RationalGRL: A Framework for Argumentation and Goal Modeling

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

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

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

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

Abstract. Goal modeling languages capture the relations between an information system and its environment using high-level goals and their relationships with lower level goals and tasks. The process of constructing a goal model usually involves discussions between a requirements engineer and a group of stakeholders. While it is possible to capture part of this discussion process in the goal model, for instance by specifying alternative solutions for a goal, not all of the arguments can be found back in the resulting model. For instance, reasons for accepting or rejecting an element or a relation between two elements are not captured. In this paper, we investigate to what extent argumentation techniques from artificial intelligence can be applied to goal modeling. We apply the argument scheme for practical reasoning (PRAS), which is used in AI to reason about goals to the Goal-oriented Requirements Language (GRL). We develop a formal metamodel for the new language, link it to the GRL metamodel, and we implement our extension into jUCMNav, the Eclipse-based open source tool for GRL.

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

1 Introduction

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

One of the first activities in RE are the "early-phase" requirements engineering activities, which include those that consider how the intended system should meet organizational goals, why it is needed, what alternatives may exist, what implications of the alternatives are for different stakeholders, and how the interests and concerns of stakeholders might be addressed [**?**]. This is generally referred to as goal modeling. Given the number of currently established RE methods using goal models in the early stage of requirements analysis (e.g., [**?**,**?**,**?**,**?**,**?**], overviews can be found in [**?**,**?**]), there is a general consensus that goal models are useful in RE. Several goal modeling languages have been developed in the last two decades. The most popular ones include $i*$ [**?**], Keep All Objects Satisfied (KAOS) [**?**], the NFR framework [**?**], TROPOS [**?**], the Business Intelligence Model (BIM) [**?**], and the Goal-oriented Requirements Language (GRL) [**?**], which is part of an ITU-T standard, the User Requirements Notation (URN) [**?**].

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

dustrial production processes, or government and healthcare services- are complex and are not constructed in a short amount of time, but rather over the course of several workshops. In such situations, failing to record the discussions underlying a goal model in a structured manner may harm the success of the RE phase of system development for several reasons as followed.

TODO for Sepideh(by Marc): Please improve 2 and 4 by adding references

1. It is well-known that stakeholders' preferences are rarely absolute, relevant, stable, or consistent [**?**]. Therefore, it is possible that a stakeholder changes his or her opinion about a modeling decision in between two goal modeling sessions, which may require revisions of the goal model. If previous preferences and opinions are not stored explicitly, it is not possible to remind stakeholders of their previous opinions, thus risking unnecessary discussions and revisions. As the number of participants increases, revising the goal model based on changing preferences can take up a significant amount of time.
2. Other users who were not the original authors of the goal model may have to make sense of the goal model, for instance, to use it as an input in a later RE stage. If these user have no knowledge of the underlying rationale of the goal model, it may not only be more difficult to understand the model, but they may also end up having the same discussions as the previous group of stakeholders.
3. Alternative different ideas and opposing views that could potentially have led to different goal diagrams are lost. For instance, a group of stakeholders specifying a goal model for a user interface may decide to reduce softgoals "easy to use" and "fast" to one softgoal "easy to use". Thus, the resulting goal model will merely contain the softgoal "easy to use", but the discussion as well as the decision to reject "fast" are lost. This leads to a poor understanding of the problem and solution domain. In fact, empirical data suggest that this is an important reason of RE project failure [**?**].
4. It is not possible to reason about changing arguments and their effect on the goal model, and vice versa. A stakeholder may change his or her opinion, but it is not always directly clear what its effect is on the goal model. Similarly, a part of the goal model may change, but it is not possible to reason about the consistency of this new goal model with the underlying arguments. This becomes more problematic if the participants constructing the goal model change, since modeling decisions made by one group of stakeholders may conflict with the underlying arguments put forward by another group of stakeholders.

Our research goal is to apply practical reasoning and argumentation theory from Artificial Intelligence (AI) to the goal modeling process, with the expectation that doing so will help resolve issues 1-4 above. We identified several important requirements for our framework: (1) it must be able to formally model parts of the discussion process in the early-requirements phase of an information system, (2) it must be able to generate goal models based on the formalize discussions, (3) it should have formal traceability links between goal elements and arguments in the discussions, (4) it must have tool support, (5) a methodology must be identified that allows the framework to be used by practitioners, and (6) the framework must identify arguments and rationales that might not have been found, or might have been lost, otherwise.

The first requirement might seem to be a little imprecise, since it does not specify exactly which parts of the discussion process, the framework should capture. However, if we combine this requirement with the second requirement, it get clear that we should capture the parts of the discussions that are required to form goal models. The sixth requirement is the validation against other goal modeling methodologies, and has not yet been completed.

In this context, the main research question is: *What is a suitable framework to formally capture the discussions between stakeholders such that it can generate goal models, and how to implement this framework?* The first five requirements are the success criteria of our approach, that is, the satisfaction of the five requirements will result in a positive answer to the research question. **TODO for all(by Sepideh): The last sentence has to be changed in the end to reflect the result when the paper/evaluation is done. If we write it this way, it means we are unsure of the result.**

1.1 Contributions

The contributions of this paper are aligned with our five success criteria. In order to formalize discussions of the early requirements phase of an information system (requirement 1), we use a technique from AI called *argument schemes*. This allows us to formulate arguments and counter-arguments using so-called *critical questions*. By combining arguments and critical questions, one can create a structured representation of a discussion. We, first, propose a set of arguments and questions we expect to find in those discussions. We, then, validate this initial set with transcripts containing discussions about the architecture of an information sys-

tem. After this, we obtain our final set of arguments and critical questions.

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

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

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

1.2 Organization

The first two sections are introductory: Section 2 contains background and introduces the Goal-oriented Requirements Language (GRL) [] and argument schemes, while Section 3 provides a brief and high-level overview of our framework and methodology using an example.

The rest of the article is in line with the main contributions. We develop the list of argument schemes in two iterations, corresponding to two sections: In Section 4, we introduce an initial list of Argument Schemes for Goal Modeling (GMAS) and associated critical questions based on an existing argument scheme. In Section **??**, we evaluate the initial list of argument schemes by annotating transcripts from discussions about an information system, which results in our final list of argument schemes and critical questions. In Section **??**, we develop a formal model for argument schemes and critical questions and we describe RationalGRL tool in Section 6. In Section 7, we propose our methodology while we discuss the related work in Section 8.

2 Background: Goal-oriented Requirements Language and Argument Schemes

In this section, we first introduce and motivate the Goal-oriented Requirements Language (GRL), which is the goal modeling language used to connect to the argumentation framework. Next, we introduce and motivate *the Argument Scheme for Practical Reasoning* (PRAS), which is a particular argument scheme that is used to form arguments and counter-arguments about situations involving goals.

2.1 Goal-oriented Requirements Language (GRL)

The User Requirements Notation (URN) [**?**], an ITU-T Standard, is one of the first modeling languages in the area of RE which aims at the standardization of visual notations to analyze functional (behavior) and non-functional requirements (NFRs).

URN allows software and requirements engineers to identify requirements for a proposed or an evolving system and to review such requirements for correctness and completeness. URN combines two complementary notations: the *Goal-oriented Requirement Language* (GRL) [**?**] for goals and NFRs, and *Use Case Maps* (UCMs) [**?**] for scenarios, business processes and functional requirements. In this article, we focus on GRL models and their links to argumentation.

GRL is a visual modeling language for specifying intentions, business goals, and non-functional requirements (NFRs) of multiple stakeholders. It can be used to specify alternatives that have to be considered, decisions that have been made, and rationales for making decisions. A GRL model is a connected graph of intentional elements that optionally are part of actors. Figure 1 illustrates a GRL diagram. An actor () represents a stakeholder of a system, or the system itself (e.g. In Figure 1, Meeting Participant). Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available, and softgoals to be satisfied. Softgoals () differentiate themselves from goals () in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable, often in a binary way. Softgoals (e.g. User

Friendly) are often more related to NFRs, whereas goals (such as Agreeable Meeting Dates) are more related to functional requirements. Tasks ( ) represent solutions to (or operationalizations of) goals and softgoals. In Figure 1, some of the tasks are Attend Meetings and Arrange Meetings. In order to be achieved or completed, softgoals, goals, and tasks may require resources ( ) to be available (e.g. Agreement)

Different links connect the elements in a GRL model. AND, IOR, and XOR decomposition links ( ) allow an element to be decomposed into sub-elements. In Figure 1, the task Participate in Meetings is AND-decomposed to the tasks Attend Meetings and Arrange Meetings as well as the softgoal Convenient Meeting Date. Contribution links ( ) indicate desired impacts of one element on another element. A contribution link has a qualitative contribution type or a quantitative contribution. Softgoal User Friendly has some positive qualitative contribution to the softgoal Low Efforts. Dependency links ( ) model relationships between actors. For example, actor Meeting Scheduler depends on the actor Meeting Participant to Agree to Date via the resource Proposal Dates to fulfill its task Schedule a Meeting.

GRL is based on $i*$ [**?**] and the NFR Framework [**?**], but it is not as restrictive as $i*$. Intentional elements and links can be more freely combined, the notion of agents is replaced with the more general notion of actors, i.e., stakeholders, and a task does not necessarily have to be an activity performed by an actor, but may also describe properties of a solution.

TODO for Sepideh(by Floris): can you haver a look at the motivation for using GRL (below)? It used to be a separate section but I think it fits best here, have a look at what we need. I agree that we can mention jUCMNav but not use it as motivation since we will use online tool.

GRL is an international standard with well defined syntax and semantics. This allows us to combine the theory of formal argumentation with GRL precisely, hereby satisfying requirement 3 as described in the introduction. Moreover, GRL has an open source tool called jUCMNav, which simplifies an implementation. Since implementation is a key concern for us, the extensive tool support is a strong argument in favor of GRL.

According to Amyot et al. [**?**], GRL has several benefits over other existing goal modeling languages, such as integration with a scenario notation (UCM), support for both qualitative and quantitative attributes (for contributions levels and satisfaction levels), there is a clear separation of GRL model elements from their graphical representation, and it provides support for providing a scalable and consistent representation of multiple views/diagrams of the same goal model (see [**?**, Ch.2] for an more details). Finally, URN and jUCMNav provide traceability links between concepts and instances of goal modeling and behavioral design models. This traceability enables direct impact analysis of goal changes on a design and vice versa. This traceability feature seems to be a good fit with our current research: We aim to add traceability links between intentional elements and their underlying arguments. **TODO for Sepideh(by Sepideh): I need to improve: GRL has has the capability to be extended via profile, metadata and rules, and due to this capability, it has been extended to many domain. This feature also helps us to apply our argumentation to other domain such as compliance or enterprise architecture.**

Online Meeting Scheduler Example

In this paper, we use the classic example of an online meeting scheduler that has been used in the literature [], [], [] several times. The meeting scheduler has been modeled with Tropos [] and $i*$. We remodel this example with GRL in jUCMNav for the purpose of our framework.

The meeting scheduler system has three actors, **Meeting Initiator**, **Meeting Scheduler** and **Meeting Participant**. The **Meeting Scheduler** has a goal that for every meeting request, it has to try to find the best time and location in a way that most of the intended participants (i.e. **Meeting Participant**) can attend the meeting. The **Meeting Initiator** organizes the meetings by asking the participants to give their availability for a range of dates and time as well as the dates that they are not available. Based on these information, the **Meeting Scheduler** needs to pick a date/time and a location from the available dates/times given by the participants in a way that the most number of participants can attend. Participants should then accept or reject the event.

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

TODO for Marc(by Sepideh): I think you can add some more or remove/modify some of these based on your needs in the future sections

– Why are only the two softgoals **Quick** and **Low effort** selected for the task **Organize Meeting**? Why is, for example, a goal **Selecting the most convenient time** not included in the analysis for the actor **Meeting Initiator**?

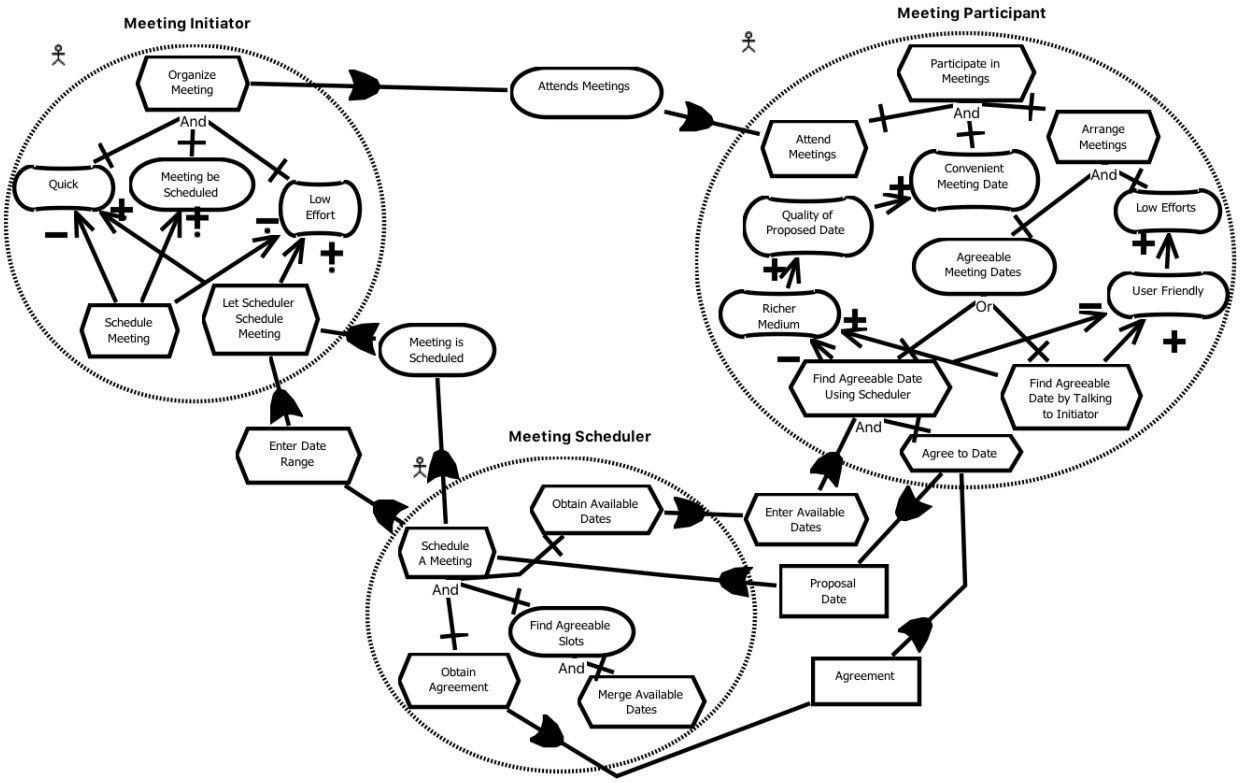

Fig. 1: GRL Model for Online Meeting Scheduler

- What does **Quality of Proposed Date** mean?
- How can one decide if the quality is low or high? Who is in charge? **TODO for Sepideh(by Floris): do you mean the Quality of Proposed Date? Or quality of the GRL diagram?**
- Does the meeting initiator use an email to inform the meeting participants to enter their availabilities?
- Does the meeting scheduler uses an online calendar to obtain the available dates or an online website?
- How much time do the participants have to enter their availabilities? Is that important? Do they get a reminder from either the initiator or the scheduler? Is the reminder sent via an email?
- What does **Richer Medium** mean?

TODO for all(by Marc): Improved this part, what do you think? These are the type of the questions that we cannot answer just by looking at the GRL models. The model in figure 1 does not contain information about discussions that let up to the resulting elements of the model, such as by various clarification steps for the naming, or alternatives that have been considered for the relationships. In order to address this, we use Argument Schemes for Goal Modeling (GMAS). GMAS can help us decide what intentional elements should remain in GRL and what new ones to be added or deleted, provide rationale behind the design decisions and the relationships between the links.

2.2 Argument Scheme for Practical Reasoning (PRAS)

Reasoning about which goals to pursue and actions to take is often referred to as *practical reasoning*, and has been studied extensively in philosophy (e.g. [**?**,**?**]) and Artificial Intelligence [**?**,**?**]. One approach is to capture practical reasoning in terms of arguments schemes and critical questions [**?**]. The idea is that an instantiation of such a scheme gives a presumptive argument in favor of, for example, taking an action. This argument can then be tested by posing critical questions about, for instance, whether the action is possible given the situation, and a negative answer to such a question leads to a counter-argument to the original presumptive argument for the action.

A formal approach to persuasive and deliberative reasoning about goals and actions has been presented by Atkinson et al. [**?**], who define the Practical Reasoning Argument Scheme (PRAS). PRAS follows the following

basic argument structure.

$$\begin{aligned} &\text{I have goal } G \\ &\text{Doing action } A \text{ will realize goal } G \\ &\text{Which will promote value } V \\ &\textit{Therefore} \text{ I should do action } A. \end{aligned} \tag{1}$$

So, for example, we can say that

> **Find agreeable meeting date** is a goal,
> **Find agreeable date by talking to initiator** will realize the goal **(Find) agreeable meeting date**,
> Which will promote the value **User friendliness**
> *Therefore*
> We should perform action **Find agreeable date by talking to initiator**

Practical reasoning is defeasible, in that conclusions which are at one point acceptable can later be rejected because of new information. Atkinson *et al.* [**?**] define a set of critical questions that point to typical ways in which a practical argument can be criticized by, for example, questioning the validity of the elements in the scheme or the connections between the elements. Some examples of critical questions are as follows.

1. Will the action bring about the desired goal?
2. Are there alternative ways of realizing the same goal?
3. Are there alternative ways of promoting the same value?
4. Does doing the action have a side effect which demotes some other value?
5. Does doing the action promote some other value?
6. Is the action possible?
7. Can the desired goal be realized?
8. Is the value indeed a legitimate value?

These critical questions can point to new arguments that might counter the original argument. Take, for example, critical question 8. If new evidence comes up indicating that meeting schedulers do not care about user friendliness we have a counterargument to the above argument, 'evidence shows that user friendliness is not a legitimate value'. Another way to counter an argument for an action is to suggest an alternative action that realizes the same goal (question 2) or an alternative goal that promotes the same value (question 3). For example, we can argue that **Find agreeable date using scheduler** also realizes the goal **Find agreeable meeting date**, which gives us a counterargument to the original argument – to find a date by talking to the initiator – that also follows PRAS.

In argumentation, counterarguments are said to *attack* the original arguments (and sometimes vice versa). In the work of Atkinson et al. [**?**], arguments and their attacks are captured as an *argumentation framework* of arguments and attack relations as introduced by Dung [**?**][1]. Figure **??** shows an argumentation framework with three arguments from the above example: the argument for **Find agreeable date by talking to initiator** (A1), the argument for **Find agreeable date using scheduler** (A3), and the argument that **User friendliness** is not a legitimate value (A2). The two alternative PRAS instantiations are A1 and A3. These arguments mutually attack each other, as **Find agreeable date by talking to initiator** and **Find agreeable date using scheduler** are considered to be mutually exclusive. Argument A2 attacks A1, as it questions the legitimacy of the value **User friendliness**.

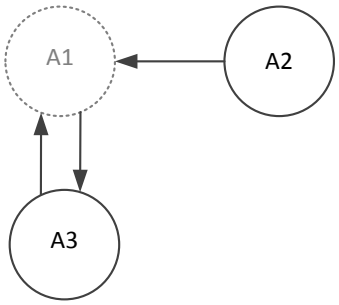

Fig. 2: Argumentation framework

Given an argumentation framework, the acceptability of arguments can be determined according to the appropriate argumentation semantics. The intuition is that an argument is acceptable if it is *undefeated*, that is, any argument that attacks it, is itself defeated. In the argumentation framework in Figure 2, argument A2 is undefeated because it has no attackers. This makes A1 defeated, because one of its attackers, A2, is undefeated. A3 is then also undefeated, since its only attacker, A1, is defeated by A2. Thus, the set of undefeated (justified) arguments given the argumentation framework in Figure 2 is $\{$A2, A3$\}$.

In some cases, it is more difficult to determine whether or not an argument is defeated. Take, for example, the argumentation framework with just A1 and A3: they attack each other, they are alternatives and without any explicit preference, it is impossible to choose between the two. It is, however, possible to include explicit preferences between arguments when determining argument acceptability [**?**]. Take, for example, A1 and A3.

[1] Full definitions of Dung's [**?**] frameworks and semantics will be given in section 4. In this section, we will briefly discuss the intuitions behind these semantics.

If we say that we prefer the action **Find agreeable date by talking to initiator** (A1) over the action **Find agreeable date using scheduler** (A3), we remove the attack from A1 to A3 (Figure 3, left). This makes A1 the only undefeated argument, whereas A3 is now defeated. It is also possible to give explicit arguments for preferences [**?**]. These arguments are then essentially attacks on attacks. For example, say we prefer A1 over A3 because 'the **Meeting participant** does not like working with a scheduler' (A4). This can be rendered as a separate argument that attacks the attack from A3 to A1 (Figure 3, right), removing this attack and making {A1, A4} the undefeated justified set of arguments.

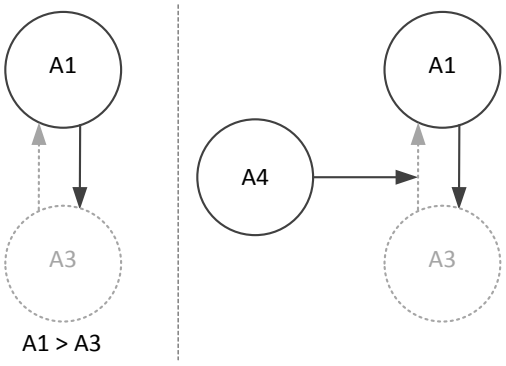

Fig. 3: Preferences between arguments

Practical Argumentation and Goal Modeling

Practical reasoning in the PRAS framework as described above provides a formally sound framework for defeasible reasoning about goals and actions that adheres to the acceptability semantics of Dung [**?**] and its various extensions [**?**,**?**]. The usefulness of PRAS for the analysis of practical reasoning situations has been shown in different areas such as e-democracy [**?**], law [**?**], planning [**?**] and choosing between safety critical actions [**?**]. The question in this paper is how PRAS can be adopted for use in goal modeling, so that we can capture the discussions between stakeholders that build a goal model as formal argumentation, thus adding a new evaluation technique for goal models that allows us to assess the *acceptability* of elements of a goal model (as opposed to the *satisfiability* **TODO for Floris(by Marc): Which article do you want to cite here?**[]).

TODO for Floris(by Marc): From here on you seem to be discussing related work. This kind of distracts us from the main point of this article, and the section becomes quite long. Perhaps it is better to make the main arguments about why our approach is useful here, and move the discussion of our previous work and Jureta to related work?

PRAS (actions, goals, values) and GRL (tasks, goals, softgoals) have some obvious similarities, and in previous work [**?**,**?**,**?**] we presented arguments based on PRAS of the form "G is a goal, Action A realizes G *Therefore* perform action A" (see Figure 4, left, where the arrow denotes an inference step). Such arguments, which can be constructed using the online OVA tool[2], can be combined with further arguments providing, for example, expert opinions about the practical reasoning elements (Figure 4, left).

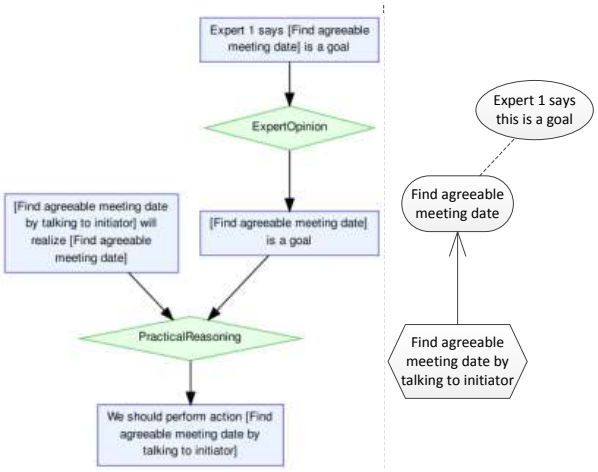

Fig. 4: Structured argument based on PRAS (left) and the corresponding GRL diagram (right)

Arguments built in OVA can automatically be translated to GRL diagrams using further online tooling[3] [**?**]. This translation takes the elements of the arguments and directly translates this to GRL elements. Tasks/actions, goals and softgoals/values can be directly translated from the arguments to the GRL diagrams, as can realization statements. Elements of arguments that have nothing to do with goal modelling (e.g. the expert opinion premise in Figure 4) can be included in the GRL diagram as *beliefs*. Finally, attacks between arguments are captured as negative contribution links. Take, for example, **TODO for .(by example): A** very similar approach to argumentation and goal models had already been taken by Jureta et al. [**?**] who proposed a formal argumentation model that can be used to justify goal-modelling decisions. This argumentation model, while not explicitly based on a practical reasoning scheme, is essentially

[2] http://ova.arg-tech.org/
[3] https://github.com/RationalArchitecture/RationalGRL

the same as our previous argumentation model based on PRAS: a formal model of structured arguments with goals as premises and tasks as conclusions, and a translation function of these arguments to goal diagrams.

The core problem of the use of argumentation in the above-mentioned work [**?**,**?**,**?**], is that it focuses mainly on structured arguments, which are then directly translated to goal models as in Figure 4. The question is what the added value of this exercise is. Firstly, the arguments and goal models contain many of the same elements. The idea is that arguments show the rationales behind goal models and the alternative or possibly conflicting views that were lost in the goal-modeling process. But GRL already contains a *belief* element that allow one to provide beliefs underlying a goal model, an *XOR decomposition* link that allows for modeling of alternative ways to realize goals, and *negative contribution* links for capturing conflicting goals.

The point is that argumentation models are not needed to precisely render goals, tasks, beliefs and the relations between them, as GRL is already perfectly suited for this. Rather, argumentation should guide the goal modelling *process*. Instead of static, formal argument trees, we need a dynamic argumentation process in which goals and tasks are critically analysed and which captures the dialectical nature of justification. Jureta et al. [**?**] take a step towards such a process by defining a high-level decision process that organizes argumentation and clarification techniques in relation to goal modelling. However, the argumentation techniques in this process are focused on building precisely the type of arguments that we argue are redundant next to GRL diagrams, namely structured arguments like in Figure 4 which can be translated more or less directly to goal models. The decision process itself is never formally defined, and how to go from a discussion on goals and tasks to a goal model remains largely implicit.

What is needed is a formal framework that captures the discussions about goal models, and allows for (semi-)automated construction of goal models given this discussion. In section XX, we argue that a combination of Atkinson et al.'s [**?**] critical questions and the dialectical semantics as formally captured by Dung's work [**?**] are ideally suited to provide such a formal, dynamic framework.

3 Preview of the Framework

TODO for Marc(by Marc): Give an example showing how our framework works without the technical details.

4 Argument Schemes for Goal Modeling (GMAS)

In this section we develop a set of argument schemes and critical questions for goal modeling. The final list of argument schemes and critical questions is shown in table 1. The first four argument schemes (AS0-AS4) are argument for an element of a goal model, the next seven (AS5-AS11) are about relationships, the next two (AS12-AS13) are about intentional elements in general, and the last is (Att) is a generic counterargument for any type of argument that has been put forward. For each critical question, the right column shows the effect of answering the critical questions affirmatively, which can be DISABLE (disable the element of the original argument), INTRO (introduce a new element and argument), REPLACE (replace the element of the original argument), ATTACK (attack an argument directly). For instance, if the critical question CQ0 "Is the actor relevant?" is answered with "No", then the actor of the original argument AS0 will be disabled in the goal model.

We obtained the list in table 1 by analyzing transcripts of discussions between students about the development of an information system. In the first subsection we provide details of this analysis process with concrete examples, after which we analyze our results in the next subsection. In the last subsection we provide several examples of the effect of answering a critical question. In the next section, we develop a formal language for our argument schemes, critical questions, and the effect of answering them.

4.1 Details experiment

The transcripts we used are created as part of two master theses on improving design reasoning [**?**,**?**].

Subjects The subjects for the case study are three teams of Master students from the University of Utrecht, following a Software Architecture course. Two teams consist of three students, and one team consists of two students.

Experimental Setup The assignment used for the experiments is to design a traffic simulator. Designers are provided a problem description, requirements, and a description of the desired outcomes. The original version of the problem descrption [**?**] is well known in the field of design reasoning since it has been used in a workshop[4], and transcripts of this workshop have been analyzed in detail [**?**]. Although the concepts of traffic

[4] `http://www.ics.uci.edu/design-workshop/`

Argument scheme		Critical Questions		Effect
AS0	Actor a is relevant	CQ0	Is the actor relevant?	DISABLE
AS1	Actor a has resource R	CQ1	Is the resource available?	DISABLE
AS2	Actor a can perform task T	CQ2	Is the task possible?	DISABLE
AS3	Actor a has goal G	CQ3	Can the desired goal be realized?	DISABLE
AS4	Actor a has softgoal S	CQ4	Is the softgoal a legitimate softgoal?	DISABLE
AS5	Goal G decomposes into tasks $T_1, \ldots, T_n$	CQ5a	Does the goal decompose into the tasks?	DISABLE
		CQ5b	Does the goal decompose into any other tasks?	REPLACE
AS6	Task T contributes to softgoal S	CQ6a	Does the task contribute to the softgoal?	DISABLE
		CQ6b	Are there alternative ways of contributing to the same softgoal?	INTRO
		CQ6c	Does the task have a side effect which contribute negatively to some other softgoal?	INTRO
		CQ6d	Does the task contribute to some other softgoal?	INTRO
AS7	Goal G contributes to softgoal S	CQ7a	Does the goal contribute to the softgoal?	DISABLE
		CQ7b	Does the goal contribute to some other softgoal?	INTRO
AS8	Resource R contributes to task T	CQ8	Is the resource required in order to perform the task?	DISABLE
AS9	Actor a depends on actor b	CQ9	Does the actor depend on any actors?	INTRO
AS10	Task T_1 decomposes into tasks $T_2, \ldots, T_n$	CQ10a	Does the task decompose into other tasks?	REPLACE
		CQ10b	Is the decomposition type correct? (AND/OR/XOR)	REPLACE
AS11	Task T contributes negatively to softgoal S	CQ11	Does the task contribute negatively to the softgoal?	DISABLE
AS12	Element IE is relevant	CQ12	Is the element relevant/useful?	DISABLE
AS13	Element IE has name n	CQ13	Is the name clear/unambiguous?	REPLACE
-	-	Att	Generic counterargument	ATTACK

Table 1: List of argument schemes (AS0-AS13, left column), critical questions (CQ0-CQ12, middle column), and the effect of answering them (right column).

lights, lanes, and intersections are common and appear to be simple, building a traffic simulator to represent these relationships and events in real time turns out to be challenging. Participants were asked to use a think-aloud method during the design session. The assignment was slightly adjusted to include several viewpoints as end products in order to conform to the course material [**?**]. The final problem descriptions can be found in Appendix A of Schriek's master thesis [**?**]. All groups were instructed to apply the *functional architecture method*, focusing on developing the *context*, the *functional*, and the *informational* viewpoints of the traffic simulator software. The students had two hours for the tasks, and the transcripts document the entire discussion. The details of the transcripts are shown in table 2.

	transcript t_1	transcript t_2	transcript t_3
participants	2	3	3
duration	1h34m52s	1h13m39s	1h17m20s

Table 2: Number of participants and duration of the transcripts.

Annotation method We started with an initial list of 10 argument schemes and 18 critical questions that we derived from PRAS. We annotated transcripts with the arguments and critical questions from this list. If we found arguments or critical questions that did not appear in the original list, we added them and counted them as well. Argument schemes that did not appear were removed from the list, but critical questions were not removed (see discussion in section 4.2). Most of the occurrences were not literally found back, but had to be inferred from the context. This can be seen in the various examples we will discuss.

TODO for Floris(by Marc): I'd like to say something in the next paragraph about the difficulty of extracting arguments from text. Do you know any references? It is generally known in the argumentation literature that it can be very difficult to annotate arguments correctly. Arguments are often imprecise, lack conclusion, and may be supported by non verbal communication that is not captured in the transcripts. Still, since research on argument extraction in the requirement engineering domains is in its infancy, we believe that our evaluation is useful by itself. Furthermore, our annotation is openly available[5], we provide parts of our annota-

[5] **TODO for Marc(by Marc): provide url**

tion in Appendix A, and most of the examples from this article come from the transcripts. In this way, we aim to make our annotation process as transparent as possible.

Results We found a total of 159 instantiations of the argument schemes AS0-AS11 in the transcripts. The most used argument scheme was AS2: "Actor A has task T", but each argument scheme has been found back in the transcripts at least twice (table 3). A large portion (about 60%) of the argument schemes we found involved discussions around tasks of the information system (AS2, AS10).

We annotated 41 applications of critical questions. Many critical questions (about 55%) involved clarifying the name of an element, or discussing about the relevance of it (CQ12, CQ13).

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 to softgoal	6	2	0	**8**
AS7	Goal contributes 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**
AS11	Task contributes negatively to softgoal	2	1	0	**3**
CQ2	Task is possible?	2	2	1	**5**
CQ5a	Does the goal decompose into the tasks?	0	1	0	**1**
CQ5b	Goes 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**
CQ12	Is the element relevant/useful?	2	3	2	**7**
CQ13	Is the name clear/unambiguous?	3	10	3	**16**
-	Generic counterargument	0	2	2	**4**
TOTAL		69	80	69	**222**

Table 3: Number of occurrences of AS0-AS9, CQ0-CQ12 in the transcripts. Critical questions not appearing in this table were not found back in the transcripts.

For each transcript, we manually create a GRL model from the argument schemes and critical questions we found in them, in order to verify whether the arguments put forward by the participants were sufficiently informative. An example of such a model is shown in figure 5. We added green and red dots to various elements and relationships in the figure. A green dot indicates there is an underlying argument for the element that is accepted, while a red dot indicates a rejected underlying argument. Note that if the underlying argument is rejected, the corresponding GRL element has been disabled. Some elements do not have a corresponding green or red dot. In that case, we have inferred the elements from the discussion, but we could not explicitly find back arguments for it.

We found that answering a critical questions can have four different effects on the original argument:

- *INTRO*: Introduce a new goal element or relationship with a corresponding argument. This operation does not attack the original argument of the critical question, but rather creates a new argument. For instance, suppose argument scheme AS5 is instantiated as follows:: "Goal *Agreeable meeting dates* OR-decomposes into tasks *Find agreeable date using scheduler* and *Find agreeable date by talking to initiator*" (figure 1). Suppose now the critical question CQ5b: "Does the goal *Agreeable meeting dates* decompose into other tasks?" is answered with "yes, namely *Find agreeable date by contacting secretary*". This results in a new instantiation of AS5, namely: "Goal *Agreeable meeting dates* OR-decomposes into tasks *Find agreeable date using scheduler*, *Find agreeable date by talking to initiator*, and *Find agreeable date by contacting secretary*. As a result, the goal model will contain the corresponding task *Find agreeable date by contacting secretary*, as well as an OR-decomposition relation from the goal *Agreeable meeting dates* to that task.
- *DISABLE:* Disable the element or relationship of the argument scheme to which the critical questions pertains. This operation does not create a new argument, but only disables (i.e., attacks) the original one. For instance, suppose argument scheme AS0 is instantiated with: "Actor *meeting initiator* is relevant" (figure 1). This argument can be attack with critical question CQ0: "Actor *meeting initiator* is not relevant". As a result, the argument for the actor is attack, and actor *meeting initiator* is disabled in the goal model.
- *REPLACE:* Replace the element of the argument scheme with a new element. This operation both introduces a new argument and attacks the original one. For instance, suppose argument scheme AS2 is instantiated wit: "Actor *meeting initiator* can perform task *schedule meeting*. This argument can be attacked with critical question CQ13: "The task *schedule meeting* is unclear, it should be *schedule board meeting*". This results in replacing the original argument with the new argument "Actor *meeting initiator* can perform task *schedule board meeting*.

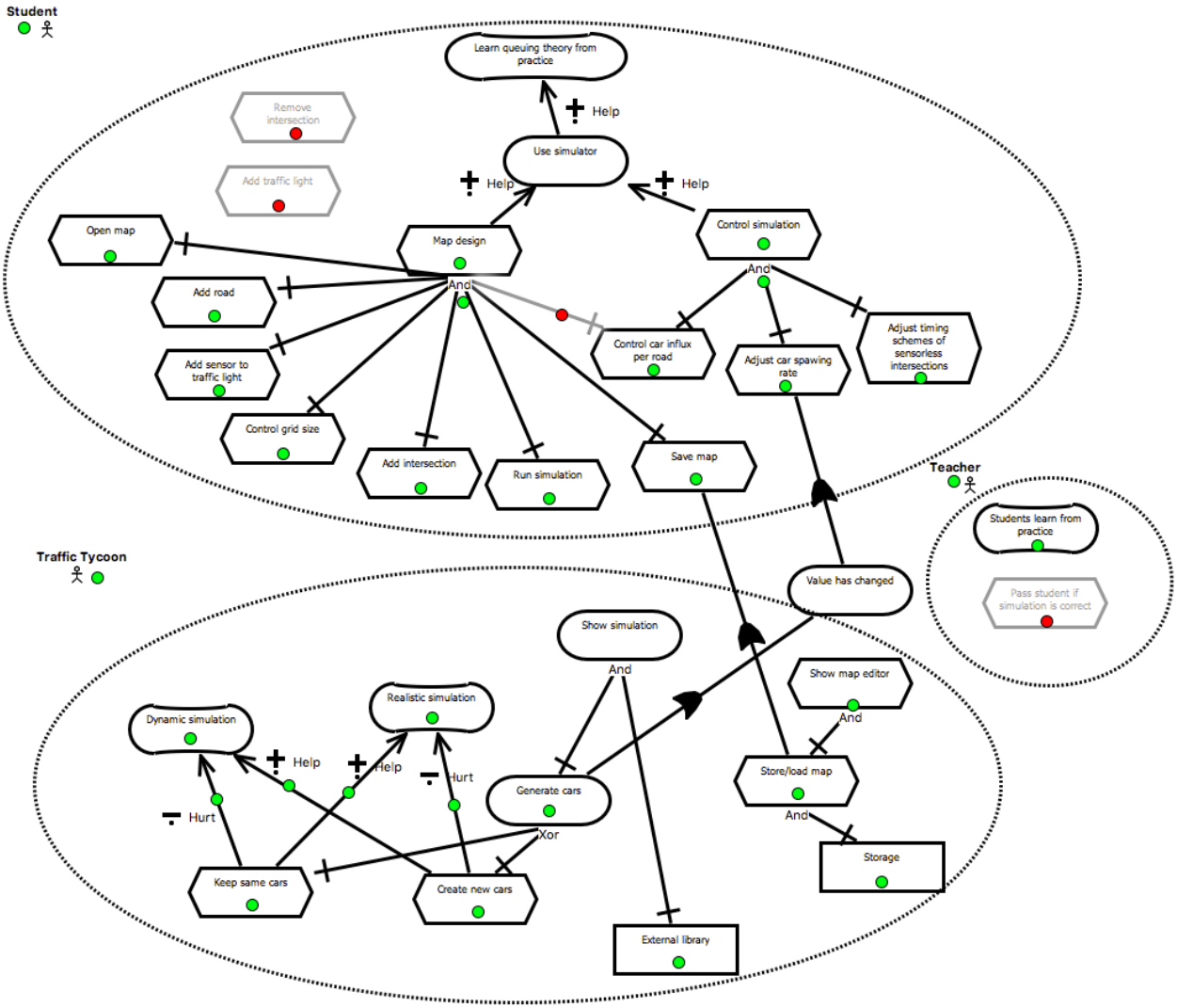

Fig. 5: The GRL model manually constructed from transcript t_1. Green dots indicate accepted underlying arguments, red dots indicate rejected underlying arguments. Elements and relationships with no dot have been inferred by us.

In the goal model, the description of the task should change accordingly.

– *ATTACK:* Attack any argument with an argument that cannot be classified as a critical question. For instance, suppose argument scheme AS4 is instantiated as follows: "Actor *meeting initiator* has goal *quick*. For instance, suppose argument scheme AS0 is instantiated with "Actor *meeting scheduler* is relevant". Suppose this argument is attacked with critical question CQ0: "Actor *meeting initiator* is not relevant", because the participants decide to fully automate the meeting scheduling process. However, this critical question may in turn be attacked again, if new evidence comes up. For instance, it may turn out that it is sometimes more desirable to schedule meeting by hand. In this case, the generic counter argument "Actor *meeting initiator* is relevant, because it is sometimes easier to schedule meetings manually" attack the argument "Actor *meeting scheduler* is not relevant". As a result, the original argument "Actor *meeting scheduler* is relevant" is accepted again, and as a result is shown in the goal model.

In section 4.3 we provide examples we found in the transcripts for all of these four effects.

4.2 Analysis

Analysis of the argument schemes Our initial list of argument schemes consists of AS1-AS4, AS5-AS9 (figure 1). Therefore, the difference between the initial list

of argument schemes and those found back in the transcripts is quite small. We found it surprising that we were able to find back all the schemes in the transcript at least twice, even more since the topic of discussion wasn't goal models, but more generally the architecture of an information system. This gives us an indication that these argument schemes are able to capture arguments used in those type of discussions to some extent.

More generally, we observed that our initial list is rather limited, which is a consequence of the fact that it is derived from PRAS. Since PRAS only considers very specific types of relationships, we are not able to capture many other relationships existing in GRL. GRL has four types of intentional elements (softgoal, goal, task, resource) and four types of relationships (positive contribution, negative contribution[6], dependency, decomposition), allowing theoretically $4^3 = 64$ different types of argument schemes, of which we currently only consider 11. Our analysis however shows that many of these schemes are not often used, and thus gives us some confidence in the resulting list.

Analysis of the critical questions The difference between the initial list of critical questions and those we found back in the transcripts is much larger than for the critical questions. On the one hand, we found back few of the critical questions we initially proposed. However, this does not mean that they weren't implicitly used in the minds of the participants. If a participants makes an argument for a contribution from a task to a softgoal, it may very well be that she was asking herself the question "Does the task contribute to some other softgoal?". However, many of these critical questions are not mentioned explicitly. If we assume this explanation is at least partially correct, then this would mean that critical questions may still play a role when formalizing the discussions leading up to a goal model, and it would be limiting to leave them out of our framework. In the context of tool support, we believe that having these critical questions available may stimulate discussions.

4.3 Examples

We now discuss various instantiations of argument schemes and the result of answering critical questions in more detail. For each example we provide transcript excerpts, a visualization of the arguments, and the corresponding goal model elements. We provide a legend for our visualization notation in figure 6.

[6] In fact, a contribution can be any integer in the domain [-100,100], but for the sake of simplicity we only consider two kinds of contributions here.

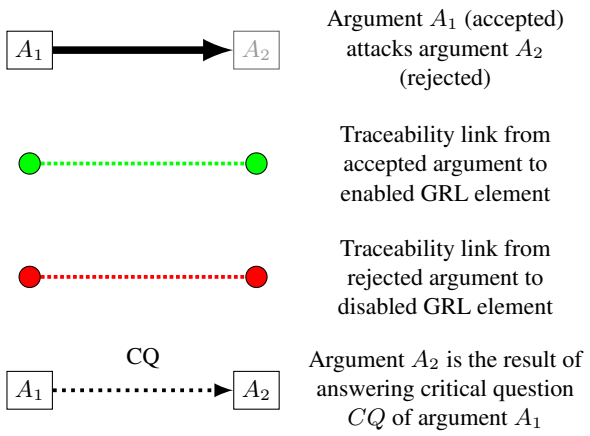

Fig. 6: Legend of the various elements and relationships we use for the examples in this article.

Example 1: Disable task *Traffic light*

The transcript excerpt of this example is shown in table 4 in the appendix and comes from transcript t_1. In this example, participants are summing up functionality of the traffic simulator, which are tasks that the student can perform in the simulator. All these tasks can be formalized are instantiations of argument scheme AS2: "Actor *Student* has task T", where $T \in$ {Save map, Open map, Add intersection, Remove intersection, Add road, Add traffic light}".

Once all these tasks are summed up, participant P1 notes that the problem description states that all intersections in the traffic simulator have traffic lights, so the task *Add traffic light* is not useful. We formalized this using the critical question CQ12: "Is task *Add traffic light* useful/relevant?".

We visualized the some of the argument schemes, critical questions, and traceability links with the GRL model in figure 7. On the left side of the image, we see three of the instantiated argument schemes AS2. The bottom one, "Actor *Student* has task *Add traffic light*", is attack by another argument generated from applying critical question CQ12: "*Add traffic light* is useless (*All intersections have traffic lights*). As a result, the corresponding GRL task is disabled. The other two tasks are enabled and have green traceability links.

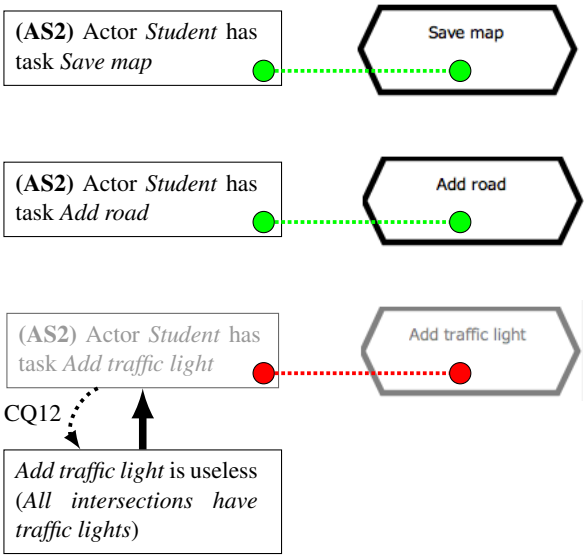

Fig. 7: Argument schemes and critical questions (left), GRL model (right), and traceability link (dotted lines) for the traffic light example.

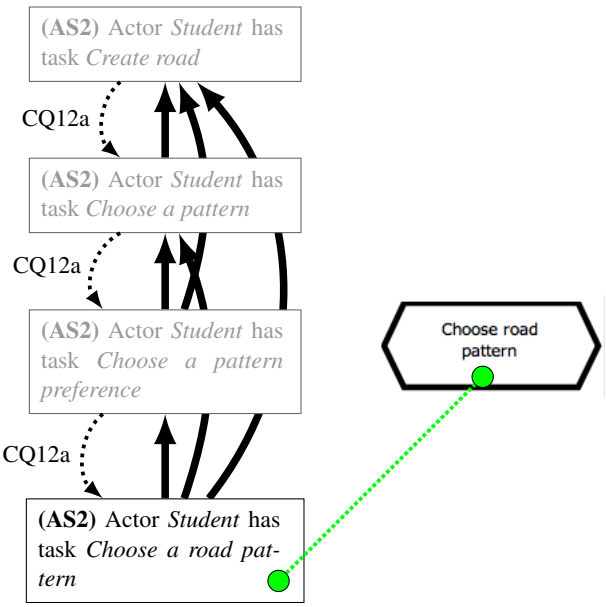

Fig. 8: Argument schemes and critical questions (left), GRL model (right), and traceability link (dotted line) of the road pattern example.

Example 2: Clarify task *Road pattern*

The transcript excerpt of the second example is shown in table 5 in the appendix and comes from transcript t_3. It consists of a number of clarification steps, resulting in the task *Choose a road pattern*.

The formalized argument schemes and critical questions are shown in figure 8. The discussion starts with the first instantiation of argument scheme AS2: "Actor *Student* has task *Create road*". This argument is then challenged with critical question CQ12: "Is the task *Create road* clear?". Answering this question results in a new instantiation of argument scheme AS2: "Actor *Student* has task *Choose a pattern*". This process is repeated two more times, resulting in the final argument "Actor *Student* has task *Choose a road pattern*". This final argument is unattacked and has a corresponding intentional element (right image).

What is clearly shown in this example is that a clarifying argument attacks all arguments previously used to describe the element. For instance, the final argument on the bottom of figure 8 attacks all previous arguments. If this was not the case, then it may occur that a previous argument is *reinstatiated*, meaning that it becomes accepted again because the argument attacking it is itself attacked.

Example 3: Decompose goal *Simulate*

The transcript excerpt of this example is shown in table 6 in the appendix and comes from transcript t_3. It consists of a discussion about the type of decomposition relationship for the goal *Simulate*.

The visualization of this discussion is shown in figure 9. Each GRL element on the right has a corresponding argument on the left. Moreover, the original argument for an AND-decomposition is attacked by the argument for the OR decomposition, and the new argument is linked to the decomposition relation in the GRL model.

Example 4: Reinstate actor *Development team*

The transcript excerpt of this example is shown in table 7 in the appendix and comes from transcript t_3. It consists of two parts: first participant P1 puts forth the suggestion to include actor *Development team* into the model. This is then questioned by person P2, who argues that the professor will develop the software, so there won't be any development team. However, in the second part, participant P2 argue that the development team should be considered, since the professor doesn't develop the software.

We formalize this using a *generic counterargument*, attacking the critical question. The first part of the discussion is shown in figure 10. We formalize the first statement as an instantiation of argument scheme AS0: *Actor development team is relevant*. This argument is

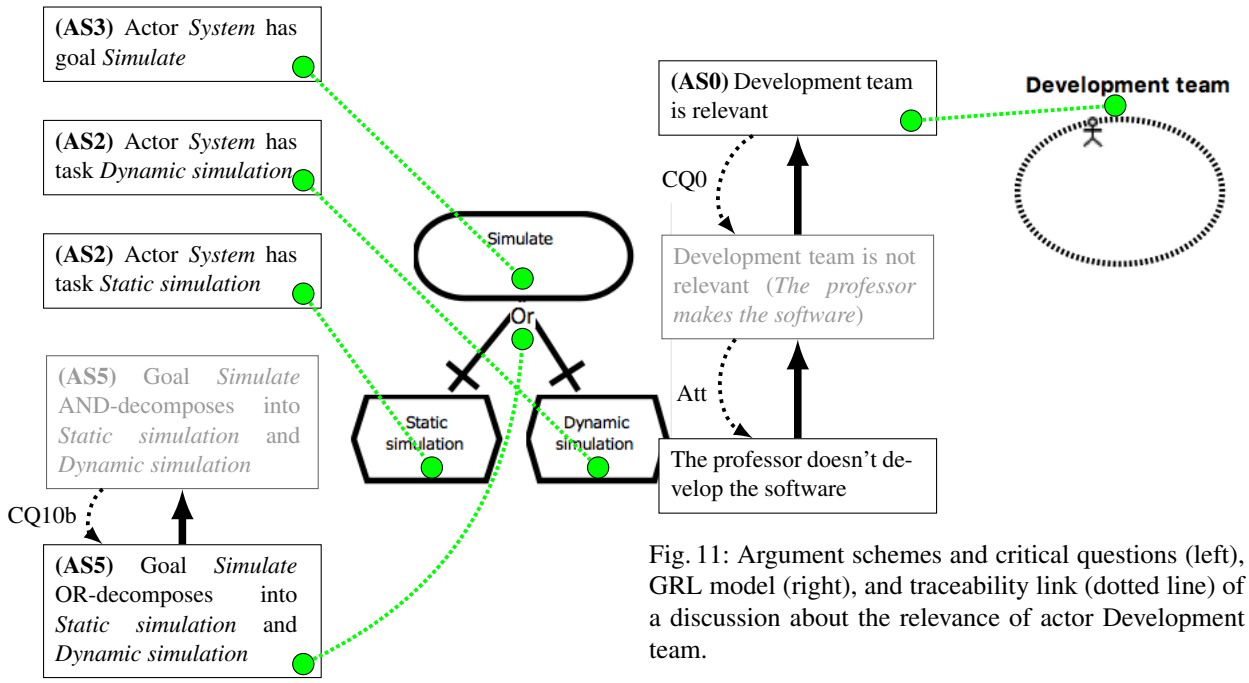

Fig. 9: Argument schemes and critical questions (left), GRL model (right), and traceability link (dotted line) of the goal decomposition example.

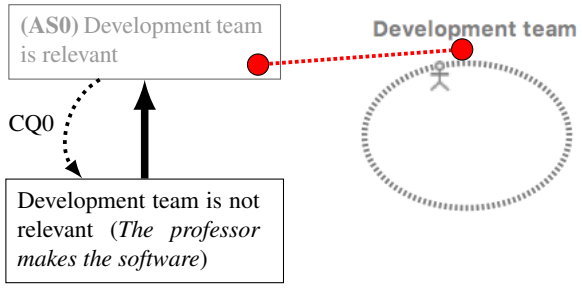

Fig. 10: Argument schemes and critical questions (left), GRL model (right), and traceability link (dotted line) of a discussion about the relevance of actor Development team.

then attacked by answering critical question CQ0: *Is actor development team relevant?* with *No*. This results in two arguments, AS0 and CQ0, where CQ0 attacks AS0. This is shown in figure 10, left image.

Figure 11 shows the situation after the counter argument has been put forward. The argument "The professor doesn't develop the software" now attacks the argument "*Development team* is not relevant (*The professor makes the software*)", which in turn attacks the original argument "*Development team* is relevant". As a result, the first and the last argument are both accepted, which causes the actor in the GRL model to be enabled again.

Fig. 11: Argument schemes and critical questions (left), GRL model (right), and traceability link (dotted line) of a discussion about the relevance of actor Development team.

5 The Logical Framework

In the previous section we developed a list of critical questions and argument schemes by analyzing transcripts of discussions about the development of an information system. The resulting list is shown in table 1. We also discussed various examples of applications of the different critical questions and all four different effects (right column of table 1): INTRO, DISABLE, REPLACE, and ATTACK.

The examples and corresponding visualizations of the previous section provided some insight into how to formalize argument schemes, critical questions, and their relationship with goal models. However, if we are to implement our framework into a tool, we require a more precise formalisation of these concepts. We choose to use formal logic to specify this, because argumentation has been studied in formal logic extensively for the past decades.

In the first subsection we develop a formal language to specify a GRL model. This language consists of propositions only, and a GRL model specified in this language can be specified as a logic program directly. In the second subsection we then provide formal definitions of all argument schemes. An instantiation of an argument scheme consists of a set of statement about a GRL

model. In the third subsection we formalize the critical questions as algorithms.

5.1 Formal Language for RationalGRL

Basic concepts set theory are adequate to define the entities in a GRL model and the relations between them. We begin with the representation for the elements.

Definition 1 (RationalGRL elements). *Each element of a goal model is identified with a non-negative integer. Formally: let* $Actors \cup IEs \cup Links \subseteq \mathbb{N}$ *denote the set of elements of a goal model, where*

- $Actors$ *is the set of actors,*
- IEs *is the set of intentional elements (softgoals, goals, tasks, and resources),*
- $Links$ *is the set of links (decomposition, contribution, and dependency).*

The three sets are pairwise disjoint, i.e. $Actors \cap IEs = Actors \cap Links = IEs \cap Links = \emptyset$.

Rationale and example: We identify GRL element with an identifier and not with a name, because it is possible that the name of the element changes, for instance by applying critical question CQ12a (clarification), see the example "Clarify task *Road pattern*" above. The GRL model of figure 9 can be formalized as follows:

$$Actors = \{1\},$$
$$IEs = \{2, 3, 4\},$$
$$Links = \{5\}.$$

Definition 2 (GRL intentional element type). *For an intentional element* $i \in IEs$*, if* i *is a softgoal, goal, task, or resource, then this is respectively denoted by* $softgoal(i), goal(i), task(i)$*, and* $resource(i)$.

Rationale and example: All intentional elements are collected in the set IEs. In order to distinguish softgoals, goals, tasks, and resources, we use the corresponding propositions. In our example this looks as follows:

$$goal(2),$$
$$task(3),$$
$$task(4).$$

Note we do not require such propositions for actors, since there is only one type of actor, and all actors are contained in the set $Actors$.

Definition 3 (Links). *Given a link* $i \in Links$*, we denote the links types as follows:*

- $contrib(i, src, dest, type)$ *is a contribution from* $src \in IEs$ *to* $dest \in IEs$*, where* $type \in \{+, -\}$ *means a positive resp. negative contribution.*
- $decomp(i, src, \{dest_1, \ldots, dest_n\}, type)$ *is a decomposition of* $src \in IEs$ *into* $\{dest_1, \ldots, dest_n\} \subseteq IEs$*, where* $type \in \{AND, OR, XOR\}$ *means respective an AND, OR, and XOR decomposition.*
- $dep(i, src, dest)$ *is a dependency from* $dest \in IEs$ *to* $dest \in IEs$.

Rationale and example: Similarly to the intentional elements, we distinguish between the different types of links between IEs using the corresponding propositions. The formalisation of the link in figure 9 looks as follows:

$$decomp(5, 2, \{3, 4\}, OR)$$

Definition 4 (Name of IE or Actor). *The name* p *of a element* $i \in IEs \cup Actors$ *is denoted by* $name(i, p)$.

Rationale and example: We formalize the description of an actor or an intentional element with a proposition $name(i, p)$. Continuing our example of formalizing figure 9, we have

$$name(1, \text{system}),$$
$$name(2, \text{simulate}),$$
$$name(3, \text{static simulation}),$$
$$name(4, \text{dynamic simulation}).$$

Note the decomposition link with identifier 5 does not have a name, since decompositions are already distinguished by their type (OR, XOR, AND).

Definition 5 (Elements of an actor). *Given an actor* $i \in Actors$ *and a element* $j \in IEs \cup Links$*, we use* $has(i, j)$ *to denote that element* j *belongs to actor* j.

Rationale and example: In order to denote that an intentional element or a link belongs to an actor, we use has statements. This looks as follows in our example:

$$has(1, 2),$$
$$has(1, 3),$$
$$has(1, 4),$$
$$has(1, 5).$$

Definition 6 (Disabled GRL element). *A disabled element* $i \in Actors \cup IEs$ *is denoted by* $disabled(i)$.

Rationale and example: If a GRL element corresponds to an argument that is rejected, then this argument should be disabled. For instance, the GRL model

in figure 10 can be formalized as follows:

$$Actors = \{1\},$$
$$name(1, \text{development team}),$$
$$disabled(1)$$

This concludes our definitions to formalize a GRL model. An advantage of our approach is that we can formalize a GRL model using a set of facts, which can be directly be formalized as a logic program. We will come back to this in the last part of this section.

5.2 Formal argumentation

In the next section, we formalize an argument as a set of statements about a GRL model in the language we introduced in the previous section. First, we formalize an argumentation framework and its semantics, which is used to determine sets of acceptable arguments.

Definition 7 (Argumentation framework). *An argumentation framework $AF = (Args, Att)$ consists of a set of arguments $Args$ and an attack relationship $Att : Args \times Args$, where $(A_1, A_2) \in Att$ means that argument $A_1 \in Args$ attacks arguments $A_2 \in Args$.*

Rationale and example: This is the standard definition used by Dung [**?**]. For instance, the argumentation framework from figure 2 can be formalized as:

$$Args = \{A_1, A_2, A_3\}$$
$$Att = \{(A_2, A_1), (A_1, A_3), (A_3, A_1)\}$$

TODO for Marc(by Marc): explain semantics

5.3 Algorithms for instantiating the argument schemes

We formalize an argument as a set of statements about a GRL model in the language we introduced in the previous section. For instance, $A = \{actor(0), name(0, student)\}$ is an argument stating that identifier 0 represent an actor, and that the name of this actor is *Student*. An argument scheme then corresponds to adding one or more arguments to the set of arguments.

In the following algorithms, we assume the following global variables:

- id the current highest identifier of the elements. This variable is increased for each new element that is added.
- $Args$ contains the set of all arguments.
- Att contains the set of all attack relations.

Algorithm 1 Applying AS0: Actor a is relevant

procedure $AS_0(a)$
  $id \leftarrow id + 1$
  $A \leftarrow \{actor(id), name(id, a)\}$
  $Args \leftarrow Args \cup A$
end procedure

Rationale and example: Algorithm 1 takes one argument, namely the name of the actor a. In figure 11, the application of the argument scheme $AS_($Development team$)$, results in one argument: $Args = \{\{actor(0), name(0, \text{Development team})\}\}$.

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

procedure $AS_1(a_{id}, n)$
  $id \leftarrow id + 1$
  $A \leftarrow \{resource(id), name(id, n), has(a_{id}, id)\}$
  $Args \leftarrow Args \cup A$
end procedure

Rationale and example: This argument scheme takes two arguments, the identifier a_{id} of the actor and the resource name n.

The formalisations of argument scheme AS2 to AS4 are exactly the same as AS1 but then for respectively tasks, goals, and softgoals. Therefore, we have omitted them here. We continue with argument scheme AS5.

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

procedure $AS_5(g_{id}, \{T_1, \ldots, T_n\}, type)$
  $T_{id} = \emptyset$
  for T_i in $\{T_1, \ldots, T_n\}$ **do**
    if $\exists_{A \in Args}\{task(t_{id}), name(t_{id}, T_i)\} \subseteq A$ **then**
      $T_{id} \leftarrow T_{id} \cup \{t_{id}\}$
    else
      $id \leftarrow id + 1$
      $A \leftarrow \{task(id), name(id, T_i)\}$
      $Args \leftarrow Args \cup A$
      $T_{id} \leftarrow T_{id} \cup \{id\}$
    end if
  end for
  $id \leftarrow id + 1$
  $A \leftarrow \{decomp(id, g_{id}, T_{id}, type)\}$
  $Args \leftarrow Args \cup A$
end procedure

Rationale and example: The procedure in algorithm 3 takes three arguments: g_{id} is the identifier of goal G, $T = (T_1, \ldots, T_n)$ is a list of decomposing task names,

and $type \in \{and, or, xor\}$ is the decomposition type. The for loop in the algorithm iterates over all tasks in list T, and if a task does not exist yet, an argument is added for it. This ensures that all tasks have corresponding arguments. Note that the algorithm assumes the goal already exists.

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

1: **procedure** $AS_6(t_{id}, s)$
2: **if** $\exists_{A \in Args}\{softgoal(i), name(i, s)\} \subseteq A$ **then**
3: $s_{id} \leftarrow i$
4: **else**
5: $id \leftarrow id + 1$
6: $A \leftarrow \{softgoal(id), name(id, t)\}$
7: $Args \leftarrow Args \cup A$
8: $s_{id} \leftarrow id$
9: **end if**
10: $id \leftarrow id + 1$
11: $A \leftarrow \{contr(id, t_{id}, s_{id}, pos)\}$
12: $Args \leftarrow Args \cup A$
13: **end procedure**

Rationale and example: The procedure in algorithm 4 takes two arguments: t_{id} is the identifier of task T, and s is the softgoal name that is contributed to. The if statements check if the softgoal exists already, and if not, an argument is added for it. This ensures that all softgoals have corresponding arguments. Note that the algorithm assumes the task already exists.

The algorithms for $AS7$ to $AS11$ all have a very similar structure as algorithm 4 and have therefore been omitted from this article.

5.4 Algorithms for answering the critical questions

We now develop algorithms for our critical questions. We start with the critical questions that have as effect DISABLE. All critical questions CQ0-CQ5a, CQ6a, CQ7a, CQ8, CQ11, and CQ12 fall into this category. The algorithm below applies when the critical question is answered affirmatively.

Algorithm 5 Applying DISABLE: Element i is disabled

1: **procedure** DISABLE(i)
2: $id \leftarrow id + 1$
3: $A \leftarrow \{disabled(i))\}$
4: $Args \leftarrow Args \cup A$
5: **end procedure**

Rationale and example: We the see disable operation is quite straightforward. It simply consists of adding an argument stating argument i is disabled.

Next we consider the REPLACE operation. This operation is less straightforward and cannot be captured by a single algorithm, since the operation depends on the type of argument scheme.

Algorithm 6 Answering CQ5b: "Does goal G decompose into any other tasks?" With: "Yes, namey into tasks $t_{n+1}, \ldots, t_k$"

1: **procedure** CQ5B($g_{id}, \{i_1, \ldots, i_n\}, type, \{t_1, \ldots, t_k\}$)
2: $T_{id} = \{i_1, \ldots, i_n\}$
3: **for** t_i in $\{t_1, \ldots, t_k\}$ **do**
4: **if** $\exists_{A \in Args}\{task(t_{id}), name(t_{id}, t_i)\} \subseteq A$ **then**
5: $T_{id} \leftarrow T_{id} \cup \{t_{id}\}$
6: **else**
7: $id \leftarrow id + 1$
8: $A \leftarrow \{task(id), name(id, t_i)\}$
9: $Args \leftarrow Args \cup A$
10: $T_{id} \leftarrow T_{id} \cup \{id\}$
11: **end if**
12: **end for**
13: $id \leftarrow id + 1$
14: $A_{new} = \{decomp(id, g_{id}, T_{id}, type)\}$
15: **for** A in $\{decomp(d, g_{id}, x, y)\} \subseteq A \mid A \in Args\}$ **do**
16: $Att \leftarrow Att \cup \{(A_{new}, A)\}$
17: **end for**
18: $Args \leftarrow Args \cup \{A_{new}\}$
19: **end procedure**

Rationale and example: Algorithm 6 is executed when critical question CQ5b is answered, which is a critical question for argument scheme AS5. Therefore, it assumes an argument for a goal decomposition of the form $decomp(d, g_{id}, \{i_1, \ldots, i_n\}, type)$ as already been generated (see algorithm 3). The goal of the algorithm is to generate a new argument of the form $decomp(d, g_{id}, \{i_1, \ldots, i_5\} \cup \{j_1, \ldots, j_k\}, type)$, where $\{j_1, \ldots, j_k\}$ are the identifiers of the additional decomposing tasks $\{t_1, \ldots, t_k\}$. The first for loop in line 3 check whether any of these tasks already exist, and if not, adds corresponding arguments for them. The list of all decomposing tasks is stored in T_{id}. The new argument for the decomposition link is then created in line 14. Finally, the last for loop on line 15 ensures that all previous arguments for decomposition links are attacked by the new argument. This ensures that only the new decomposition link will be accepted.

The REPLACE operations of CQ10a and CQ10b have a very similar structure as algorithm 6 and have therefore

been omitted. We continue with the last REPLACE operation left, namely CQ13.

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

procedure CQ13(i, n)
  $id \leftarrow id + 1$
  $ArgsN \leftarrow \{A \backslash \{name(i, x)\} \mid A \in Args, name(i, x) \in A\}$
  $A \leftarrow B \cup \{name(i, n)\}$ with $B \in ArgsN$
  $Args \leftarrow Args \cup \{A\}$
  for B in $ArgsN$ **do**
    $Att \leftarrow Att \cup \{(A, B)\}$
  end for
end procedure

Rationale and example: Algorithm 7 first collects all arguments contain a statement of the form $name(i, x)$ into the set $ArgsN$, where x can by any name. These are all the arguments that the new argument will attack, to ensure this is the only argument for a name of this element type. Since all such arguments are the same, except for the name proposition, we can take any such argument to construct a new one. The argument is added to the set of arguments, and the for loop ensures that all previous arguments are attacked.

We next turn to the INTRO operations, which are very similar to the algorithms for argument schemes. The main difference is that when answering a critical question, we can already assume an argument has been created, so we do not have to create it again.

Algorithm 8 Answering CQ6b: "Are there alternative ways of contributing to the same softgoal s_{id}?" With: "Yes, namely by doing task t"

procedure CQ6B(s_{id}, t)
  if $\exists_{A \in Args}\{task(i), name(i, t)\} \subseteq A$ **then**
    $t_{id} \leftarrow i$
  else
    $id \leftarrow id + 1$
    $A \leftarrow \{task(id), name(id, t)\}$
    $Args \leftarrow Args \cup A$
    $t_{id} \leftarrow id$
  end if
  $id \leftarrow id + 1$
  $A \leftarrow \{contr(id, t_{id}, s_{id}, pos)\}$
  $Args \leftarrow Args \cup A$
end procedure

Rationale and example: Comparing algorithm 9 with algorithm 4 shows that the two algorithms are indeed very similar.

The other critical questions with effect INTRO, namely CQ6c, CQ6d, CQ7b, and CQ9 all have a very similar structure to algorithm 9 and have therefore been omitted.

Our final critical question is the generic counterargument, which is very simple.

Algorithm 9 Generic counterargument to argument A

procedure ATTACK(A)
  $A_{new} = \{\}$
  $Args \leftarrow Args \cup \{A_{new}\}$
  $Att \leftarrow Att \cup \{(A_{new}, A)\}$
end procedure

5.5 Some examples

TODO for Marc(by Marc): Revisit the examples

6 The RationalGRL Tool

7 The RationalGRL Methodology

The methodology we propose in this paper is visualized in figure 12. There are two main activities, depicted in grey, namely "goal modeling" and "argumentation". These are two separate activities that are being done in parallel.

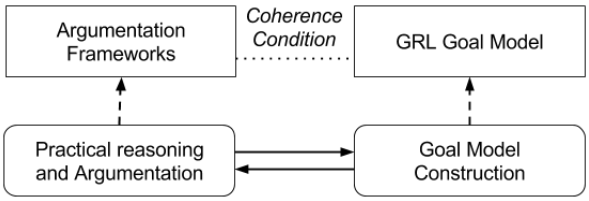

Fig. 12: The RationalGRL Methodology

8 Related Work

There are several contributions that relate argumentation-based techniques with goal modeling. The contribution most closely related to ours is the work by Jureta *et al.* [**?**]. This work proposes "Goal

Argumentation Method (GAM)" to guide argumentation and justification of modeling choices during the construction of goal models. One of the elements of GAM is the translation of formal argument models to goal models (similar to ours). In this sense, our RationalGRL framework can be seen as an instantiation and implementation of part of the GAM. One of the main contribution of RationalGRL is that it also takes the acceptability of arguments as determined by the argumentation semantics [**?**] into account when translating from arguments to goal models. RationalGRL also provides tool support for argumentation, i.e. Argument Web toolset, to which OVA belongs [**?**], and for goal modeling, i.e. jUCMNav [**?**]. Finally, RationalGRL is based on the practical reasoning approach of [**?**], which itself is also a specialization of Dung's [**?**] abstract approach to argumentation. Thus, the specific critical questions and counterarguments based on these critical question proposed by [**?**] can easily be incorporated into RationalGRL.

RationalGRL framework is also closely related to frameworks that aim to provide a design rationale (DR) [**?**], an explicit documentation of the reasons behind decisions made when designing a system or artefact. DR looks at issues, options and arguments for and against the various options in the design of, for example, a software system, and provides direct tool support for building and analyzing DR graphs. One of the main improvements of RationalGRL over DR approaches is that RationalGRL incorporates the formal semantics for both argument acceptability and goal satisfiability, which allow for a partly automated evaluation of goals and the rationales for these goals.

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

9 Conclusions and Future Work

In this paper, we developed and implemented a framework to trace back elements of GRL models to arguments and evidence that derived from the discussions between stakeholders. We created a mapping algorithm from a formal argumentation theory to a goal model, which allows us to compute the evaluation values of the GRL IEs based on the formal semantics of the argumentation theory.

There are many directions of future work. There are a large number of different semantics for formal argumentation, that lead to different arguments being acceptable or not. It would be very interesting to explore the effect of these semantics on goal models. Jureta *et al.* develop a methodology for clarification to address issues such as ambiguity, overgenerality, synonymy, and vagueness in arguments. Atkinson *et al.* [**?**] define a formal set of critical questions that point to typical ways in which a practical argument can be criticized. We believe that critical questions are the right way to implement Jureta's methodology, and our framework would benefit from it. In addition, currently, we have not considered the *Update* step of our framework (Figure **??**). That is, the translation from goal models to argument diagrams is still missing. The *Update* step helps analysts change parts of the goal model and analyze its impact on the underlying argument diagram. Finally, the implementation is currently a browser-based mapping from an existing argument diagramming tool to an existing goal modeling tool. By adding an argumentation component to jUCMNav, the development of goal models can be improved significantly.

Acknowledgments

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

A Transcripts excerpts

Respondent	**Text**	**Annotation**
0:15:11.2 (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"
0:15:34.7 (P2)	Yeah, road. Intersection, add traffic lights	
0:15:42.3 (P1)	Well, all intersection should have traffic lights so it's	**[21 critical question CQ12 for 20]** Is the task "Add traffic light" useful/relevant?
0:15:44.9 (P2)	Yeah	
0:15:45.2 (P1)	It's, you don't have to specifically add a traffic light because if you have	**[22 answer to 22]** Not useful, because according to the specification all intersections have traffic lights.
0:15:51.4 (P2)	They need-	

Table 4: Adding tasks, disabling useless task "Add traffic light" (transcript t_1)

Respondent	Text	Annotation
0:17:39.5 (P1)	And in that process there are activities like create a visual map, create a road	**[14 task (AS2)]** Student has task "Create road"
0:24:36.0 (P3)	And, well interaction. Visualization sorry. Or interaction, I don't know. So create a visual map would have laying out roads and a pattern of their choosing. So this would be first, would be choose a pattern.	**[31 critical question CQ?? for 14]** Is Task "Create road" clear? **[32 answer to 31]** no, according to the specification the student should choose a pattern.
0:24:55.4 (P1)	How do you mean, choose a pattern	
0:24:57.5 (P3)	Students must be able to create a visual map of an area, laying out roads in a pattern of their choosing	**[32a REPLACE]** "Create road" becomes "Choose a pattern"
0:25:07.5 (P1)	Yeah I'm not sure if they mean that. I don't know what they mean by pattern in this case. I thought you could just pick roads, varying sizes and like, broads of roads.	**[33 critical question CQ?? for 32a]** Is "Choose a pattern" clear? **[34 answer to 33]** No, not sure what they mean by a pattern.
0:25:26.0 (P3)	No yeah exactly, but you would have them provide, it's a pattern, it's a different type of road but essentially you would select- how would you call them, selecting a-	**[34a REPLACE]** "Choose a pattern" becomes "Choose a pattern preference"
0:25:36.3 (P1)	Yeah, selecting a- I don't know	
0:25:38.0 (P3)	Pattern preference maybe? As in, maybe we can explain this in the documentation	
0:25:43.9 (P1)	What kind of patterns though. Would you be able to select	**[35 critical question CQ?? for 34a]** Is "Choose a pattern preference" clear? **[36 answer to 35]** no, what kind of pattern?
0:25:47.4 (P3)	Maybe, I don't know it's-	**[36a rename]** "Choose a pattern preference" becomes "Choose a road pattern"
0:25:48.5 (P1)	[inaudible] a road pattern	

Table 5: Clarifying the name of a task (transcript t_3)

Respondent	Text	Annotation
0:18:55.7 (P1)	Yeah. And then two processes, static, dynamic and they belong to the goal simulate.	**[17 goal (AS3)]** Actor "System" has goal "Simulate" **[18 task (AS2)]** Actor "System" has task "Static simulation" **[19 task (AS2)]** Actor "System has task "Dynamic simulation" **[20 decomposition (AS?)]** Goal "Simulation" AND-decomposes into "Static simulation" and "Dynamic simulation"
0:30:10.3 (P1)	Yeah. But this is- is this an OR or an AND?	**[26 critical question CQ10b for 20]** Is the decomposition type of "simulate" correct? **[27 answer to 26]** No, it should be an OR decomposition.
0:30:12.6 (P2)	That's and OR	
0:30:14.3 (P3)	I think it's an OR	
0:30:15.4 (P1)	It's for the data, it's an OR	
0:30:18.1 (P3)	Yep	

Table 6: Incorrect decomposition type for goal *Simulate* (transcript t_3)

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

Table 7: Discussion about the relevance of an actor (transcript t_3)

B GRL Specification

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

% Disabled elements of actor Teacher
disabled(45).

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

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

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