then translated these arguments to goal models. However, our later coding analysis (Section 3) shows that when discussing requirements, people very often do not structure their reasoning nicely along the lines of the full argument schemes, that is, they rarely utter an argument along the lines "We have goal *G*, *A* realizes *G Therefore* we should perform *A*". A typical discussion is much more unstructured, as is clear from the transcript excerpts in Appendix B. Stakeholders propose and debate possible goals of the system, alternative options, and so on, but the argumentation is more about making different (propositional) claims, questioning these claims and proposing counterclaims than about building inference structures from premises to conclusions. Furthermore, it turned out that the stakeholders did not use many other types of argument or argument scheme, such as Expert Opinion. Hence, we decided to develop our own set of argument schemes and critical questions by coding the transcripts of discussions about the traffic simulator, so that we would meet Requirement 1: The argumentation techniques must capture the actual discussions of the stakeholders or designers in the early requirements engineering phase. Our set of schemes, questions and our coding analysis are described in the next section.

3. Argument schemes for goal modeling: A coding analysis

Recall that **Requirement 1** of our RationalGRL framework is that the argumentation 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 coding analysis to examine which types of discourse are used during discussions of system requirements, and how these discourse types can be captured as arguments and critical questions [41]. The type of coding we performed is *a priori coding*: we manually coded transcripts of discussions using a pre-existing list of argument schemes and critical questions based on GRL and the practical reasoning scheme. In this section, we present our coding analysis.[3]

In order to obtain actual requirements discussions, we turned to a recent series of experiments by Tang et al. [45]. 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 coding analysis.

Before coding the transcripts, we came up with an initial list of 11 argument schemes (AS0–AS10 in Table 1) based on GRL, practical reasoning, and an initial quick reading of some of the transcripts. As discussed in the previous section, we found that the discussion has a particular argumentative structure consisting of claims, questions and counterclaims. Hence, our argument schemes are not of the more traditional, syllogistic form but rather represent types of claims about a goal model.

---

[3]All original transcripts, codings, and models are available in our online repository, which can be found at http://www.rationalgrl.com.

Table 1

Occurrences of argument schemes AS and critical questions CQ in the three transcripts t_1, t_2 and t_3

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

AS0 to AS4 are schemes that concern a single element of a 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 about the links between GRL intentional elements. Our initial list also contained 18 critical questions, inspired by the questions associated with the original Practical Reasoning Argumentation Scheme [3]. 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 coding was performed by one author and subsequently checked by other authors. As the transcripts contain spoken language, the coding 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 5, 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 6, 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 7, 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 5, 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 sub-tasks, therefore, we added AS10 and CQ10a. Furthermore, since there were many discussions on the relevance and the clarity of
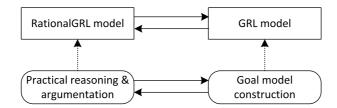
Fig. 4. The RationalGRL framework.

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 5 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 7 (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 we 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 coding analysis 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 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 RationalGRL 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 coding analysis.

4.1. The RationalGRL critical questions and argument schemes for practical reasoning

A core aspect of the RationalGRL framework are the argument schemes – in particular the critical questions – 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. Again, we note that the argument schemes are not of the traditional, syllogistic form, but rather represent types of claims about goal models. These schemes could be rendered in a syllogistic form: for example, we could say that the full reasoning encapsulated by scheme AS0 is something like "*a* is an actor in the information system *Therefore a* should be included as an actor in the goal model".

It is important to note that the list we provide here is not exhaustive – it is 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 8.2 for more details.

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. 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 5 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 Fig. 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.

Table 2

List of argument schemes (AS0–AS12, Att), 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

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

- *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 Fig. 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.
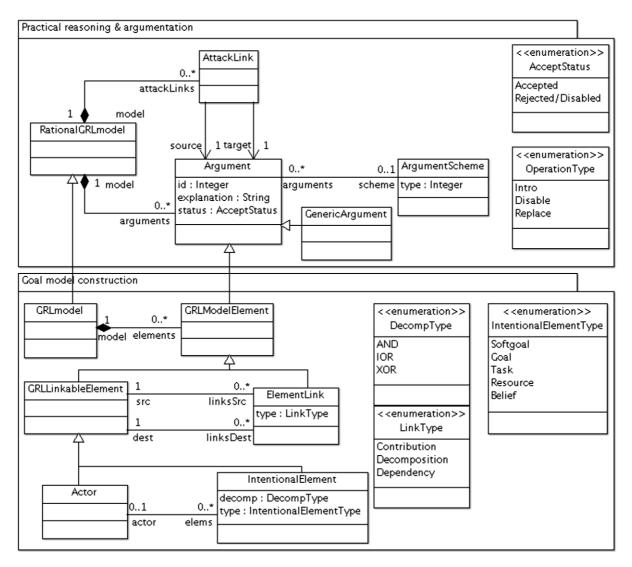

Fig. 6. The RationalGRL metamodel. The *goal model construction* package (bottom) is the GRL metamodel. The *practical reasoning & argumentation* package is the RationalGRL extension.

The complete metamodel of the language can be found in Fig. 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. Fig. 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, and by instantiating such a scheme we obtain an Argument. Therefore, each argument is associated with at most one scheme, but a scheme can be instantiated in multiple ways. A RationalGRLmodel is composed 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. 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 Fig. 4, the class of RationalGRLmodel is a superclass of GRLmodel: besides arguments about GRL elements, we can also have arguments that do not relate to any of the GRL elements.

4.3. From practical reasoning to RationalGRL models: Examples from the coding analysis

We now discuss the interactions between the *practical reasoning & argumentation* (bottom left element of the framework in Fig. 4) on the one hand, and *RationalGRL models* (top left element of the framework in Fig. 4) on the other hand. We provide informal examples of the links between the practical reasoning found in our coding analysis transcripts and RationalGRL models – the formal grounding for this connection can be found in the RationalGRL metamodel (Section 4.2) and 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 first show how instantiating argument schemes leads to the introduction of new RationalGRL elements in a model. Take the example in Fig. 7, which is based on the excerpt from transcript t_3 shown in Table 7 (Appendix B). On the left side, the arguments found in the transcript are shown, together with the argument scheme they are based on. The participants in the discussion argue that 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
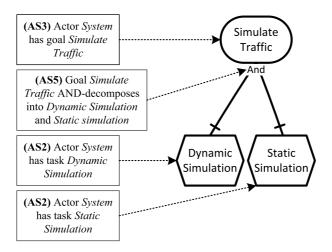
Fig. 7. RationalGRL model (right) with instantiated argument schemes (left): introducing new elements (operation INTRO).
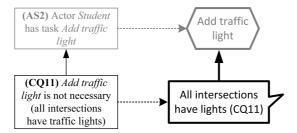

Fig. 8. Disabling elements (operation DISABLE).

introduced. These GRL elements are shown on the right side of Fig. 7; the dashed arrows indicate the links between the practical reasoning and argumentation on the left and the RationalGRL model on the right.

Example 2: Disabling GRL elements by answering critical questions (DISABLE). The excerpt from transcript t_1 used in this example is shown in Table 5 (Appendix B). Here, participants first sum up functionality of the traffic simulator, which can be captured as instantiations of AS2. On left side of Fig. 8 one such instantiation is shown, which leads to the addition of the task *Add traffic light* in the Rational-GRL model on the right side of Fig. 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 Fig. 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-hand side of Fig. 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 7 in Appendix B. It consists of a discussion about the type of decomposition relationship for the goal *Simulate Traffic* (Fig. 9). Recall that in Example 1,
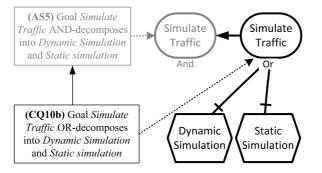
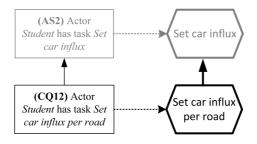Fig. 9. Replacing an element (operation REPLACE).

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

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 6 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*" (Fig. 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*.

Example 5: Defending the addition of an actor (DISABLE). The excerpt from transcript t_3 used in this example is shown in Table 8 in Appendix B. Actor *Development Team* is introduced with an argument based on AS0 (Fig. 11). This is then attacked by arguing that the professor will develop the software, thus, 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.
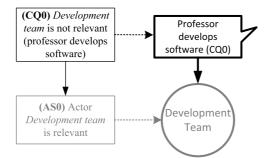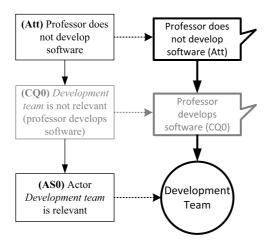
Fig. 11. Disabling an element (operation DISABLE).

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

5. RationalGRL: Logical framework

In Section 4, we have shown through a language definition and informal examples from our coding analysis that it is possible to trace elements of a 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 a 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 the RationalGRL framework. 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 (Fig. 6). We first provide a formal specification of a GRL model in Section 5.1, and we extend it 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 Rational-GRL 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.

5.1. Formal specification of GRL

We formalize a GRL model based on the metamodel (Fig. 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* ∪ *LinkTypes* ∪ {*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* ∈ *IETypes* specifies the type of the element,
- *name* ∈ *Names* is a string description of the element,
- *decomptype* ∈ *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 the notation and improve readability.

Throughout this section, we use the example in Fig. 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.
- *name* ∈ *Names* is a string description of its name.

A set of actors is denoted by *Act*.

---

[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.

We can formalize the actor in our example (Fig. 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 (Fig. 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 Fig. 2 is as follows.

$$IE = \big\{ (1, \textit{Resource}, \text{Car objects}),$$

$$(2, \textit{Softgoal}, \text{Realistic simulation}),$$

$$(3, \textit{Softgoal}, \text{Simple design}),$$

$$(4, \textit{Softgoal}, \text{Easy to use}),$$

$$(5, \textit{Goal}, \text{Generate cars}, \textit{XOR}),$$

$$(6, \textit{Task}, \text{Create new cars}),$$

$$(7, \textit{Task}, \text{Keep same cars}) \big\}$$

$$Act = \big\{(0, Actor, \text{Traffic Simulator})\big\}$$

$$R_{ActIE} = \big\{(ActIE, 0, 1), \dots, (ActIE, 0, 7)\big\}$$

$$Link = \big\{(8, Dep, 4, 1)$$

$$(9, PosContr, 6, 2)$$

$$(10, NegContr, 6, 3)$$

$$(11, Decomp, 6, 5)$$

$$(12, Decomp, 7, 5)\big\}$$

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*[5] 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: $\forall(ActIE, i, j) \in R_{ActIE} : \exists act_i \in Act \land \exists ie_j \in IE$.
(3) An intentional element belongs at most to one actor: $\forall ie_i \in IE : |\{(ActIE, j, i) \in R_{ActIE}\}| \leqslant 1$.
(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 Fig. 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 Fig. 5), namely *generic arguments* and *attack links*. We illustrate the new elements using the RationalGRL model in Fig. 13, which is an extension of Fig. 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,

---

[5]Note that *validity* is a bit of an overloaded term and has different meanings in different areas of research. In our setting, a valid model simply means that certain criteria are satisfied. It does not imply any real world "validation", in the sense that the model makes sense in the "real world".
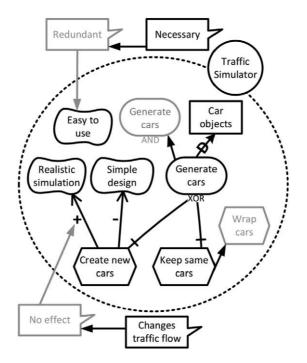
Fig. 13. Example RationalGRL model (extension of Fig. 2).

- *type = GenArg* states that the tuple is a generic argument,
- *name* ∈ *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 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 (Fig. 6) in which the class Argument is a superclass of GenericArgument and GRLModelElement. In sum, we define an argument as any one of the GRL IEs or links, or a generic argument. This corresponds with the argument schemes in Table 2: an argument that is a GRL model element, can be seen as a claim that this element should be part of the model.

Examples of arguments in Fig. 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 (Fig. 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 *Arg* (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 Fig. 13. Let us first enumerate all the arguments used in this example:

$A_0 = (0, Actor, \text{Traffic simulator})$

$A_1 = (1, Task, \text{Car objects})$,

$A_2 = (2, Softgoal, \text{Realistic simulation})$,

$A_3 = (3, Softgoal, \text{Simple design})$,

$A_4 = (4, Softgoal, \text{Easy to use})$,

$A_5 = (5, Goal, \text{Generate cars}, AND)$,

$A_5' = (5, Goal, \text{Generate cars}, XOR)$,

$A_6 = (6, Task, \text{Create new cars})$,

$A_7 = (7, Task, \text{Wrap cars})$,

$A_7' = (7, Task, \text{Keep same cars})$,

$A_8 = (8, Dep, 4, 1)$,

$A_9 = (9, PosContr, 6, 2)$,

$A_{10} = (10, NegContr, 6, 3)$,

$A_{11} = (11, Decomp, 6, 5)$,

$A_{12} = (12, Decomp, 7, 5),$

$A_{13} = (13, GenArg, \text{Redundant}),$

$A_{14} = (14, GenArg, \text{Necessary}),$

$A_{15} = (15, GenArg, \text{No effect}),$

$A_{16} = (16, GenArg, \text{Changes traffic flow}),$

$A_{17} = (ActIE, 0, 1), \ldots, A_{23} = (ActIE, 0, 7)$

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

$$Arg = \big\{A_0, \ldots, A_5, A_5', A_6, A_7, A_7' \ldots, A_{23}\big\}$$
$$Att = \big\{\big(A_5', A_5\big), \big(A_7', A_7\big), (A_{13}, A_4), (A_{14}, A_{13}),$$
$$(A_{15}, A_9), (A_{16}, A_{15})\big\}$$

In Fig. 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 [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, grounded extension). Suppose an argumentation framework $AF = (Arg, Att)$, two sets of arguments $S \cup S' \subseteq Arg$, and some argument $A \in Arg$. Then:

- S *attacks* A if some argument in S attacks A,
- S *defends* A if for each B such that B attacks A, S attacks B,
- S is *conflict-free* if there are no $A_1, A_2 \in S$ such that $(A_1, A_2) \in Att$,
- S is *admissible* if S is conflict-free and defends each argument in it.
- S is a *complete extension* if it is an admissible set that contains all the arguments that it defends.
- S is the *grounded extension* if it is the minimal (with respect to $\subseteq$) complete extension.

Let us explain these definitions using the example argumentation framework, in Fig. 14, which is a subset of our RationalGRL model from Fig. 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}.[6] 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}.

---

[6]In the argumentation literature, argument A_{14} is often said to *reinstate* argument A_4.
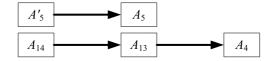
Fig. 14. Example argumentation framework, subset of the RationalGRL model from Fig. 13.

The notion of admissible sets gives rise to various possible extensions of an argumentation framework; we use the grounded extension, which is the minimal admissible set that contains all the arguments it defends all the arguments. In Fig. 14, there is one grounded extension, namely $\{A_4, A'_5, A_{14}\}$. Note that there are many more semantics and types of extensions in the literature. However, for current purposes the grounded semantics are sufficient, also because the algorithms in Section 5.4 do not generate mutually attacking arguments.

The status of arguments can now be determined based on the grounded extension. Recall from the metamodel (Fig. 6) that an argument can be *accepted* or *rejected*.

Definition 14 (Argument Acceptability). An argument is *accepted* w.r.t. an argumentation framework AF if it is in the grounded extension of AF, and *rejected* if it is not in the grounded extension of AF.

In Fig. 14, we have that arguments A_4, A'_5 and A_{14} are accepted and arguments A_5 and A_{13} are rejected. Returning to our larger example from Fig. 13, we can see that arguments A_5, A_7, A_{13} and A_{15} are rejected and all the other arguments are accepted.

5.3. Translating between RationalGRL and GRL

Now that we have formalized both GRL and RationalGRL models, 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.

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 an 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.

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 (grounded) extensions of the RationalGRL model (Definition 13). A GRL model is then generated from the grounded extension. 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

---

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**

Algorithm 2 RationalGRL to GRL translation

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

---

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 Fig. 13 to the GRL model in Fig. 2, where the latter is a translation of the former.

While we have defined a notion of a *valid GRL model* (Definition 7), we have not yet done the same for a RationalGRL model. 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.

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 subsections 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. We illustrate this with a single example algorithm for AS5.

We have slightly reworded critical question AS5 in Algorithm 3. 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 add the decomposition of goal "Generate cars", expressed as argument $A_5 = (5, Goal,$ 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 $AS_5(5,$ Keep same cars), and results in two new arguments: $A_7 = (7, Task,$ Keep same cars) and $A_{12} = (12, Decomp, 7, 5)$, which are both added to the set of arguments *Arg*.

The critical questions of type *INTRO* are all very similar, 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 3. This is the same for all the other critical questions of type *INTRO*. Therefore, they 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 the following algorithm, 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 4 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

---

Algorithm 3 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**

Algorithm 4 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**

---

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 Figs 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(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 4), 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 the grounded extension is $\{A_2\}$, which means that the resulting GRL model is empty, because generic arguments are not translated to GRL.

Algorithms for the remaining critical questions are omitted here since they are not very complex.

5.4.3. REPLACE algorithms

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

While the original critical question CQ5c is specific to the decomposition between a goal and a task, Algorithm 5 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 versions of the IE are not part of the grounded 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 Fig. 7. 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:

- $A_1 = (0, Goal, \text{Simulate traffic}, AND)$
- $A_2 = (1, Task, \text{Dynamic simulation}, AND)$

Algorithm 5 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**

- $A_3 = (2, \textit{Task}, \text{Static simulation}, \textit{AND})$
- $A_4 = (3, \textit{Decomp}, 1, 0)$
- $A_5 = (4, \textit{Decomp}, 2, 0)$

Suppose algorithm $CQ_{Decomp}(0, XOR)$ is executed for this RationalGRL model. On lines 2 and 3 a new argument $A_6 = (0, \textit{Goal}, \text{Simulate traffic}, OR)$ 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 5, 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 do not show it here.

6. The RationalGRL development process 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 development process (**Requirement 4**) and discussing a prototype RationalGRL tool (**Requirement 5**).

6.1. RationalGRL development process

We propose the development process shown in Fig. 15 to develop a RationalGRL model (see [20]). We assume that the initial GRL models have been created based on requirements specification documents. The rest of the steps are as follows:
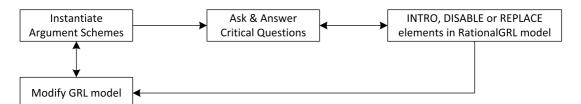
```
┌─────────────────┐     ┌─────────────────┐     ┌──────────────────────────┐
│   Instantiate   │────▶│   Ask & Answer  │◀───▶│  INTRO, DISABLE or REPLACE │
│ Argument Schemes│     │ Critical Questions│     │ elements in RationalGRL model│
└─────────────────┘     └─────────────────┘     └──────────────────────────┘
        ▲                                                      │
        │                                                      │
        ▼                                                      │
┌─────────────────┐                                            │
│ Modify GRL model │◀──────────────────────────────────────────┘
└─────────────────┘
```

Fig. 15. The RationalGRL development process.

(1) Instantiate Argument Schemes (AS) – Here 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 RationalGRL Development Process.

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 an existing tool support (see Section 8), ii) it allows us to conduct user studies in the next section, exploring the usability and usefulness of the RationalGRL framework compared to other goal modeling languages, and iii) tool support is an excellent demonstration to show that our formal framework and algorithms actually work. In this section, we briefly highlight some
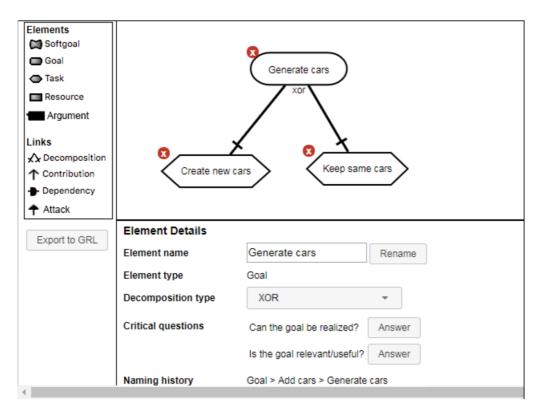
Fig. 16. Overview of the RationalGRL tool.

features of the tool, and we explain some of its current limitations. The tool can be accessed from: www.rationalgrl.com

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 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 Fig. 16. 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 16 shows the details pane for the goal 'Generate cars'. The user can change the name of the IE by 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

Fig. 17. Critical question details pane for "is the softgoal relevant/useful?".

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 pane can be used to answer the critical questions of the RationalGRL framework (Table 2): Fig. 16, 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). Consider Fig. 17, 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.

If in the example of Fig. 17, 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 (Fig. 18). This is an implementation of a DISABLE algorithm (Section 5.4.2) similar to Algorithm 4: 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, Fig. 19. A 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 Fig. 18, 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 (Fig. 19). 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, non-empty grounded extension (cf. Section 5.2) – if the user creates an attack cycle, an error message is shown.

Argument schemes and critical questions. The tool contains more argument schemes and critical questions than those that we initially collected in Table 2 – for example, we have added extra critical questions for contribution links from task to goal, which are not in Table 2. As mentioned before, our table is not meant to be exhaustive, and it is straightforward to add more argument schemes and critical questions if necessary.

Fig. 18. Effect of answering CQ11: "is the softgoal relevant/useful?" with "no".

Fig. 19. Attacking an argument with a generic argument "necessary".

Export to jUCMNav. GRL is implemented in the open-source tool jUCMNav [36].[7] jUCMNav is used actively in research, and the tool has been extended in many directions. 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, web-based is sufficient for our current aims, and may benefit the community in general. Our tool is thus not meant to replace jUCMNav.

The connection to jUCMNav is maintained in the form of an export function from the RationalGRL tool to jUCMNav, essentially an implementation of the RationalGRL to GRL translation (Algorithm 2). 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

---

[7]http://jucmnav.softwareengineering.ca/foswiki/ProjetSEG.

recommend using the Graphviz dot[8] 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 20 shows the model that was discussed earlier in this paper (Fig. 13) in the RationalGRL tool. Recall that translating this model to GRL provided us with the model in Fig. 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 Fig. 21, which is the same as Fig. 2.

Our translation and export function uses the argument acceptability as a way of determining the GRL model. Take for example, the RationalGRL model in Fig. 22. A positive contribution is added from "Keep same cars" to "Simple design". More importantly, 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 Fig. 22) are not included. This can be seen in Fig. 23, where the jUCMNav GRL model that was exported from the RationalGRL tool is shown. The pair of Figs 22 and 23 also nicely shows the added value of RationalGRL: Fig. 22 shows that there can be a larger discussion and rationalization underlying even a fairly simple goal model such as the one in Fig. 23.

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 Fig. 16, 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 (Fig. 16) without introducing new elements that attack the original ones (cf. Fig. 13, where the old decomposition type *AND* of "Generate traffic" is shown as an explicit argument).

7. RationalGRL evaluation

In addition to the coding analysis (Section 3), which forms the basis of the RationalGRL framework, we also performed a user evaluation with 16 expert users. The objective of this evaluation was to determine whether the users found the components of RationalGRL (arguments, argumentation semantics, critical questions) useful, and whether they found it easier to keep track of and express opinions and beliefs using RationalGRL compared to standard GRL. Regarding the usefulness of the RationalGRl features and the to GRL, the following two broad hypotheses follow from our main points in this paper:

H1 The components of RationalGRL – arguments, critical questions and determination of the status of arguments – are a useful extension to standard goal modeling in GRL.

H2 The components of RationalGRL make it easier to express, determine the effect of and communicate to other stakeholders one's opinions and beliefs about a goal model.

---

[8]http://www.graphviz.org/.

In the rest of this section, we will describe our experiment in more detail and further specify these hypotheses. In addition, we wanted to know some basics about the user experience. The aim was not to conduct a full user experience study for the RationalGRL tool, but usability could influence the participants ideas about the RationalGRL framework and language – a very nice and easy to use tool tends to make people more positive towards a particular modeling language, whereas irritations about a bad tool will lead to a more negative disposition.

7.1. Experiment design

The idea was to have our participants perform a small modeling task, and then ask them what they thought of RationalGRL. Thus, our experiment consists of three parts:

(1) Explanation: We explain to the participants the basics of the RationalGRL development process and the RationalGRL tool.
(2) Modeling: We ask the participants to model a summarized discussion in the RationalGRL tool.
(3) Survey: We present the participants with a survey containing questions about the features of RationalGRL, and some questions about the usability of the RationalGRL tool.

The instructions for these three parts are provided in a single document that we sent out to the participants.[9] The explanation (Part 1) starts with a general explanation of GRL and standard goal modeling similar to the explanation provided in Section 2.2, then briefly discusses the components of RationalGRL (argument, attack, critical questions) and finally the tool is briefly described (similar to Section 6.2). For Part 2, we provided a small section of a transcript and some context information (see Appendix D) and asked the participants to model this in the RationalGRL tool and take a screenshot of their final model. The idea is that the participants act as if they are "present" at a discussion about the early-stage requirements and are asked to model the goal model corresponding to these requirements.

In order to allow the participants to compare their models to standard GRL models, after the modeling exercise we provide the participants with Fig. 2, which models the discussion for Part 2 (Appendix D) in standard GRL.

For Part 3, we provided the participants with a survey.[10] The survey starts with general questions about the participants (years of experience etc.). The second part of the survey concerns the specific features of RationalGRL, asking participants to rate RationalGRL on a Likert scale from 1 (very useless) to 5 (very useful):

Q1 Do you think the arguments and counterarguments of RationalGRL are a useful extension to standard goal modeling?
Q2 Do you think the critical questions and answers in the details pane are a useful extension to standard goal modeling?
Q3 Do you think the automatic determination of the status of arguments and elements is a useful extension to standard goal modeling?

The following questions ask the participants to rate RationalGRL vs. a standard goal modeling language on a Likert scale from 1 (much more difficult) to 5 (much easier):

Q4 Do you think using RationalGRL instead of a standard goal modeling language makes it easier or more difficult to for someone to express beliefs and opinions in a goal model?

---

[9]See www.rationalgrl.com, page "Empirical Study".
[10]https://goo.gl/forms/fDSMUnAV20wy7kbY2

Q5 Do you think using RationalGRL instead of a standard goal modeling language makes it easier or more difficult to for someone to determine the effect of beliefs and opinions on the resulting goal model?

Q6 Do you think using RationalGRL instead of a standard goal modeling language makes it easier or more difficult to for someone who is not the original author to understand the goal model?

We also ask two open questions regarding the strengths and weaknesses of RationalGRL compared to GRL. The survey further contains questions about the RationalGRL tool user experience, such as whether it was easy or difficult to get started with the tool and to model the example case. We, furthermore, ask open questions about the strengths and weaknesses of the tool, possible improvements and whether the participant thinks the tool could be used in practice in future.

In order to test our main hypotheses H1 and H2, we have to formulate the appropriate null hypotheses and alternative hypotheses. For H1 and H2, we hypothesize that the participants will, on average, rate the components of RationalGRL as useful (rating 4) or very useful (rating 5) for Q1, Q2 and Q3, and as making it easier (rating 4) or significantly easier (rating 5) for questions Q4, Q5 and Q6. In other words, the null hypothesis and alternative hypothesis for H1 and H2 are as follows:

- $H1_0$ and $H2_0$: median rating for questions Q1–Q6 is 3 or lower.
- $H1_a$ and $H2_a$: median rating for questions Q1–Q6 is higher than 3.

Note that for each question (Q1–Q6), we can test the null and alternative hypothesis separately, giving insight into exactly which components of RationalGRL were deemed useful. We set the significance level for each hypothesis at $\alpha = 0.05$.

Participants. We asked 16 participants in our network to participate in the experiment. Most of the participants were therefore either employed in industry or staff and (ex-)students from university. For those working in industry, all participants took the experiment outside of working hours to avoid conflicts of interest. Two thirds of the participants had either a PhD or Master degree, 20% a Bachelor degree, and one respondent responded with "Other". All but one participant had a year or more experience with software development. The average experience was 6.2 years (standard deviation 6.7), with 10 participants having less than 5 years of experience, 6 participants having 8 or more years of experience and one participant having 25 years of experience. The participants also judged themselves to be quite competent in early-phase requirements engineering: on average they gave themselves a rating of 3.2 out of 5 (standard deviation 1.3) with half of the participants claiming they were at least "competent" (rating 4) or "very competent" (rating 5). However, experience with goal modeling languages was markedly less: the average rating was 1.9 (standard deviation 1.3), with 9 participants claiming never used a goal modeling language (rating 1), and only two participants displaying regular use (monthly, rating 4, or weekly, rating 5). The participants that had experience with goal modeling languages had mostly used i* (5 users), with 2 users being familiar with GRL and 2 users having used another goal modeling language.

7.2. Results

First, the participants had to perform the modeling task. They produced RationalGRL models containing on average three arguments (in addition to goals, tasks, etc.) from the short transcript. Some of the examples of RationalGRL models, created by the users, can be found in Appendix D.1.[11]

---

[11]The full results of the survey can be downloaded from www.rationalgrl.com, on the page "Empirical study".

Table 3

Participant ratings and statistical results of the usefulness of the components of RationalGRL

	Very useless	Useless	Neutral	Useful	Very useful	Median	Mode	*p*-value
Q1	0 (0.0%)	0 (0.0%)	3 (21.4%)	8 (57.1%)	3 (21.4%)	4	4	0.0005
Q2	0 (0.0%)	3 (21.4%)	3 (21.4%)	6 (42.9%)	2 (14.3%)	4	4	0.1133
Q3	0 (0.0%)	0 (0.0%)	2 (14.3%)	7 (50.0%)	5 (35.7%)	4	4	0.0002

Table 4

Participant ratings and statistical results of whether the components of RationalGRL make reasoning about a goal model easier

	Very difficult	Difficult	Neutral	Easy	Very easy	Median	Mode	*p*-value
Q4	0 (0.0%)	0 (0.0%)	4 (30.8%)	4 (30.8%)	5 (38.5%)	4	5	0.0020
Q5	0 (0.0%)	0 (0.0%)	6 (46.2%)	5 (38.5%)	2 (15.4%)	4	3	0.0078
Q6	0 (0.0%)	1 (8.3%)	4 (33.3%)	7 (58.3%)	0 (0.0%)	4	4	0.0352

After the modeling task, the survey questions were asked. We first asked the participants some questions about user experience. The users where generally positive about RationalGRL's clear User Interface (UI), the fact that the tool gives a nice overview of the goals and opinions in a case and that it was straightforward to use and understand. With respect to weaknesses and improvements to the RationalGRL tool, the users mentioned additional UI functionalities such as the possibility to save models, having an "undo" function, and flipping the arrow after adding them. Users also suggested various language-related improvements. Some users mentioned they missed the possibility to attack links, while others mentioned that not all GRL elements are supported (for instance, it is currently not possible to add actors).

The next set of questions concerned the specific features of the RationalGRL modeling language and the difference with standard goal modeling languages. Table 3 shows the respondents' answers to questions Q1–Q3, and Table 4 shows the answers to questions Q4–Q6. We concluded with two open questions about the comparison between RationalGRL and other goal modeling languages. When asked about the advantages of RationalGRL over standard goal modeling languages, many users agreed that making arguments explicit may force end users to have a more structured discussion: "Yes, having the possibility to add arguments seems quite useful", 'Generally I think its useful to explicitly document arguments of a discussion", "Clear communication about argumentation and forcing people to think in those clear terms.", "...you can add arguments and that you can answer questions that help you to develop arguments", "It's useful that discussion and explanation are close to the diagrams", "a way to see how decisions are being shaped.". Furthermore, one user stated that RationalGRL successfully "tries to capture the rationale behind the modeling process' and "has a simple way to compute the status of the arguments".

When asked about the weaknesses, users mentioned the complexity as the most important weakness: "The apparent increase in complexity might lead to negative perceptions", "adding yet another layer of complexity scares me". The concern that the modeling process may involve too much cognitive overhead was mentioned a few times though: "I think the overhead of inputting a (detailed) discussion in a structured manner into any system makes adoption difficult", "the manual input is too complex and takes too much time. An automated process of parsing the conversation log would be much more helpful", and "I still believe in the value of arguments, but there should be less confusing ways to capture them". One user mentioned that complexity and cognitive overload is a problem with goal modeling in general: "Goal models are already complex (...) I have worked for years on the effect of context on

goal models, and my conclusion is that this was very interesting academic work but with close-to-zero practical implications, unfortunately.".

7.3. Analysis of results

Hypothesis H1.   The first analysis concerns hypothesis H1, whether the components of RationalGRL – i.e. arguments, critical questions and determination of the status of arguments – are a useful extension to standard goal modelling in GRL. Table 3 summarizes the medians, modes, and *p*-values for a one-tailed single-sample sign test for the relevant questions Q1–Q3. We chose the simplest non-parametric test available so as to have to take as few assumptions as possible w.r.t. the distribution of answers for each question. As it is evident from the results, the participants were generally positive about the components of RationalGRL. For Q1 and Q3 the *p*-values are both significantly lower than $\alpha = 0.05$, which means that for these questions we can accept $H1_a$ and reject $H1_0$. This means that the addition of arguments and the use of arguments to determine the acceptability of elements of the goal model are deemed to be useful to very useful by the participants. As can be seen in Table 3: no participant found the addition of arguments and argument status useless or very useless, and 78.6% (Q1) respectively 85.7% (Q3) found the new components useful or very useful. Furthermore, these findings fit with the answers to the open questions (see Section 7.2). For Q2, the *p*-value indicates that the result is not significant with $\alpha = 0.05$, which means we cannot reject the null hypothesis $H1_0$. This means that the participants did not think the addition of critical questions like to the RationalGRL tool was significantly useful. Thus, we can say that the addition of critical questions is perhaps not as useful as we had hypothesized. This again fits the descriptive statistics in Table 3: 21.4% of respondents found critical questions "useless". Furthermore, in the open questions a number of participants indicated that the exact difference between some of the critical questions was unclear, and that some of the critical questions seemed unnecessary, at least for the small modeling exercise given in the experiment.

Hypothesis H2.   The second hypothesis states that components of RationalGRL make it easier to express one's opinions and beliefs about a goal model, determine the effect of one's opinions and beliefs about a goal model and communicate one's opinions and beliefs about a goal model to other stakeholders. Table 4 provides a summary of the statistical analysis of the relevant questions Q4–Q6. As can be seen, all the scores are significantly higher than the median of 3 for $\alpha = 0.05$, which means that we can accept $H2_a$ and reject $H2_0$ for all the separate questions Q4–Q6. In other words, the fact that the participants rate the components of RationalGRL as making it, on average, easier to reason with their beliefs and opinions about a goal model is very unlikely to be due to chance.

With respect to Q4, participants stated that arguments make it much easier to express their opinions. None of the participants found expressing their beliefs (much) more difficult, and 69.3% found it easier or much easier. This corresponds to the answers to Q1, where participants said the arguments were a useful addition. For Q5, participants agree that RationalGRL makes it easier to determine the effect of one's beliefs on the goal model. Combining this with the results for question Q3, which asked about the usefulness of determining the status, we can say that the participants overall found this a useful feature which makes working with conflicting beliefs easier. One participant remarked that "The reasoning [in the example] seems simplistic [...] the added value of [the formal argumentation] would be justified if the reasoning is more complex". Thus, the simplistic case in the modeling exercise possibly detracts from the results for Q5 and Q3. Finally, participants were less sure that arguments would help in communicating the goal model to others. This may be due to the fact that in the experiment, communication of opinions to others was not really tested, leaving the participants to guess what this effect would be.

7.4. Threats to validity

Internal validity is about the validity of the experiment results given our experiment setup and interpretation of findings. The main threat here is that the modeling task the participants were asked to do is not realistic. The task is quite small, and based on an existing transcript of a requirements discussion. In a realistic setting, users of RationalGRL would model much more complex requirements together with the stakeholders. The problems with complexity and cognitive overload only start to play a role for larger models, so while the participants were generally positive in the ranking questions Q1–Q6, some reservations were expressed in the open questions as to whether this approach would work in a realistic setting.

Another threat to internal validity is that many of the participants had no experience with goal modeling languages at all. Hence, the perceived usefulness of the arguments could be influenced by the fact that the participants, for the first time, were given a goal modeling language to perform structured requirements analysis. In other words, perhaps they found a structured modeling language generally quite useful. One way to better check the difference between standard goal modeling and RationalGRL would be to first provide users with a standard goal modeling tool and then have them use RationalGRL. An even more thorough way to test the usefulness of RationalGRL over, for example, standard GRL would be to compare a test and control group, where the test group uses RationalGRL and the control group uses standard GRL. We leave these types of further empirical studies for future research.

External validity concerns the generalizability of the results. As the number of participants is fairly small (16), this threatens the generalizability of our findings. However, the type of participants is realistic: software engineers having a lot of experience with requirements engineering, as opposed to, for example, engineering students with little experience. Furthermore, the results are in line with existing research on design rationale. On the one hand, these existing studies, like our study, show that structured modeling languages suffer from the common problem of high cognitive overhead [43]. One the other hand, these studies show that the inclusion of argumentation and critical questions in design and requirements discussions improve the reasoning [39,45].

7.5. Discussion

Our results point in some interesting directions for further discussion and research.

(1) *Arguments and automatically determining acceptability are useful.* Participants were enthusiastic about adding arguments to a goal modeling language, and they believed the grounded argumentation semantics in the tool are intuitive. This is a positive signal for the addition of techniques from computational argumentation to goal modeling languages: just adding arguments as, for example, labels on goals and tasks does not allow one to compute the status of elements in a goal model in the way our approach does. Furthermore, automatically determining the status of elements will have a bigger impact when working with larger, more complex goal models, for which manually determining the impact of arguments is not feasible.

(2) *Critical questions need to be clear and have a prominent place in the process of building a goal model.* While participants overall found the critical questions (CQ's) useful, their impact was markedly less than the addition of arguments. This can have various causes. Participants mentioned they did not use the details pane of the tool, which can be used to ask and answer critical questions, that much. Furthermore, it was also mentioned that the difference between different questions was not always clear. Finally, the experiment task, modeling an existing discussion,

may also have influenced the CQ's perceived usefulness. The CQ's are meant to be asked during the actual RationalGRL development process in which goal models are constructed and critically analyzed from scratch.

(3) *High cognitive overhead.* A concern that was raised often in the evaluation is that, in its current form, RationalGRL has a relatively high cognitive overhead. Goal modeling is by itself already a cognitively high-effort activity, and the fact that we add more elements to the language does not improve this.

Taking the above into account, it seems natural in future work to focus more on the RationalGRL development process.[12] Users like to be guided during modeling phase, and making arguments explicit in the modeling process was indicated as useful, but argumentation should be integrated into the *process* of goal modeling more than in the goal models themselves, keeping the goal models simple and thus lessening cognitive overhead. Critical questions are a natural fit for this, as they play a key role in the RationalGRL development process, but in its current form they are only mentioned in the details pane of the tool.

8. Discussion

8.1. Related work

Design rationale. Argumentation in software design has been the subject of research in so-called *design rationale* (DR) [34,42], which deals with an explicit documentation of the reasons behind decisions made when designing a system or software architecture. Already in 1970, the first article on Issue-based Information Systems ([29]) talked about "argumentative processes" in information system design. IBIS was further developed in the 80's and 90's in the influential work on issue-driven design methods such as gIBIS [11] (see also [34,42] for further work on arguments or rationales in software design).

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 [22,40] has been made, where argument diagrams represent reasoning from premises to conclusions. One of the problems of such methods and tools is the cognitive overload that results from having to learn and use such tools at the same time as having to discuss and think about complex software designs. Furthermore, not much of the research on design reasoning has empirically investigated which techniques and how much information designers use in design reasoning, which information is missing, and how the use of design reasoning techniques affects design discussions and eventually the quality of design discourse. More recent research on DR, therefore, moves away from the idea that all decisions have to be explicitly diagrammed and focuses more on empirically investigating how critical reflection can help when designing [39,45], or which parts of the design process are best explicitly documented [16].

Two relevant articles [9,17] capture software design processes as argumentation dialogues between parties. Black *et al.* [9] provide a more formal rendering of the initial informal ideas by Finkelstein and Fuks [17], and propose a simple argumentation dialogue protocol. Similar to us, they base their argumentation on an analysis of transcripts of software engineers arguing in the Traffic Signal Simulator task

---

[12]Note that in order to keep the study simple for the users, we did not explicitly ask the respondents to follow the development process from Section 6.1.

[46]. They provide different argumentative move types, such as propose and challenge (similar to our critical questions). This preliminary work showed how techniques from computational argumentation can be applied in software design discussions.

Software design and requirements engineering (RE) are very closely related [38] and hence, the insights from the DR literature are directly applicable to RE – although for reasons unknown, the DR literature has been largely ignored by the goal-modeling community. 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 a goal model. In this respect, we may seem to go against the trend in DR research to focus less on diagramming and more on guidelines for design processes. There are two reasons for this: first, as opposed to DR diagramming, goal modeling is still very much an active area of research [21] with clear practical applications [37]. Secondly, we note that RationalGRL also includes a development process and critical questions that encourage reflection when thinking about the possible goals and functionalities of a system, which connects with the research ideas on DR [39,45].

Requirements engineering.   There are numerous examples in the RE literature that mentions arguments, questions or conflicts in some way. Already in 1997, Smith [44] proposed a framework to develop arguments, which related a model of a proposed software system to a model of its environment, and showed that these together achieve properties described by a model of requirements. In a similar vein, Lockerbie *et al.* [30] describe procedures that analysts follow to use satisfaction arguments in combination with $i*$. Horkoff and Yu [25] introduce analysis procedures for the $i*$ goal-oriented framework, allowing users to ask questions such as "what if?", or "are certain goals achievable? how? or why not?". Finally, Hassin and Amyot [24] analyze the correctness of GRL models by performing statistical analysis on generated questionnaires, which allows for early detection of potential conflicts between stakeholders of the system. The mentioned work, however, does not consider arguments in a similar way as RationalGRL, that is, to include the explicit opinions of stakeholders.

A number of approaches in the field of requirements engineering take arguments into account explicitly. One early example comes from Haley *et al.* [23], 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 how 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 conflicts among arguments or determining the acceptability of arguments. Yu *et al.* [52] further extend the framework proposed by Haley *et al.*, including algorithms for Dung-style argumentation semantics [13] and a database including 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.* [52] Section 3.1).

Another recent example of the use of arguments in goal-oriented requirements engineering is the work by Murukannaiah *et al.* [35], who propose Arg–ACH, where 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, beliefs that are consistent and inconsistent with 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 a goal, which is, then, modeled using the Carneades tool [22]. 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 two groups, one with ACH and one with Arg–ACH, performed 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.

Another example of argumentation in RE concerns the use of argumentation in requirements elicitation. Ionita *et al.* [26] 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 [13] are then used to determine the risks that are still valid, and those that have been successfully rejected. El-rakaiby *et al.* [14] 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.

It is clear from these literature that arguments play a core role in RE. Murukannaiah *et al.* [35] 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.* [14] provide a case study similar to our research, identifying, many counterarguments specifically with respect to realizability, 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 [14,26,52].

The work on RationalGRL puts the insights from the literature in a broader framework. For example, there is a specific focus on security requirements [23,26,52] or the reasoning is about single goals or tasks instead of about the wider context as represented in a goal model [14,23,35,52]. 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 (Fig. 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 already been included implicitly in existing goal modeling languages. At its core, goal modeling is a process of argumentation: high-level goals are reasons for lower-level goals, tasks and resources. Hence, refinement and decomposition techniques used in requirements engineering [47] 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 for 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. 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 [27]), 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* (cf. Definition 14 given potentially contradictory opinions of stakeholders).

More explicit rationales for goal models are included in the original GRL specification [2] in the form of *belief* elements, which can be attached to other intentional elements (goals, tasks). The content of a belief is a textual description, which can explain, for instance, the rationale for including a goal in a model. In this regard, they seem like arguments or reasons for the different elements in a goal model. However, in the literature the precise usage and semantics of belief elements in GRL remain unclear: according to the metamodel [2], they can be connected to other intentional elements (including beliefs) through any type of link, but in jUCMNav tool, there is a separate *belief link* for connecting beliefs to goals and tasks only, and beliefs are not used in any of the algorithms to compute the satisfiability of goals and tasks. We are also not aware of many examples of the usage of belief elements in the literature. Thus, GRL belief elements can be understood as metadata that can be attached to an element.

There are further advantages of using RationalGRL over GRL with belief elements. For example, none of the GRL specifications allows to attach beliefs to other elements, thus it is not possible, for example, to provide a rationale for why some tasks positively contributes to a goal. Furthermore, the RationalGRL arguments follow clear schemes with associated critical questions explicitly tailored towards reasoning about goals and actions or tasks, while beliefs in GRL offer no further guidance as to what kinds of rationales can be given for elements in a goal model. In addition, beliefs in GRL seem not to influence a goal model in any way. Arguments in RationalGRL, in contrast, play an important role in forming the resulting goal model: if an element is defeated by an argument, it is not part of the resulting goal model.

Goal modeling and argumentation.   There are several contributions to the literature that explicitly relate argumentation-based techniques with goal modeling. Bagheri and Ensan [5] and Mirbel and Villata [32] 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 [13]. For example, if in a 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 the computation of consistent sets of goals and tasks in a goal model. In RationalGRL, we have opted not to provide such an argumentation-theoretic semantics to goal models, because GRL already has quite fine-grained satisfiability semantics [2], which take into account conflicts, decompositions and dependencies. RationalGRL focuses on what is not captured in the work discussed above [5,32], 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 [5,32] for goal models, as the elements of a goal model are already arguments in RationalGRL (Fig. 6).

The contribution most closely related to ours is the work by Jureta *et al.* [27]. Jureta *et al.* propose a "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 Development Process in Fig. 15 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 for conclusions are given as formal structured arguments.[13] 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. In addition, 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 [48,49], we extended Jureta *et al.*'s work and translated argument diagrams to GRL models (an automatic translation tool is discussed in [49]). 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.

8.2. Future work

RationalGRL provides a basic framework for argumentation in requirements engineering, and all aspects of this framework are intended to be fully extensible. We see many 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 [35], discussions about security requirements

---

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

[23,26,52], architectural principles [31], legal compliance [19], and so forth. From a knowledge engineering perspective, including 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.* [8], critical questions for argument schemes 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 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 do not include the possibility of expressing preferences between mutually inconsistent arguments. Using the work by Modgil [33], it is possible to provide explicit arguments for preferences, thus allowing us to make a reasoned choice between two options. Furthermore, it would be interesting to investigate the link between RationalGRL, arguments and argumentation frameworks about goal models, and formal argumentation dialogues from the literature. Black et al. [6] have already shown that discussions can be captured as such dialogues, and it would be interesting to look which types of arguments or argument schemes correspond to their different move types in design discussions.

Belief revision. An interesting avenue for further investigation is the theoretical foundation of RationalGRL. Since it is one of our main goals to develop a framework that can be used by practitioners, we developed a formalism that is geared towards implementation using algorithms and a compact propositional logic. However, there is a long strand of research in Artificial Intelligence focusing on developing postulates that characterize how an agent can revise its knowledge base in a rational way [15]. The most famous postulates are the so-called AGM postulates [1]. It would be very interesting to study whether (a variation of) these rationality postulates can be used to characterize the algorithms we have developed in this article, and more generally, what rationality means in our setting. While there is formal work in Artificial Intelligence on the connections between argumentation and belief revision (see e.g. [15], there is very little work on applying the combination to requirements engineering. The only work we are aware of that deals with his subject is by Gervasi and Zowghi [18], who use a logical framework for belief revision to detect inconsistencies in software specifications and then revise those specifications so that the inconsistencies are removed. They do not explicitly connect their framework to goal models, however, and do not connect their framework to specific argumentation using schemes in the way RationalGRL does.

Process and 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 new features described above, such as new argument schemes and reasoning with preferences.

The user evaluation of the tool revealed that in order to implement the tool in realistic settings, we need to incorporate arguments in the goal modeling process in a more collaborative way. Extending the tool

for collaborative online goal modeling, similar to Github is one of the possibilities: 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. In addition, 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 dialogues between parties [9,17], which are natural fit with the question-answer format employed in the RationalGRL framework.

Empirical validation. We have performed a coding analysis and found more than 200 occurrences of arguments and questions. In this study, we performed a simple experiment with 16 participants[14] to evaluate the usability and usefulness of RationalGRL compared to other goal modeling languages. However, what is still lacking is an industrial-strength case study testing extensively whether, and how, the tools and techniques provided by the RationalGRL framework improve early phase requirements engineering. The complexity of the domain makes focused experiments difficult, but not impossible. For example, both Murakannaiah *et al.* [35] and Tang *et al.* [45] provided a realistic but bounded problem for their subjects to solve, and similar experiments are imaginable to evaluate 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 in which multiple individuals have to work asynchronously might benefit from the extra tools offered by RationalGRL. Furthermore, the outcome also depends on which sets of tools and techniques offered by RationalGRL is tested: the development process, the tool, or a list of critical questions to serve as reminders during the goal modeling process.

8.3. Conclusion

The process of constructing a goal model involves discussions between requirements engineers 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.

---

[14]See page "Empirical study" on www.rationalgrl.com.

Our list of argument schemes and critical questions in Table 2 was constructed by performing a coding analysis in which we analyzed 153 pages of transcripts of discussions among designers of a traffic simulator information system. In this way, we ensure that the 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 Fig. 6 clearly specifies the formal traceability links elements of GRL goal model and the underlying arguments. (**Requirement 2**). In addition to this meta-model, 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 [13]. 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 accept-able given potentially contradictory opinions of stakeholders. Thus, we add a new formal evaluation technique for goal models that allows us to assess the *acceptability* of elements of a goal model (in addition to their *satisfiability* [2]).

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 development process, 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,[15] for modeling argument schemes and critical questions and for reasoning about goal models with respect to them (**Requirement 5**). We performed a small empirical evaluation with 16 users. The results indicate that while the cognitive overhead of RationalGRL (or goal modeling in general) is high, combining goal modeling with argumentation is perceived as useful. Furthermore, using a formal semantics to compute which arguments and ultimately goal models are acceptable given the stakeholders' opinions valid argument was perceived as very useful.

Appendix A. UCI design workshop prompt

A.1. *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

---

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

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.

Appendix B. Transcript excerpts

Table 5

Adding tasks, disabling unnecessary task "add traffic light" (transcript t_1)

Speaker	Text	Coding
0:15:11 (P1)	And then, we have a set of actions. Save map, open map, add and remove intersection, roads	**[20 task (AS2)]** Student has tasks "save map", "open map", "add intersection", "remove intersection", "add road", "add traffic light" [INTRO]
0:15:34 (P2)	Yeah, road. Intersection, add traffic lights	
0:15:42 (P1)	Well, all intersection should have traffic lights so it's	**[21 critical question CQ11 for 20]** Is the task "Add traffic light" useful/relevant?
0:15:44 (P2)	Yeah	**[22 answer to 21]** Not useful, because according to the specification all intersections have traffic lights. [DISABLE]
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 6

Clarifying the name of a task (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 7

Incorrect decomposition type for goal *simulate* (transcript t_3)

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?
0:30:12 (P2)	That's and OR	**[27 answer to 26]** No, it should be an OR decomposition. [REPLACE]
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 8

Discussion about the relevance of an actor (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	

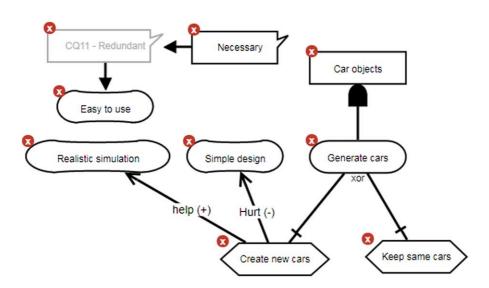## Appendix C. From RationalGRL to GRL – examples from the tool

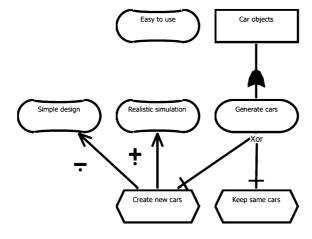Fig. 20. RationalGRL model of Fig. 13 modeled in the RationalGRL tool.

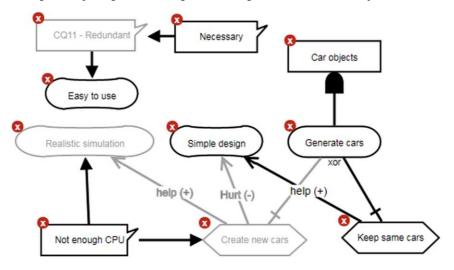Fig. 21. Exporting the model Fig. 20 to GRL gives the above model in jUCMNAv.

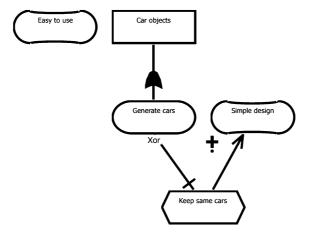Fig. 22. Adding an argument that attacks multiple GRL elements to Fig. 20.

Fig. 23. Exporting the model Fig. 22 to GRL gives the above model in jUCMNAv.

Appendix D. Modeling task – traffic controller

A group of three engineers (E1, E2, and E3) is tasked with designing a traffic flow simulation program. They receive a task specification, focusing on how the simulator will handle the generation of cars in the simulation. Below is an excerpt discussing the early requirements of the system. You are asked to model this discussion in as much detail as possible in the RationalGRL tool (http://www.rationalgrl.com), using all the tools RationalGRL puts at your disposal (GRL elements, critical questions, arguments).

As the RationalGRL prototype does not yet have a save functionality, please save a screenshot of your final model.

Person	Transcript
E1	Let's start by adding the actor "Traffic Simulator". Do you think we need to add more actors?
E2	I don't think so. Let's start with the softgoals and derive the main goals and tasks from that.
E3	The task description states that the simulator should be easy to use.
E2	I'm not sure that we should include that. It seems to me this is redundant since any UI should be easy to use.
E1	No I disagree, for some UI's it is more important than others, and the fact that they mention it explicitly in the task description means we should probably add it anyway.
E3	I agree, let's keep it in, it seems necessary.
E1	Ok so we have one softgoal 'easy to use'. Anything else?
E2	Here it say the simulation should be realistic, and the design should be simple. I guess those are two softgoals as well.
E3	Ok sounds good. So one goal of the simulation is to Generate cars. Shall we just add that as a goal?
E1	Yes let's do that.
E2	I think we then have two main tasks for that goal: The simulator should be able to create new cars in the simulation, and it should be able to wrap the cars when they go out of the screen.
E3	So Generate cars AND-decomposes into those two tasks?
E1	I don't think so actually. I think it is a XOR-decomposition. The simulation either creates new cars or wrap them when they are being reused.
E2	Hmm makes sense. But then I think we should rename Wrap cars to Keep same cars.
E3	Yes that is better, then Create new cars and Keep same cars are the two alternatives for the goal Generate cars.
E1	Hmm what do you think about the contribution links? Should we add any?
E2	Well I think if we create new cars, the simulation will be more realistic because it becomes less static. At the same time it will probably make the design less simple.
E3	Yes so create new cars contributes positively to the realistic design, and negatively to simple design?
E2	Hmm now I think about it, actually I think creating new cars doesn't have much effect on the realism of the simulation.I don't think the user will notice the difference.
E1	Why not? If you create new cars it changes the flow of the traffic, so I think the user will definitely notice this.
E2	Okay well, ok maybe. Yes I think you are right. Let's keep the contribution link in then.

D.1. Example RationalGRL models made by participants
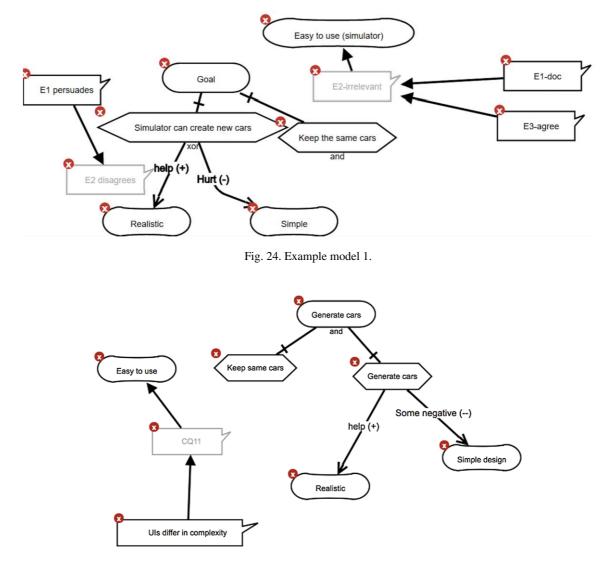

Fig. 24. Example model 1.
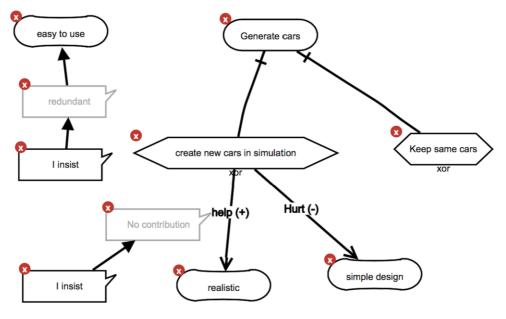

Fig. 25. Example model 2.

Fig. 26. Example model 3.

References

[1] C.E. Alchourrón, P. Gärdenfors and D. Makinson, On the logic of theory change: Partial meet contraction and revision functions, *The journal of symbolic logic* **50**(2) (1985), 510–530. doi:10.2307/2274239.

[2] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton and E.S.K. Yu, Evaluating goal models within the goal-oriented requirement language, *International Journal of Intelligent Systems* **25** (2010), 841–877. doi:10.1002/int.20433.

[3] K. Atkinson and T. Bench-Capon, Practical reasoning as presumptive argumentation using action based alternating transition systems, *Artificial Intelligence* **171**(10) (2007), 855–874. doi:10.1016/j.artint.2007.04.009.

[4] K. Atkinson, T. Bench-Capon and P. McBurney, PARMENIDES: Facilitating deliberation in democracies, *Artificial Intelligence and Law* **14**(4) (2006), 261–275. doi:10.1007/s10506-006-9001-5.

[5] E. Bagheri and F. Ensan, Consolidating multiple requirement specifications through argumentation, in: *Proceedings of the 2011 ACM Symposium on Applied Computing*, ACM, 2011, pp. 659–666. doi:10.1145/1982185.1982332.

[6] P. Bellini, R. Mattolini and P. Nesi, Temporal logics for real-time system specification, *ACM Comput. Surv.* **32**(1) (2000), 12–42. doi:10.1145/349194.349197.

[7] T.J. Bench-Capon and P.E. Dunne, Argumentation in artificial intelligence, *Artificial intelligence* **171**(10–15) (2007), 619–641. doi:10.1016/j.artint.2007.05.001.

[8] 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) (2003), 125–165. doi:10.1023/B:ARTI.0000046007.11806.9a.

[9] 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*, Springer, 2013, pp. 95–110.

[10] L. Chung, B.A. Nixon, E. Yu and J. Mylopoulos, *Non-functional Requirements in Software Engineering*, Vol. 5, Springer Science & Business Media, 2012.

[11] J. Conklin and M. Begeman, gIBIS: A hypertext tool for exploratory policy discussion, *ACM Transactions on Office Information Systems* **6**(4) (1988), 303–331. doi:10.1145/58566.59297.

[12] B. Curtis, H. Krasner and N. Iscoe, A field study of the software design process for large systems, *Communications of the ACM* **31**(11) (1988), 1268–1287. doi:10.1145/50087.50089.

[13] 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) (1995), 321–357. doi:10.1016/0004-3702(94)00041-X.

[14] 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)*, IEEE Press, 2017, pp. 51–60. doi:10.1109/RE.2017.27.

[15] M.A. Falappa, G. Kern-Isberner and G.R. Simari, Belief revision and argumentation theory, in: *Argumentation in Artificial Intelligence*, Springer, 2009, pp. 341–360. doi:10.1007/978-0-387-98197-0_17.

[16] 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) (2013), 21. doi:10.1145/2491509.2491515.

[17] A. Finkelstein and H. Fuks, Multiparty specification, in: *ACM SIGSOFT Software Engineering Notes*, Vol. 14, ACM, 1989, pp. 185–195.

[18] V. Gervasi and D. Zowghi, Reasoning about inconsistencies in natural language requirements, *ACM Transactions on Software Engineering and Methodology (TOSEM)* **14**(3) (2005), 277–330. doi:10.1145/1072997.1072999.

[19] S. Ghanavati, Legal-urn framework for legal compliance of business processes, PhD thesis, University of Ottawa, 2013.

[20] 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), CEUR Workshop Proceedings*, Vol. 1829, 2017, pp. 97–102.

[21] E. Gonçalves, M.A. de Oliveira, I. Monteiro, J. Castro and J. Araújo, Understanding what is important in iStar extension proposals: The viewpoint of researchers, *Requirements Engineering* (2018), 1–30.

[22] T.F. Gordon, Visualizing Carneades argument graphs, *Law, Probability & Risk* **6**(1–4) (2007), 109–117. doi:10.1093/lpr/mgm026.

[23] 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) (2008), 133–153. doi:10.1109/TSE.2007.70754.

[24] J. Hassine and D. Amyot, A questionnaire-based survey methodology for systematically validating goal-oriented models, *Requirements Engineering* **21**(2) (2016), 285–308. doi:10.1007/s00766-015-0221-7.

[25] J. Horkoff and E. Yu, Interactive goal model analysis for early requirements engineering, *Requirements Engineering* **21**(1) (2016), 29–61. doi:10.1007/s00766-014-0209-8.

[26] 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)*, IEEE, 2014, pp. 7–12. doi:10.1109/ESPRE.2014.6890521.

[27] I.J. Jureta, S. Faulkner and P.Y. Schobbens, Clear justification of modeling decisions for goal-oriented requirements engineering, *Requirements Engineering* **13**(2) (2008), 87–115. doi:10.1007/s00766-007-0056-y.

[28] E. Kavakli and P. Loucopoulos, Goal modeling in requirements engineering: Analysis and critique of current methods, in: *Information Modeling Methods and Methodologies*, J. Krogstie, T.A. Halpin and K. Siau, eds, Idea Group, 2005, pp. 102–124. ISBN 1-59140-376-6. doi:10.4018/978-1-59140-375-3.ch006.

[29] W. Kunz and H. Rittel, Issues as elements of information systems, Working Paper, 131, Institute of Urban and Regional Development, University of California, Berkeley, California, 1970.

[30] J. Lockerbie, N.A.M. Maiden, J. Engmann, D. Randall, S. Jones and D. Bush, Exploring the impact of software requirements on system-wide goals: A method using satisfaction arguments and i* goal modelling, *Requirements Engineering* **17**(3) (2012), 227–254. doi:10.1007/s00766-011-0138-8.

[31] 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)*, 2016, http://www.marcvanzee.nl/publications/2016/caise2016_eaprinciples.pdf.

[32] 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, Proceedings*, Montpellier, France, August 27–28, 2012, pp. 110–127.

[33] S. Modgil, Reasoning about preferences in argumentation frameworks, *Artificial Intelligence* **173**(9) (2009), 901–934. doi:10.1016/j.artint.2009.02.001.

[34] T.P. Moran and J.C. Carroll (eds), *Design Rationale*, L. Erlbaum Associates, 1996.

[35] 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)*, IEEE, 2015, pp. 156–165. doi:10.1109/RE.2015.7320418.

[36] G. Mussbacher and D. Amyot, Goal and scenario modeling, analysis, and transformation with jUCMNav, in: *ICSE Companion*, 2009, pp. 431–432.

[37] K. Neace, R. Roncace and P. Fomin, Goal model analysis of autonomy requirements for unmanned aircraft systems, *Requirements Engineering* **23**(4) (2018), 509–555. doi:10.1007/s00766-017-0278-6.

[38] B. Nuseibeh, Weaving together requirements and architectures, *Computer* **34**(3) (2001), 115–119. doi:10.1109/2.910904.

[39] 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) (2016), 394–426.

[40] C. Reed and G. Rowe, Araucaria: Software for argument analysis, diagramming and representation, *International Journal on Artificial Intelligence Tools* **13**(04) (2004), 961–979. doi:10.1142/S0218213004001922.

[41] J. Saldaña, The coding manual for qualitative researchers, Sage, 2015.

[42] S.B. Shum and N. Hammond, Argumentation-based design rationale: What use and what cost?, *International Journal of Human-Computer Studies* **40**(4) (1994), 603–652. doi:10.1006/ijhc.1994.1029.

[43] 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*, Springer, 2006, pp. 111–132. doi:10.1007/978-3-540-30998-7_5.

[44] S.R. Smith, Constructing and revising formal arguments in requirements engineering, *Requir. Eng.* **2**(4) (1997), 199–216. doi:10.1007/BF02745372.

[45] A. Tang, F. Bex, C. Schriek and J.M.E. van der Werf, Improving software design reasoning – a reminder card approach, *Journal of Systems and Software* **144** (2018), 22–40. doi:10.1016/j.jss.2018.05.019.

[46] UCI, Design Prompt: Traffic Signal Simulator, Accessed: 2016-12-27.

[47] A. Van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in: *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[48] 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)*, IEEE Press, 2015, pp. 220–225, http://www.marcvanzee.nl/publications/2015/renext2015_argumentation_for_requirements_engineering2.pdf.

[49] 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)*, 2016, pp. 553–560, http://www.marcvanzee.nl/publications/2016/er2016_rationalgrl.pdf.

[50] D. Walton, C. Reed and F. Macagno, *Argumentation Schemes*, Cambridge University Press, 2008.

[51] 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*, 1997, pp. 226–235, http://dl.acm.org/citation.cfm?id=827255.827807. ISBN 0-8186-7740-6. doi:10.1109/ISRE.1997.566873.

[52] 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** (2015), 102–116. doi:10.1016/j.jss.2015.04.065.