RationalGRL: A Framework for Argumentation and Goal Modeling

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

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

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

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

Abstract. Goal modeling languages capture the relations between an information system and its environment using high-level goals and their relationships with lower level goals and tasks. The process of constructing a goal model usually involves discussions between a requirements engineer and a group of stakeholders. While it is possible to capture part of this discussion process in a goal model, for instance by specifying alternative solutions for a goal, not all of the arguments can be found in the resulting model. For instance, 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 a goal modelingn approach. 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

1.1 Requirements Engineering and Goal Modeling

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

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 [51]. These activities fall under the umbrella of goal modeling. The are a large number of established RE methods using goal models in the early stage of requirements analysis (e.g., [24, 13, 11, 9, 8], overviews can be found in [42, 21]). Several goal modeling languages have been developed in the last two decades as well. The most popular ones include $i*$ [52], Keep All Objects Satisfied (KAOS) [44], the NFR framework [9], TROPOS [16], the Business Intelligence Model (BIM) [18], and the Goal-oriented Requirements Language (GRL) [2].

1.2 Problem Statement

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

First, it is well-known that stakeholders' preferences are rarely absolute, relevant, stable, or consistent [25]. 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 straightforward to remind stakeholders of their previous opinions which results in having 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.

Second, other stakeholders, such as new developers on the team 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 or at the design and development phase. If these users 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.

Third, alternative and different ideas and opposing views that could potentially have led to different goal diagrams could be 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 merely contains 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 [10].

Fourth and lastly, in goal models in general, "rationale" behind any decision is static and does not immediately impact the goal models when they change. That is, current goal modeling languages have limited support for reasoning about changing beliefs and opinions, and their effect on the goal model. A stakeholder may change his or her opinion, but it is not always directly clear what its effect is on the goal model. Similarly, a part of the goal model may change, but it is not possible to formally reason about the consistency of this new goal model with the underlying beliefs and arguments in the goal modeling languages. 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 beliefs put forward by another group of stakeholders.

1.3 Research Questions

The aim of this research is to resolve the above issues by developing a framework with tool-support, which combines goal modeling approaches with argumentation techniques from Artificial Intelligence (AI) research [4]. We have identified five important requirements for our framework:

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

The five requirements above serve as the success criteria of our approach. That is, if our framework satisfies these requirements, we reach our goal.

In this context, the main research question is:

> **RQ.** What are the constructs, mechanisms and rules for developing a framework that formally captures the discussions between stakeholders such that it can generate goal models?

To answer the research question above and solve some of the concerns mentioned in Subsection 1.2, we propose a framework called RationalGRL. The RationalGRL framework combines the Goal-oriented Requirements Language (GRL) with a technique from argumentation and discourse modeling called *argument schemes* [49].

In past, we introduced our preliminary steps towards RationalGRL framework [46, 45, 48] with its tool-support [47]. In our previous work, argument diagrams are integrated with GRL models to allow better representation of stakeholder's arguments and discussions. In our current paper, we improve RationalGRL framework by adding argument schemes and critical questions to argumentation framework and provide traceability links between the argumentation framework and GRL. For this, we modify and extend our metamodel to specify how our framework extends GRL, and we use this as the basis for a prototype web-based implementation [1]. We develop an initial list of argument schemes and critical questions by analyzing a set of transcripts containing discussion about an information system. Our framework is fully extensible, meaning that argument schemes and critical questions can be added by users for specific problem domains. We develop a methodology for using RationalGRL, which consists of developing goal models and posing arguments in an integrated way. This methodology is intended to be used by practitioners in the field. We formalize RationalGRL using propositional logic, and develop algorithms for adding goal elements and arguments. We illustrates the steps of our framework and how the framework can be applied in real cases with a traffic simulator example.

1.4 Organization

This article is organized as follows. Section 2 contains background and introduces our running example, the Goal-oriented Requirements Language (GRL) [2], and argument schemes. Section 3 provides a brief and high-level overview of our framework, together with a meta-model and the methodology. Section 4 contains an in depth explanation of how we obtained an initial set of argument schemes by annotating transcripts from discussions about an information system, and in Section 5 we provide several examples of these arguments schemes. In Section 6 we formalize RationalGRL and our arguments schemes using propositional logic and we use well-known argumentation semantics to compute which arguments are accepted and which are rejected. We also develop various algorithms for the argument schemes in this section. In Section 7 we provide a brief overview of the prototype tool we developed for RationalGRL. Finally, Section 8 contains a discussion, covering related work, future work, and a conclusion.

2 Background: Goal-oriented Requirements Language and argument schemes

In this section, we first introduce our running example, after which we introduce the Goal-oriented Requirements Language (GRL) [2], which is the goal modeling language we use to integrate with the argumentation framework. Next, we introduce argument schemes, and in particular, we discuss the *practical reasoning argument scheme (PRAS)* [4], which is an argument scheme that is used to form arguments and counter-arguments about situations involving goals. Finally, we give an overview of the integration between PRAS and GRL.

[1] The tool is available at ¡GITHUB LINK¿

2.1 Running example: Traffic Simulator

We use the traffic simulator design case to explain the concepts and the framework in this paper. Traffic simulator design is the topic of discussion in the transcripts used for the argumentation framework. In this exercise, designers are provided with a problem description, requirements, and a description of the desired outcomes. The problem description is given in full in Appendix A, and is summarized as follows: 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 how various theories (such as queuing theory) around traffic lights work, a software analyst is hired to specify the goal and requirements of the system. To this end, a piece of software will be developed in which students can create visual maps of an area, regulate traffic, and so forth. The original version of the problem descrption [40] is well known in the field of design reasoning and it has been used in a workshop[2]. Transcripts of this workshop have been analyzed in detail [33]. Although the concepts of traffic 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.

In this Section and in Section **??**, we focus on the simplified example of traffic simulator case for simplicity. In Section **??** we provide the detailed example of the traffic simulator and its model.

2.2 Goal-oriented Requirements Language (GRL)

GRL is a visual modeling language for specifying intentions, business goals, and *non-functional requirements* of multiple stakeholders [2]. GRL is part of the User Requirements Notation, an ITU-T standard, that combines goals and non-functional requirements with functional and operational requirements (i.e. use case maps) in one. GRL can be used to specify alternatives that have to be considered, decisions that have been made, and rationales for making decisions. A GRL model is a connected graph of intentional elements that optionally are part of the actors. All the elements and relationships used in GRL are shown in Figure 2.

Figure 1 illustrates a simplified GRL diagram from the traffic simulator design exercise (a full example is shown in Figure 6 and will be discussed in Section 4). An actor (⬭) represents a stakeholder of a system or the system itself (`Traffic Simulator`, Figure 1). Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be

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

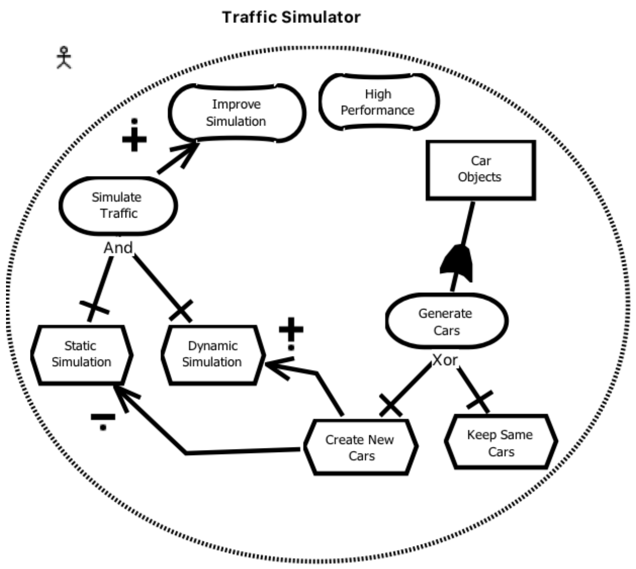


Fig. 1: Partial GRL Model of Traffic Simulator Example

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. `Improve Simulation`) are often related to non-functional requirements, whereas goals (such as `Simulate Traffic`) are related to functional requirements. Tasks (⬡) represent solutions to (or operationalizations of) goals and softgoals. In Figure 1, some of the tasks are `Create new cars` and `Keep same cars` which are alternative way of achieving the goal, `Generate Cars`. In order to be achieved or completed, softgoals, goals, and tasks may require resources (▭) to be available (e.g., `Car Objects`).

Different links connect the elements in a GRL model. AND, IOR (Inclusive OR), and XOR (eXclusive OR) decomposition links (+—) allow an element to be decomposed into sub-elements. In Figure 1, the goal `Generate cars` is XOR-decomposed to the tasks `Create new cars` and `Keep same cars`. Contribution links (⟶) indicate desired impacts of one element on another element. A contribution link has a qualitative contribution type or a quantitative contribution. Task `Create new cars` has a *help* qualitative contribution to the task `Dynamic Simulation`. Dependency links (—■—) model relationships between actors. Here, the goal `Generate Cars` depends on the resource `Car Objects`.

GRL is based on $i*$ [52] and the NFR Framework [9], but it is not as restrictive as $i*$. Intentional elements and links can be more freely combined, the notion of agents is replaced with the more general notion of actors, i.e., stakeholders, and a task does not necessarily have to be an activity performed by an actor, but may also describe properties of a solution. GRL has a well-defined syntax and semantics. Furthermore, GRL provides support for providing a scalable and consistent representation of multiple views/diagrams of the same goal model (see [15, Ch.2] for more details). GRL is also linked to Use Case Maps via URNLinks (▶) which provide traceability between concepts and instances of the goal model and behavioral design models. Multiple views and traceability links are a good fit with our current research: we aim to add traceability links between intentional elements and their underlying arguments.

GRL has six evaluation algorithms which are semi-automated and allow the analysis of alternatives and design decisions by calculating the satisfaction value of the intentional elements across multiple diagrams quantitatively, qualitatively or in a hybrid way. The satisfaction values from intentional elements in GRL can also be propagated to the elements of Use Case Maps. jUCMNav, GRL tool-support, also allows for adding new GRL evaluation algorithms [31]. GRL also has the capability to be extended through metadata, links, and external OCL constraints. This allows GRL to be used in many domains without the need to change the whole modeling language. This feature also helps us apply our argumentation to other domains such as compliance, which we explain in more detail in Section 8.2.

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

– Why does actor `Traffic Simulator` have softgoal `High Performance`, which is not linked to any of the goals `Generate Cars` and `Simulate Traffic`?
– What does `Keep same cars` mean?
– Why does task `Create New cars` contribute negatively to `Static simulation` and positively to `Dynamic simulation`?
– Why does `Simulate Traffic` AND-decompose into two tasks?

These are the type of the questions that we cannot answer by just looking at GRL models. The model in Figure 1 does not contain information about discussions that lead to the resulting model, such as various clarification steps for the naming, or alternatives that have been considered for the relationships. In this article we aim to address this shortcoming.

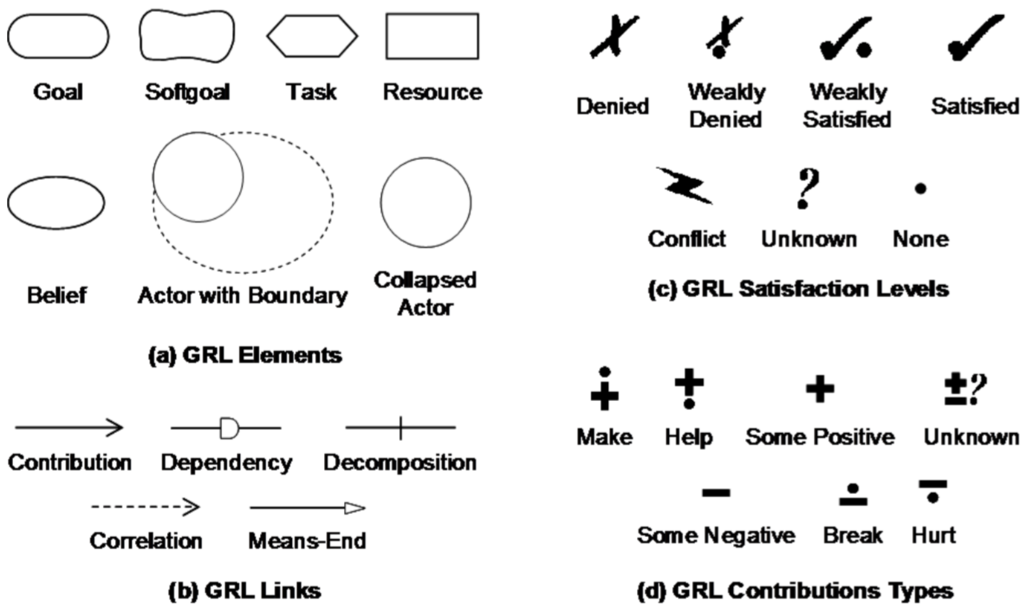


Fig. 2: Basic elements and relationships of GRL

2.3 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. [34, 50]) and artificial intelligence [6, 4]. One approach is to capture practical reasoning using arguments schemes and critical questions [50]. Applying an argument schemes results in an argument in favor of, for example, taking an action. This argument can, then, be tested with critical questions about, for instance, whether the action is possible given the situation. A negative answer to such a question leads to a counterargument to the original argument for the action.

Atkinson et al. [4] develop an argument scheme for practical reasoning, which they call the *Practical Reasoning Argument Scheme* (PRAS). This argument scheme is as follows[3]:

We have goal G,
Doing action A realizes goal G,
Which will contribute positively to the softgoal S
Therefore
We should perform action A

[3] The original argument schemes uses slightly different terminology: we have replaced *value* with *softgoal*. Our results do not depend on these adjustments, but they make the relation between PRAS and GRL more clear.

The capital letters G, A, and S are variables, which can be instantiated with concrete goals, actions, and softgoals. Once instantiated, we obtain an argument. We can for instance instantiate the argument scheme above with the elements of Figure 1 as follows:

We have goal `Simulate Traffic`,
Doing action `Dynamic Simulation` realizes goal `Simulate Traffic`,
Which contribute positively to the softgoal `Improve Simulation`
Therefore
We should perform action `Dynamic Simulation`.

Atkinson *et al.* [4] 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 realize the desired goal?
2. Are there alternative ways of realizing the same goal?
3. Are there alternative ways of contributing to the same softgoal?
4. Does performing the action contribute negatively to some other softgoal?
5. Is the action possible?
6. Can the desired goal be realized?
7. Is the softgoal indeed a legitimate softgoal?

These critical questions can point to new arguments that might counter the original argument. Take, for example, critical question 1: if we find that `Dynamic Simulation` actually does not realize goal `Simulate Traffic`, we can form a counterargument.

Another way to elaborate an argument for an action is to suggest an alternative action that realizes the same goal (question 2) or an alternative goal that contributes to the same softgoal (question 3). For example, can argue that another task `Semi-dynamic Simulation` also realizes the goal `Simulate Traffic`.

In argumentation, counterarguments are said to *attack* the original arguments (and sometimes vice versa). In the work of Atkinson et al. [4], arguments and their attacks are captured as an *argumentation framework* of arguments and attack relations. Given an argumentation framework, once can compute very precisely which arguments are accepted and which are rejected using argumentation semantics [14]. These semantics have been studied extensively in AI, and are currently a popular topic of research (see [41] for an overview).

2.4 Combining PRAS and GRL

The usefulness of argument schemes, and PRAS in particular, for the analysis of practical reasoning situations has been shown in different areas such as e-democracy [7], law [3], planning [27] and choosing between safety critical actions [39]. In this article, we aim at capturing the stakeholder's discussions as formal argumentation based on PRAS to decide whether intentional elements and their relationships are shown in the resulting goal model. This approach provides a rationalization to the elements of the goal model in terms of underlying arguments, and furthermore, it helps understanding why certain other elements have been accepted and others have been rejected.

Argumentation schemes and their associated critical questions are very well suited for modeling discussions about a goal model: as Murukannaiah et al. [29] have shown, they can guide users in systematically deriving conclusions and making assumptions explicit. This can also be seen from the obvious similarities between PRAS (actions, goals, values) and GRL (tasks, goals, softgoals) in the example above.

However, there are also some differences between PRAS and GRL. Not all elements and relationships of GRL fit into PRAS. For instance, PRAS does not have a notion of "resource", and many of the relationships of GRL do not occur in PRAS. Furthermore, it is not directly clear whether the critical questions as proposed by Atkinson can actually apply to GRL. Therefore, we develop our own set of argument schemes and critical questions in Section 4 by analyzing transcripts of discussions about the traffic simulator. Before doing so, we give an overview of our framework in the next section.

3 RationalGRL Methodology and Metamodel

In this section we present a high-level overview of the RationalGRL framework. In the first subsection, we present a methodology, specifying how practitioners can use RationalGRL to create goal models with traceability links to underlying arguments. In the second subsection, we link the new argumentation elements and relations to the existing elements and relations of GRL in a metamodel. This metamodel serves as the specification for an implementation. In Section 7 we briefly discuss our prototype implementation based on this metamodel.

3.1 RationalGRL Methodology

As mentioned in Section 1, the RationalGRL framework uses concepts from practical reasoning argument scheme (PRAS) to help integrating goal models with the detailed discussions and arguments the stakeholders pose during the analysis phase. That is, the RationalGRL framework includes two main parts: Argumentation modeling and GRL modeling.

For the GRL part, we first need to create the "initial" GRL model by analyzing the non-functional requirements in the requirements specification document and by refining the high-level goals into operationalized tasks. For the argumentation part, arguments and counterarguments are put forward about various parts of the goal model. These two parts, GRL and argumentation, are developed iteratively and each side can impact the other side so that the models can be refined or new critical questions and argument schemes can be instantiated. For example, answering a critical question *Is the task* A *possible?*, instantiated from the argumentation model, can result in removing or adding a task in the GRL model. Similarly, if, for example, we add a new intentional element to the GRL model, it can lead to a new critical question relevant to this intentional element and its relationships. Figure 3 presents an overview of RationalGRL framework with its components and their relationships. 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.

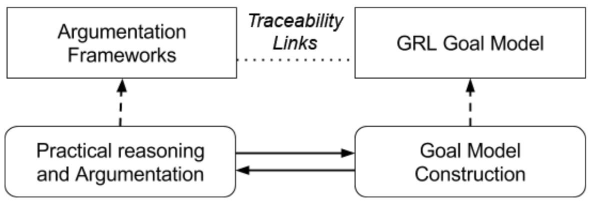


Fig. 3: The RationalGRL Framework

We propose the following methodology (shown in Figure 4) to develop an instance of the RationalGRL framework. Here we assume that the initial GRL models have been created based on the requirements specification documents and the discussions of the stakeholders. The rest of the steps are as follows:

(1) Instantiate Argument Schemes (AS) – In this step, we start from the list of arguments schemes of the argumentation framework. We select an intentional element from the initial GRL model that we want to analyze and we instantiate a relevant argument scheme from the already existing list of argument schemes or by adding a new one. 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 instantiating an argument scheme, we invoke related critical questions to attack the argument by counter-arguments. Each argument scheme includes one or more critical questions. 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. Thus, it is possible to go back and forth between this step and the step (1).

(3) Decide on Intentional Elements and their Relationships – By answering a critical question, one of the four following cases can occur: INTRO, DISABLE, REPLACE or ATTACK. Any of these cases can impact the arguments and corresponding GRL intentional elements. INTRO means that a new argument scheme is created. That means, the current argument scheme related to the critical question does not get attacked. In the case of DISABLE, the intentional element or its related links are disabled or removed from the models. 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. ATTACK is a generic counterargument which attacks any argument with another argument when new evidence occurs.

(4) Modify GRL Models – In this step, we modify the GRL models based on the situation of step (3). That is, one of the following situation can happen with respect to the initial GRL model: 1) a new intentional element or a new link is introduced; 2) an existing intentional element or an existing link gets disabled (removed) from the model; or 3) an existing intentional element or link is replaced by a new one. This results in a new modified GRL. The new GRL model can then impact the argument schemes and instantiate another argument scheme (Step (1)).

We can continue these four steps until there is no more intentional element or link to analyze or we reach a satisfactory model.

3.2 Metamodel

Figure 5 depicts the RationalGRL metamodel linking the main elements of our argumentation extension to the main GRL elements. We describe the metamodel bottom-up, starting with the GRL package.

The GRL package of our metamodel consists of the core GRL concepts, which constitute a part of the URN metamodel from Recommendation Z.151 [19]. These concepts represent the abstract grammar of the language, independently of the notation. This metamodel also formalizes the GRL concepts and constructs introduced earlier[4].

The GRL specification 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). Note that actors can be connected through links as well, which is done with Dependency links.

The Argumentation package depicts the concepts we introduced in the previous sections. An ArgumentScheme represents an (uninstantiated) scheme containing variables that can be replaced with intentional

[4] Note that for readability, some GRL concepts ave been omitted from the figure. For instance, a GRL contribution can have a qualitative strength, ranging from “Make” to “Break”. Since these concepts are not relevant to our framework, they have been omitted, but none of the results of this article depend on this.

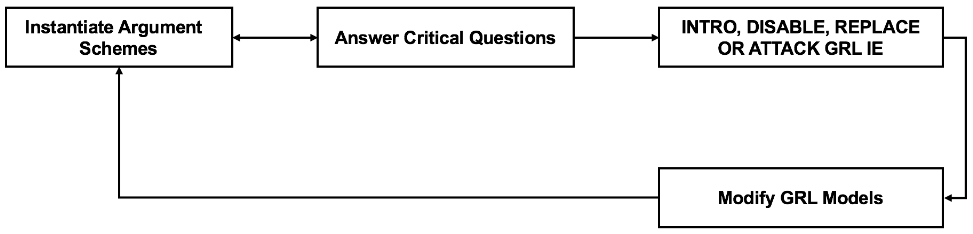

Instantiate Argument Schemes
Answer Critical Questions
INTRO, DISABLE, REPLACE OR ATTACK GRL IE
Modify GRL Models

Fig. 4: The RationalGRL Methodology

elements. CriticalQuestions are possible ways to attack or elaborate an argument scheme. As such, each critical question applies to exactly one scheme, but each scheme can be elaborated or attacked through multiple critical questions. When an argument scheme is instantiated, we obtain an Argument. Therefore, each argument is associated with exactly one scheme, but a scheme can be instantiated in multiple ways. When a critical question is answered, we obtain an AttackLink. Each AttackLink is associated with at most one critical question, but a critical question can be used to attack multiple arguments. Note that an AttackLink can also be associated with no critical questions. This allows the user to create attacks between arguments, which do not necessarily correspond to one of the critical questions. A RationalGRLDiagram is composed out of arguments and attack relations.

There is only one link between the GRL package and the Argumentation package, but it is a very important one. The link specifies that each GRLModelElement is in fact an argument. This means that each model element inherits the AcceptStatus as well, allowing GRL elements to be accepted, rejected, or undecided. 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, but can come from an external source, for instance a piece of evidence or an expert opinion. We will see various examples of such arguments in the next sections.

4 Argument Schemes for Goal Modeling

In the previous section, we provided an overview of RationalGRL methodology. In this section, we discuss our argument schemes and critical questions in more detail. It is important to note that the list of argument schemes we present in this section is not exhaustive. It is an initial list that we have obtained by annotating transcripts, but our framework is fully extensible, meaning that new argument schemes and critical questions can be added depending on the problem domain.

The list of argument schemes and critical questions that we have obtained from our analysis is shown in Table 1. The first four argument schemes (AS0-AS4) are arguments for an element of a goal model, the next seven (AS5-AS11) are related to relationships, the next two (AS12-AS13) are for intentional elements in general, and the last is (Att) is a generic counterargument for any type of argument that has been put forward.

As we have already discussed in Section 3, answering critical questions can have different impacts on the model. The right column in Table 1 shows the impact of answering each critical question affirmatively. As mentioned earlier, answering a critical question can create an argument disabling the corresponding GRL element of the attacked argument scheme (DISABLE); it can create an argument introducing a new GRL element (INTRO); it can replace the GRL element corresponding to the original argument (REPLACE), or it can simply attack an argument directly (ATTACK).

In the following subsections, we first provide details of the transcript annotation process with concrete examples (see Appendix B for transcript excerpts) and then we analyze our results.

4.1 Details experiment

The transcripts we used are created as part of two master theses on improving design reasoning [36, 35].

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 goal of the experiment is to design a traffic simulator. Participants were asked to use

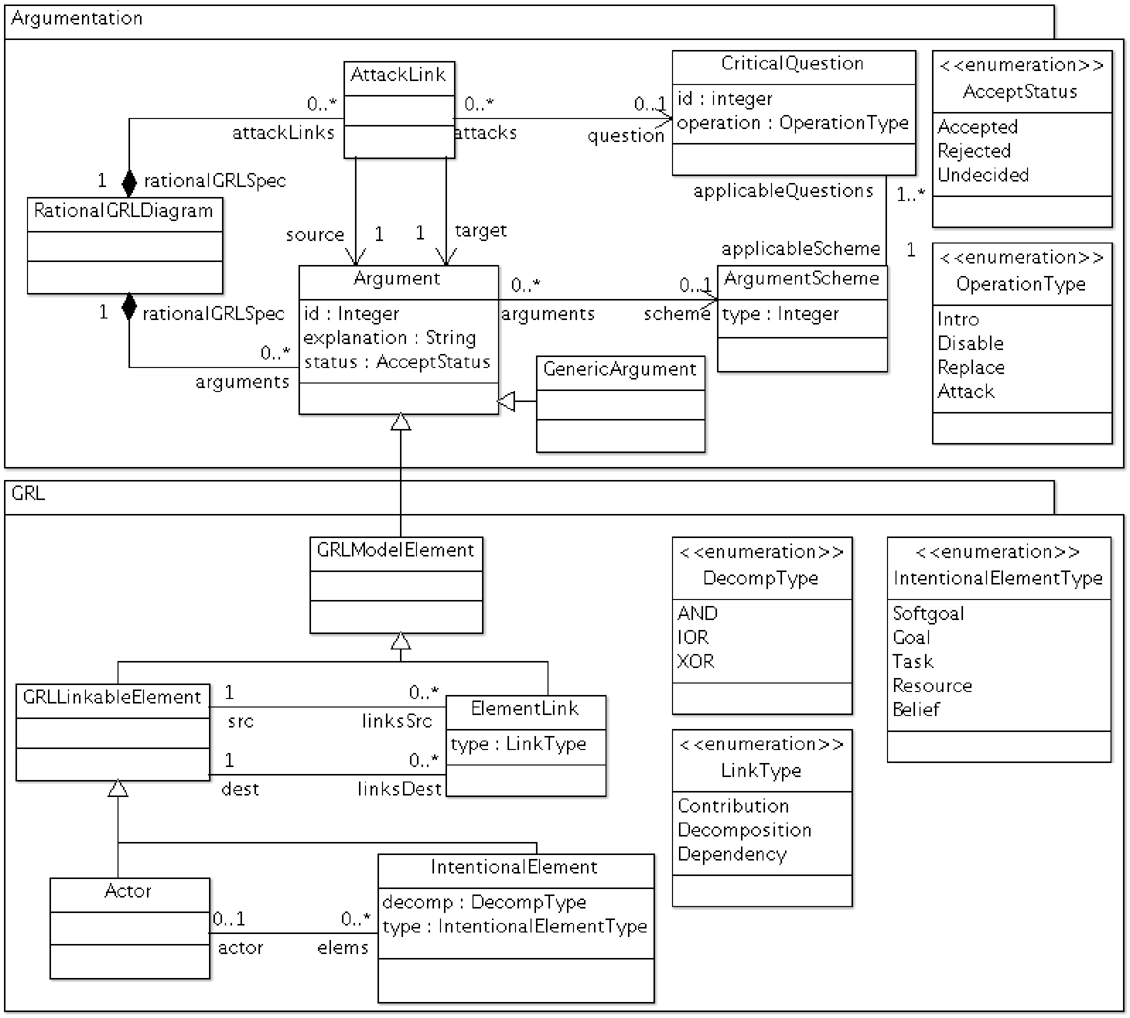


Fig. 5: The RationalGRL metamodel

a think-aloud method during the design session. The assignment was slightly adjusted to include several viewpoints as an end product in order to conform to the course material [5]. The full problem descriptions can be found in Appendix A. 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. The entire discussion was documented in transcripts. The details of the transcripts are shown in Table 2.

Process of Extracting Argument Schemes and Critical Questions To develop the list of argument schemes and critical questions, we started with an initial list of 8 argument schemes and 18 critical questions that we derived from PRAS (AS1-AS4, AS6-AS9 of Table 1). We, then, annotated transcripts with arguments and critical questions from this initial list. If there were arguments or critical questions that did not appear in the initial list, we added them to the list.

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.

4.2 Experiment Results

All original transcripts, annotations, and models are available on the following website **TODO for Marc**(by

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).

	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.

Marc): Change URL and put everything together with the tool:

`http://www.github.com/marcvanzee/RationalArchitecture`

We also provide excerpts of the annotation in Appendix B, and most of the examples we use in this article come from the transcripts.

We found a total of 159 instantiation of argument schemes AS0-AS11 in transcripts. The most used argument scheme was AS2: "Actor A has task T", however, each argument scheme is found in transcripts at least twice (Table 3). A large portion (about 60%) of argument schemes involved discussions about 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 its relevance (CQ12, CQ13).

We manually created GRL models from argument schemes and critical questions found in each transcript, to verify whether the arguments put forward by the participants were sufficiently informative. An example of such a model is shown in Figure 6. Depending on whether the underlying argument for an element is accepted or rejected, we added a green or a red dot to the element. A green dot indicates that 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 is 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 an argument for them.

Answering a critical question can have four different effects on the original argument and the corresponding GRL element:

– 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

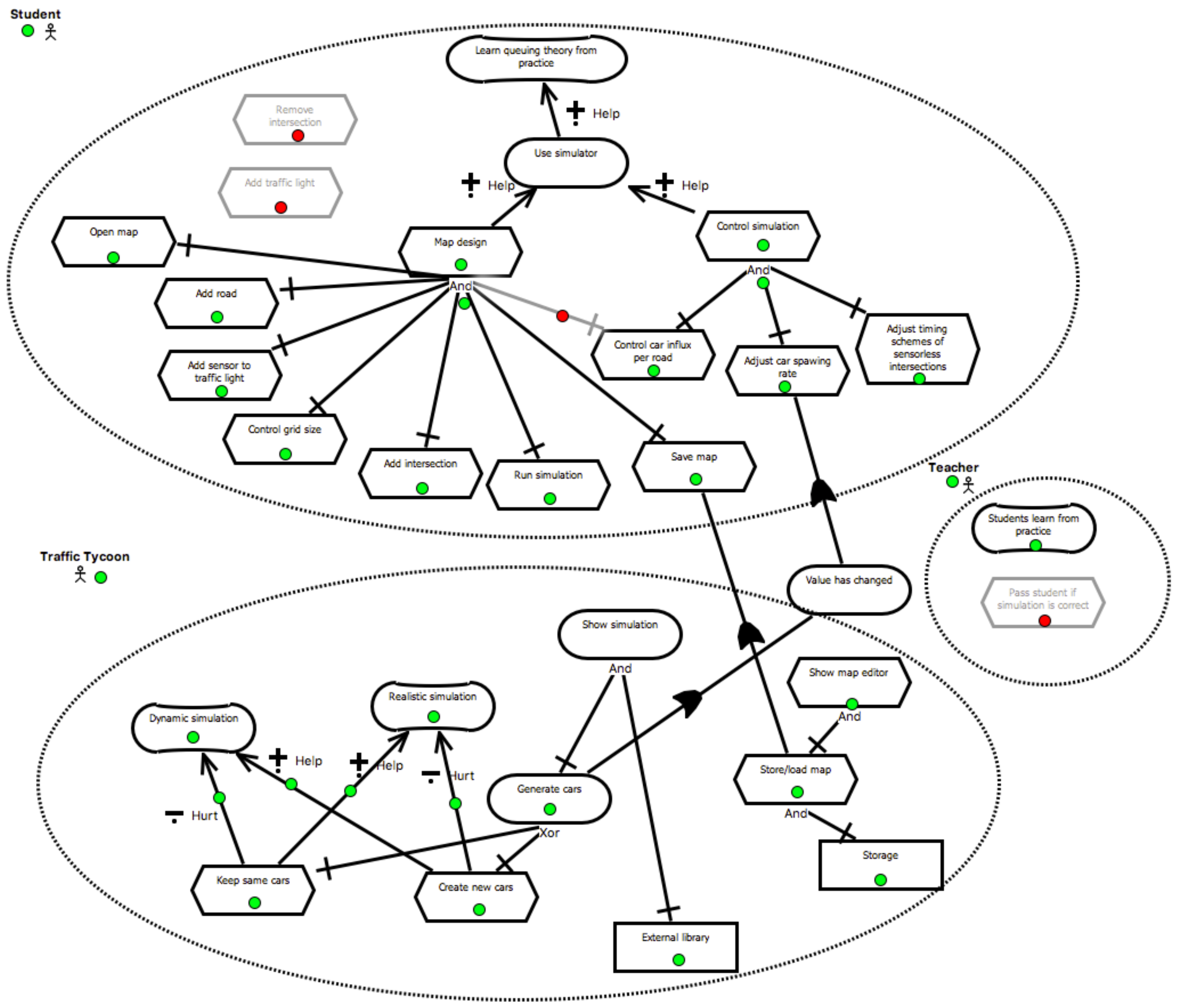


Fig. 6: The GRL model manually constructed from transcript t_1. Green dots indicate accepted underlying arguments, red dots indicate rejected underlying arguments. Elements and relationships with no dot have been inferred by us.

AS5 is instantiated as follows:: "Goal `Generate cars` XOR-decomposes into tasks `Keep same cars` and `Create new cars`" (see Figure 6). Suppose now the critical question CQ5b: "Does the goal `Generate cars` decompose into other tasks?" is answered with "Yes, namely `Choose randomly`". This results in a new instantiation of AS5, namely: "Goal `Generate cars` XOR-decomposes into tasks `Keep same cars`, `Create new cars`, and `Choose randomly`." As a result, the goal model will contain the corresponding task `Choose randomly`, as well as an OR-decomposition relation from the goal `Generate cars` to that task.

– DISABLE: Disable the element or relationship of argument scheme to which critical questions pertains. This operation does not create a new argument, but it only disables (i.e., attacks) the original one. For instance, suppose argument scheme AS0 is instantiated with: "Actor `Teacher` is relevant". This argument can be attacked with critical question CQ0: "Actor `Teacher` is not relevant". As a result, the argument related to the actor is attacked as well, and actor `Teacher` 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 with: "Actor `Student`" can per-

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.

form task `Choose a pattern preference`. This argument can be attacked with critical question CQ13: "The task `Choose a pattern preference` is unclear, it should be `Choose a road pattern`". This analysis results in replacing the original argument with the new argument "Actor `Student` can perform task `Choose a road pattern`." 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. Suppose argument scheme AS0 is instantiated with "Actor `Teacher` is relevant" and it is attacked by critical question CQ0: "Actor `Teacher` is not relevant", since the teacher does not create the system. However, if a new evidence comes up, this critical question may in return be attacked again. For instance, it may turn out that the teacher will work on developing the application herself. In this case, the generic counter argument "Actor `Teacher` is relevant, because he/she does the development of the application" attack the argument "Actor `Teacher` is not relevant".Thus, the original argument "Actor `Teacher` is relevant" is accepted again, and it is shown in the goal model.

In Section 5 we provide examples for all of these four effects.

5 Examples from the Transcripts

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 arguments, and the corresponding goal model elements. Figure 6 in the previous section shows an example of how a RationalGRL may look like when it is developed in our prototype tool. However, the argument schemes that are instantiated are left implicit in this figure (in the tool, they can be seen in the details pane, see Section 7). Since we would like to make these instantiated argument schemes explicit in the examples below, we use a different visual language here. We provide a legend for our visualization notation in Figure 7.

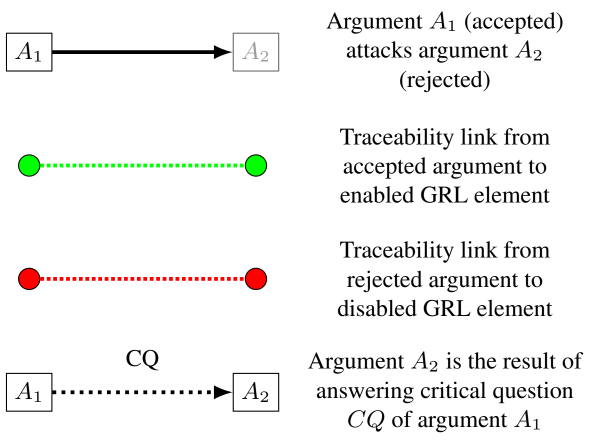

A_1
A_2
Argument A_1 (accepted) attacks argument A_2 (rejected)
Traceability link from accepted argument to enabled GRL element
Traceability link from rejected argument to disabled GRL element
CQ
A_1
A_2
Argument A_2 is the result of answering critical question CQ of argument A_1

Fig. 7: Legends of Various Elements and Relationships

Example 1: Methodology example

In this section, we present a small example for the RationalGRL methodology. This example is based on the traffic simulator example described in Section 2.1. We base argument schemes and critical questions of this example on a transcript containing discussions about the development of this traffic simulator system.

As mentioned in Section 2.1, a group of stakeholders are developing a goal model for a traffic simulator example and they are modeling the goal `Simulate Traffic` using the RationalGRL methodology. This proceeds in the following way:

– First they start with the initial GRL model (Figure 1) and they invoke the intentional element `Simulate Traffic`.
– Next, they switch to the argumentation part (step *Instantiate arguments schemes*). They answer critical question *Does Simulate Traffic AND-decompose into any tasks?* with *Yes: namely tasks "Dynamic Simulation" and "Static Simulation"*.
– As a result, two argument schemes are created, namely:
  - Actor `Traffic Simulator` has task `Dynamic Simulation`
  - Actor `Traffic Simulator` has task `Static Simulation`
– In the GRL model, this corresponds to the addition of two tasks, and an AND-decomposition from the goal *Simulate Traffic* to these two tasks.
– Next, the stakeholders test the validity of their goal model by answering more critical question. They answer two critical questions:
  - CQ *Is task "Dynamic Simulation" relevant* is answered with "No, it is not relevant since the problem specification explicitly states dynamic simulations are not required". Thus, the corresponding GRL intentional element is disabled.
  - The decomposition is changed from AND to OR, since it turned out not both tasks can be implemented together.

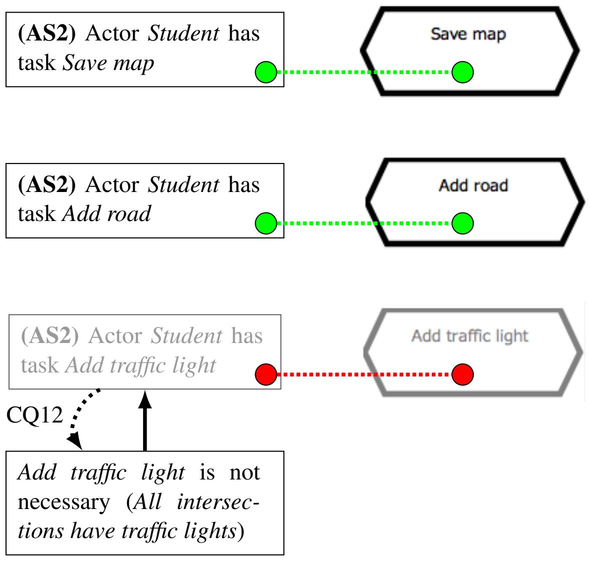
An excerpt of the RationalGRL model is shown in Figure 10.

Example 2: Disable task `Traffic light`

The transcript excerpt of this example is shown in Table 5 in the appendix and comes from transcript t_1. In this example, participants sum up functionality of the traffic simulator, which are tasks that the students can perform in the simulator. All these tasks can be formalized and are instantiated from argument scheme AS2: "Actor *Student* has tasks 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 necessary. We formalized this using the critical question CQ12: "Is task `Add traffic light` useful/relevant?".

We visualize some of the argument schemes, critical questions, and traceability links with GRL model in Figure 8. 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 attacked by another argument generated from applying critical question CQ12: "`Add traffic light` is not necessary (*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.



(AS2) Actor *Student* has task *Save map*
Save map
(AS2) Actor *Student* has task *Add road*
Add road
(AS2) Actor *Student* has task *Add traffic light*
Add traffic light
CQ12
Add traffic light is not necessary (*All intersections have traffic lights*)

Fig. 8: Argument Schemes and Critical Questions (left), GRL Model (right), and Traceability Link (dotted lines) for the Traffic Light Example.

Example 3: Clarify task `Road pattern`

The transcript excerpt of the second example is shown in Table 6 in Appendix B 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 9. 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

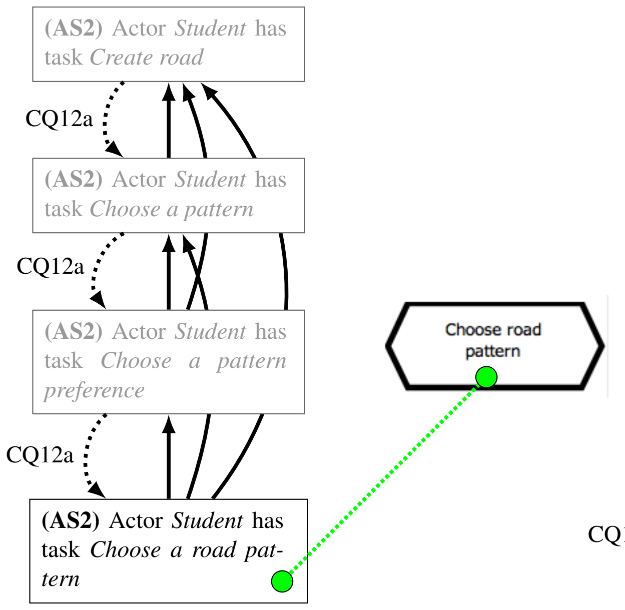

(AS2) Actor Student has task Create road
CQ12a
(AS2) Actor Student has task Choose a pattern
CQ12a
(AS2) Actor Student has task Choose a pattern preference
CQ12a
(AS2) Actor Student has task Choose a road pattern
Choose road pattern

Fig. 9: Argument Schemes and Critical Questions (left), GRL Model (right), and Traceability Link (dotted line) for the Road Pattern Example.

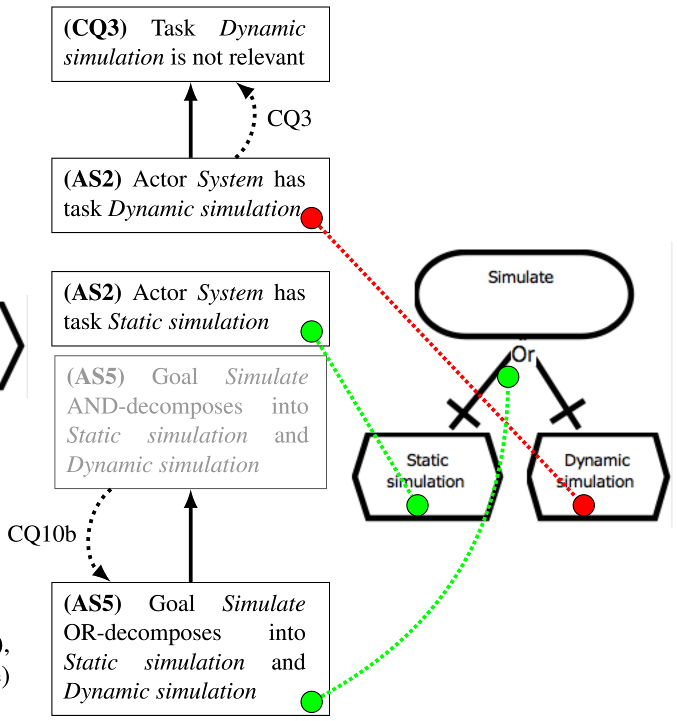

(CQ3) Task Dynamic simulation is not relevant
CQ3
(AS2) Actor System has task Dynamic simulation
(AS2) Actor System has task Static simulation
(AS5) Goal Simulate AND-decomposes into Static simulation and Dynamic simulation
CQ10b
(AS5) Goal Simulate OR-decomposes into Static simulation and Dynamic simulation
Simulate
Or
Static simulation
Dynamic simulation

Fig. 10: Argument Schemes and Critical Questions (left), GRL Model (right), and Traceability Link (dotted line) of the example.

final argument “Actor `Student` has task `Choose a road pattern`”. This final argument is ”un-attacked” 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 9 attacks all three other arguments for a name of the element. If this is not the case, then it may occur that a previous argument is *re-instantiated*, meaning that it becomes accepted again because the argument attacking it, is itself attacked. Suppose for instance that the bottom argument “Actor `Student` has task `Choose a pattern preference`” did not attack the second argument: “Action `Student` has task `Choose a pattern`”. In that case, this argument would be re-instantiated, because its only attacker “Actor `Student` has task `Choose a pattern preference`” is itself defeated by the bottom argument.

Example 4: Decompose goal `Simulate`

TODO for all(by Marc)**: Check whether the text matches the figure here** The transcript excerpt of this example is shown in Table 7 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 10. 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 5: Reinstate actor `Development team`

The transcript excerpt of this example is shown in Table 8 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` in the model. This is, then, questioned by participant `P2`, who argues that the professor will develop the software, so there will not be any development team. However, in the second part, participant `P2` argue that the development team should be considered, since the professor does not develop the software.

We formalize this using a *generic counterargument*, attacking the critical question. The first part of the discussion is shown in Figure 11. We formalize the first statement as an instantiation of argument scheme AS0: “Actor `development team` is relevant”. This argu-

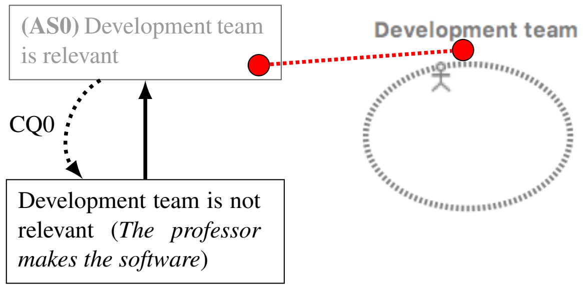

(AS0) Development team is relevant
Development team
CQ0
Development team is not relevant (The professor makes the software)

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.

ment is, 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 11, left image.

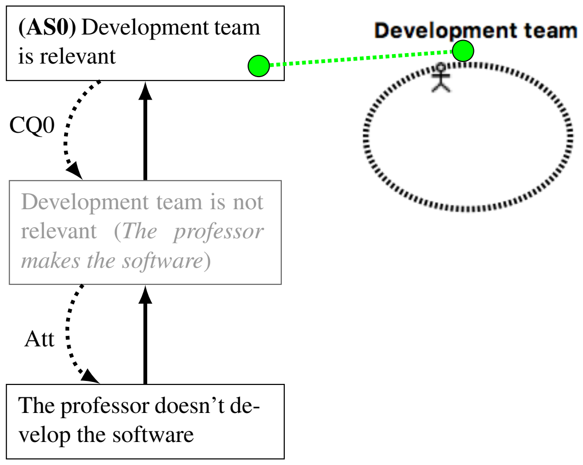

(AS0) Development team is relevant
Development team
CQ0
Development team is not relevant (The professor makes the software)
Att
The professor doesn't develop the software

Fig. 12: Argument Schemes and Critical Questions (left), GRL Model (right), and Traceability Link (dotted line) of a Discussion about the Relevance of Actor Development Team.

Figure 12 shows the situation after the counter argument has been put forward. The argument “Actor `Professor` does not 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 argument is reinstantiated, which causes the actor in the GRL model to be enabled again.

6 RationalGRL: Logical Framework

In Section 4 we developed a list of critical questions and argument schemes by analyzing transcripts of discussions about the development of a traffic simulator. The resulting list is shown in Table 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 provide some insight on how our RationalGRL framework can be used. However, since we would like to implement our framework in a tool, we require a more precise formalization. In this section, we present a formalization of our framework using formal logic. This is the main approach using in formal argumentation, so it allows us to use many of the techniques developed in that area directly (see Section 8.1 for more details).

In the first two subsections we formalize a static representation of our framework: We develop a formal language to specify a GRL model (independently of arguments) in the fist subsection, and we formalize the notion of an argument in the second subsection. In the third subsection we formalize the dynamics of our framwork: we develop algorithms for instantiating argument schemes and for answering critical questions. The formal framework we develop in this section serves as the underlying model for the prototype implementation we present in the next section.

6.1 Logical Language for RationalGRL

We assume familiarity with basic notions from propositional logic and set theory. A *propositional atom* is a grounded formula (a formula without variables) with no logical connectives $(\vee, \wedge, \rightarrow, \leftrightarrow)$ or negation $(\neg)$. We use integers $i, j, k \in \mathbb{N}$ as identifiers for actors and GRL elements.

Definition 1 (GRL Atoms). *The* set of GRL atoms At *contains the following atoms:*

- $actor(i)$*: Identifier* i *is an actor.*
- $softgoal(i)$*: Identifier* i *is a softgoal.*
- $goal(i)$*: Identifier* i *is a goal.*
- $task(i)$*: Identifier* i *is a task.*
- $resource(i)$*: Identifier* i *is a resource.*

- *$decomp(k, i, J, dtype)$: Identifier k is a dtype-decomposition (and/or/xor) from the element corresponding to identifier i to the set of elements corresponding to identifiers J.*
- *$dep(k, i, j)$: Identifier k is a dependency link from the element corresponding to identifier i to the element corresponding to identifier j.*
- *$contr(k, i, j, ctype)$: Identifier k is a ctype-contribution (+/-) from the element corresponding to identifier i to the element corresponding to identifier j.*
- *$has(i, j)$: Identifier i (which is an actor) has the element corresponding to identifier j.*
- *$disabled(i)$: The element or relationships corresponding to identifier i is disabled.*
- *$name(i, p)$: The name of the element corresponding to identifier i is p.*

For all of the above definitions we assume that all identifiers used in each atom are different, i.e.,

- $k \neq i$,$k \neq i$,$k \neq j$
- *forall* $j \in J : k \neq j, i \neq j$

An example of the specification of the GRL model in Figure 10 is shown in Table 4. The specification has been written in logic programming style. In this article we do not make use of any logic programming techniques but this would be interesting future work (see Section 8.2). A more elaborate example of a specification is shown in Appendix C, showing a complete specification of the traffic simulator GRL model in Figure **??**.

```
goal(0).
task(1).
task(2).
name(0,simulate).
name(1,static_simulation).
name(2,dynamic_simulation).
decomp(3,0,[1,2],or).
```
Table 4: Specification of the GRL model in Figure 10

6.2 Formal argumentation semantics

In the previous subsection we introduced an atomic language to specify a GRL model. In this subsection we give a formal definition of an argument, which is simply a set of atoms from our language.

Definition 2 (Argument). *An* argument $A \subseteq Att$ *is a set of atoms from Att.*

This simple definition allows us to form arguments about (parts of) a GRL model. For instance,

$$\{goal(0), name(0, development\_team)\},$$

is an argument stating that identifier 0 is a goal element with name "development team".

Note that this definition does not put any restrictions on the content of an argument. This means it is possible to form arguments that do not correspond to correct or sensible GRL models. Consider for instance the following argument:

$$\{goal(0), task(0), name(0, static\_simulation)\}.$$

This argument states that identifier 0 is both a goal and a task with name "statis simulation", which is clearly undesirable since it does not correspond to a valid GRL model. While we could have chosen to put logical constraints on the content of an argument, we choose not to do so here. Instead, we put these restrictions on the algorithms in section **??**. The algorithms are constructed in such a way that they only allow the construction of argumetns that correspond to valid GRL models.[5] The advantage of this approach is that it keeps the presentation of the formal part relatively simple.

In line with the main approaches in formal argumentation, we next introduce an argumentation framework as a set of arguments and an attack relation between the arguments.

Definition 3 (Argumentation framework [14]). *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 $A_2 \in Args$.*

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

Definition 4 (Attack, conflict-freeness, defense, and admissibility [14]). *Suppose an argumentation framework $AF = (Arg, Att)$, two sets of arguments $S \cup S' \subseteq Arg$, and some argument $A \in Arg$. We say that*

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

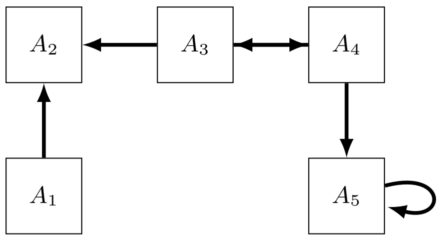

$$\begin{tikzcd} A_2 & A_3 \arrow[l] \arrow[r, leftrightarrow] & A_4 \arrow[d] \\ A_1 \arrow[u] & & A_5 \arrow[loop right] \end{tikzcd}$$

Fig. 13: Example argumentation framework.

Let us explain these definitions using the example argumentation framework in Figure 13. We say that the set $\{A_1, A_4, A_5\}$ *attacks* argument A_2, because A_1 attacks A_2. However, $\{A_1, A_4, A_5\}$ is not *conflict-free*, because A_4 attacks A_5, so this set of arguments attacks itself. If we remove A_5 from this set, then it is conflict-free. In total, all the following sets are conflict-free in Figure 13:

$$\{A_1, A_3\}, \{A_1, A_4\}, \{A_2, A_4\}, \{A_1\}, \{A_2\}, \{A_3\}, \{A_4\}, \emptyset.$$

However, not all of these sets are *admissible*. For instance, the set $\{A_2, A_4\}$ is not admissible, because A_1 attacks this set, but the set does not defend itself against this attack. The set $\{A_1, A_3\}$ is admissible, because its only attacker, A_4 is attacked by A_3.

There are a large number of different semantics using these notions to determine which arguments are acceptable, such as the *grounded*, the *preferred*, the *stable*, and the *complete* semantics. However, in this article our argumentation frameworks are very simple, in the sense that they do not introduce cycles. In future work, we aim to extend this by using preferences (see open issues in Section 8.2). An advantage of our current approach is that all the standard semantics coincide when there are no cycles, which simplifies our case. We use the preferred semantics here, but we could equivalently have chosen any other version.

Definition 5 (Preferred semantics [14]). *Suppose an argumentation framework $AF = (Arg, Att)$. A set of arguments $S \subseteq Arg$ is a* preferred extension *if and only if*

- *S is an admissible set of argument,*
- *For each admissible set of arguments $T \subseteq A$: $S \not\subset T$.*

In our example in Figure 13, the preferred extensions are $\{A_1, A_3\}$ and $\{A_1, A_4\}$.

[5] Note that we do not provide formal proofs for these claims in this paper. This is an interesting direction of future work, but it is not the focus of this article.

6.3 Algorithms for argument schemes and critical questions

sect:algorithms]

Let us briefly take stock:

- We developed an initial set of argument schemes and critical questions in Section 4, by annotating transcripts.
- We developed a visual notation to illustrate some argument schemes and critical questions in Section 5.
- We proposed a logical language consisting of atomic sentence to describe a GRL model in Section 6.1.
- We formalized an "argument" as a set of atoms from our language, and we introduced *argumentation semantics* to compute sets of accepted arguments in Section 6.2.

In this subsection we develop algorithms for applying argument schemes and critical questions. The process of applying these algorithms and generating goal models from them is shown in context of the RationalGRL methodology in Figure **??** **TODO for Marc(by Marc): Create an extended version of the methodology, containing where which parts of the formal framework are used.** These algorithms produce arguments (Definition 2) and attack relations, forming an argumentation framework (Definition 3). One can then use argumentation semantics (Definition 5) to compute sets of accepted arguments. The content of these arguments is then used to compute the resulting RationalGRL model, together with enabled and disabled GRL elements and their underlying arguments.

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

Definition 6 (Global variables). *The following variables are intended to have a* global *scope and have a* static *value, meaning the lifetime of the variable extends across the entire run of the program.*

- *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.*

Algorithms for argument schemes

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

Algortihm 1 for argument scheme AS0: The algorithm takes one argument, namely the name of the actor a. On line 2 of the algorithm, the global variable id is increased by one. This ensures that each new argument has

Algorithm 1 Applying AS0: Actor a is relevant

```
1: procedure $AS_0(a)$
2:     $id \leftarrow id + 1$
3:     $A \leftarrow \{actor(id), name(id, a)\}$
4:     $Args \leftarrow Args \cup A$
5: end procedure
```

a unique identifier. On line 3, the argument for the actor is formed, consisting of two statements, stating respectively that id is an actor, and that the name of this actor is a. Finally, on line 4 this argument is added to the global set of arguments $Args$. In Figure 12, the application of the argument scheme AS_0(Development team), results in one argument:

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

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

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

Algorithm 2 for argument scheme AS1: This argument scheme takes two arguments, the identifier a_{id} of the actor and the resource name n. The algorithm is similar to the previous one, with the difference that the newly added argument contains the statement $has(a_{id}, id)$ as well, meaning that the actor with id a_{id} has element id (which is a resource). As an example, let us formalize the argument corresponding to resource *External library* of actor *Traffic tycoon* in Figure 6. First, we assume some id is associated with the actor:

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

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

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

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

Algorithm 3 for argument scheme AS5: 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

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

```
1: procedure $AS_5(g_{id}, \{T_1, \ldots, T_n\}, type)$
2:     $T_{id} = \emptyset$
3:     for $T_i$ in $\{T_1, \ldots, T_n\}$ 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:    $A \leftarrow \{decomp(id, g_{id}, T_{id}, type)\}$
15:    $Args \leftarrow Args \cup A$
16: end procedure
```

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

Let us explain this algorithm with the XOR-decomposition of goal `Generate cars` of Figure 6. Suppose the following arguments are constructed already:

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

Suppose furthermore that someone wants to put forward the argument that goal `Generate cars` XOR-decomposes into tasks `Keep same cars` and `Create news cars`. Formally: $AS_5(0, \{generate\_cars, keep\_same\_cars\}, xor)$.
The algorithm will first set $T_{id} = \emptyset$, and then iterate over the two task names. For the first task $generate\_cars$, there does not exist an argument

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

$\{task(t_{id}), name(t_{id}, generate\_cars)\}$ yet, so a new argument is created. Suppose the following argument is created: $\{task(2), name(2, generate\_cars)\}$. After this, 2 is added to T_{id}. For the second task an argument exists already, namely $\{task(1), name(1, keep\_same\_cars)\}$, so 1 is simply added to T_{id}. After the for loop, we have $T_{id} = \{1, 2\}$. Next, the decomposition argument is created, which is $\{decomp(3, 0, \{1, 2\}, xor)\}$. This argument is added to $Args$ and the algorithm terminates.

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

```
 1: procedure $AS_6(t_{id}, s)$
 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
```

Algorithm 4 for argument scheme AS6: 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 idea behind this algorithm is very similar to the previous one, with the difference that in the current algorithm we create a single relation, while we created a set of relations in the previous algorithm. First, the if-statement on line 2 checks whether the softgoal exists already, and if not, an argument is added for it. This ensures that all softgoals have corresponding arguments. After the if-statement, the argument for the contribution link is created and it is added to the set of arguments $Args$.

Let us again illustrate this with a simple example from Figure 6. Suppose the following argument exists already: $\{task(0), name(0, keep\_same\_cars\}$, and suppose someone would like to add an argument that the task `Keep same cars` contributes positively to softgoal `Dynamic simulation`, i.e. $AS_6(0, dynamic\_simulation)$. The algorithm first checks whether an argument already exists for the softgoal, and when it finds out it does not exist, creates the argument $\{softgoal(1), name(1, dynamic\_simulation)\}$ and adds it to $Args$. Then, the argument for the contribution is added to $Args$ as well, which is $\{contr(2, 0, 1, pos)\}$.

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

Algorithms for critical questions

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

Algorithm 5 Applying DISABLE: Element i is disabled

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

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

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

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

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

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

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

The algorithm takes as input the goal identifier g_{id}, the set of existing decomposing task identifiers $i_1, \ldots, i_n$, the decomposition type, and the names of the new tasks $t_1, \ldots, t_k$ that should be added to the decomposition.

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

```
1: procedure CQ5B($g_{id}, \{i_1, \ldots, i_n\}, type, \{t_1, \ldots, t_k\}$)
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 $\{decomp(\_, g_{id}, \_, \_)\} \subseteq A \mid A \in Args\}$ do
16:        $Att \leftarrow Att \cup \{(A_{new}, A)\}$
17:    end for
18:    $Args \leftarrow Args \cup \{A_{new}\}$
19: end procedure
```

The first part of the algorithm is familiar from Algorithm 3: For each task name we check whether it already exists as an argument (line 4), and if it doesn't (line 6) we add a new argument for it. After the for-loop (line 13), a new argument is created for the new decomposition relation (14). After this, the for-loop on line 15 ensures that the new argument attacks all previous arguments for this decomposition (note that the variable "_" means "do not care"). Only at the very end the new argument is added (line 18), to ensure it does not attack itself after the for loop of line 15-17.

An example of this algorithm is shown in Figure 14.[7] Before the critical question is applied, the following arguments have been put forward:

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

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

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

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

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

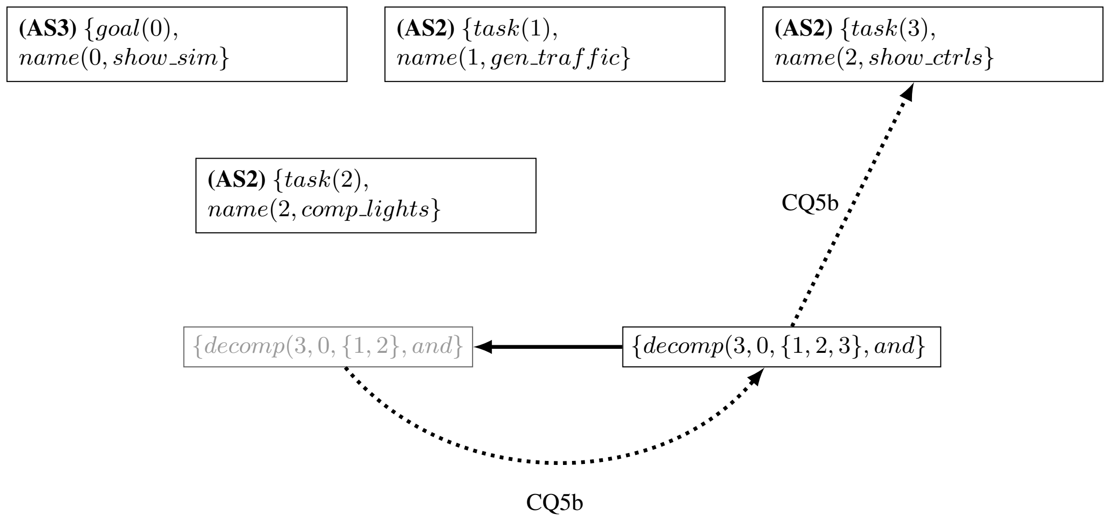
```
1: procedure CQ13($i, n$)
2:     $ArgsN \leftarrow \{A \in Args \mid name(i, x) \in A\}$
3:     $B \leftarrow B' \backslash \{name(i, \_)\}$ with $B' \in ArgsN$
4:     $A \leftarrow B \cup \{name(i, n)\}$
5:     $Args \leftarrow Args \cup \{A\}$
6:     for $C$ in $ArgsN$ do
7:         $Att \leftarrow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

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

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

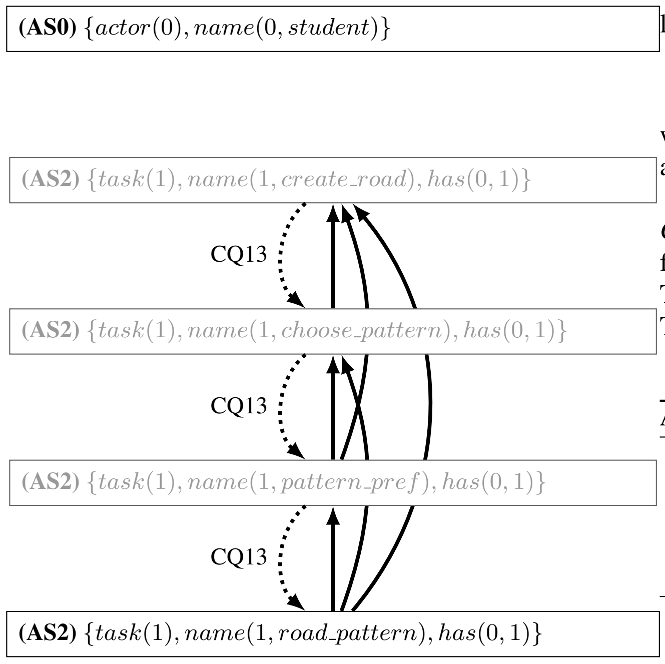
An example of of Algorithm 7 is shown in Figure 15. Let us consider the last application of CQ13 (bottom argument). Before this application, the following arguments have been put forward:

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



(AS3) $\{goal(0), name(0, show\_sim\}$
(AS2) $\{task(1), name(1, gen\_traffic\}$
(AS2) $\{task(3), name(2, show\_ctrls\}$
(AS2) $\{task(2), name(2, comp\_lights\}$
CQ5b
$\{decomp(3, 0, \{1, 2\}, and\}$
$\{decomp(3, 0, \{1, 2, 3\}, and\}$
CQ5b

Fig. 14: Example of applying critical question CQ5b (Algorithm 6)



(AS0) $\{actor(0), name(0, student)\}$
(AS2) $\{task(1), name(1, create\_road), has(0, 1)\}$
CQ13
(AS2) $\{task(1), name(1, choose\_pattern), has(0, 1)\}$
CQ13
(AS2) $\{task(1), name(1, pattern\_pref), has(0, 1)\}$
CQ13
(AS2) $\{task(1), name(1, road\_pattern), has(0, 1)\}$

Fig. 15: Applying critical question CQ13 (Algorithm 7) to the example in Figure 9.

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

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

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

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

Algorithm 8 Generic counterargument to argument A

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

Algorithm for Att (Generic counter argument: Applying a generic counter argument is very simple, and simply results on an attack on the original argument. We illustrate this by continuing our example from Figure 16 (Algorithm 1). The example is shown in Figure 16, where we see that a generic counter argument simply attacks the argument to disable the actor.

6.4 Constructing GRL models

Constructing GRL models from the arguments is extremely simple: We simply compute the extensions of the argumentation frameworks, and collect all atomic

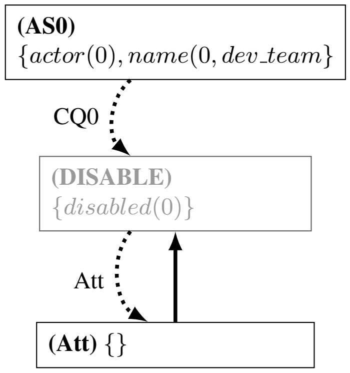


Fig. 16: Formalization of the arguments in Figure 12.

sentences in the accepted arguments. This forms out GRL model. Let us briefly do so for the examples of the previous subsection:

- Figure **??**: Since there are no attacks between the arguments, all atomic sentences are accepted. This results in the following specification:
  ```
  actor(0).
  name(0,dev_team).
  disabled(0).
  ```
  This again corresponds to the GRL model on the right-hand side of Figure 11.
- Figure 14: There is one rejected argument and five accepted ones. The resulting specification is:
  ```
  goal(0). name(0,show_simulation).
  task(1). name(1,generate_traffic).
  task(2). name(2,compute_lights).
  task(3). name(3,show_controls).
  decomp(3,0,{1,2,3},and).
  ```
- Figure 15: There are only two accepted arguments. The resulting specification is:
  ```
  actor(0). name(0, student).
  task(1).  name(1,road_pattern).
  has(0,1).
  ```
  This corresponds to the right-hand GRL model of Figure 9.
- Figure 16. There are two accepted arguments, but the *generic counterargument* does not contain any formulas. Therefore the resulting specification is:
  ```
  actor(0).
  name(0,dev_team).
  ```
  This corresponds to the right-hand GRL model of Figure 12.

7 The RationalGRL Tool

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

GRL has a well-documented and well-maintained tool called jUCMNav [31]. This tool is an extension to Eclipse. Although it is a rich tool with many features, we also believe it is not very easy to set it up. This seriously harms the exposure of the language, as well as the ability for practitioners to use it. We have started to implement a simple version of GRL as an online tool in Javascript. This makes it usable from the browser, without requiring the installation of any tool. The tool can be used from the following address:

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

A screenshot of the tool is shown in Figure 17. As shown, there are two tabs in the tool, one for “Argument” and one for “GRL”. The argument part has not been implemented yet, and the GRL part only partly, but the idea behind the tool should be clear. Users are able to work on argumentation and on goal modeling in parallel, where the argumentation consists of forming arguments and counterarguments by instantiating argument schemes and answering critical questions.

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

8 Discussion

8.1 Related work

The need for justifications of modeling choices plays an important role in different requirements engineering methods using goal models. High-level goals are often understood as reasons for representing lower-level goals (in other words, the need for low-level goals is justified by having high-level goals) and other elements in a goal model such as tasks and resources. Various refinements and decomposition techniques, often used in requirements engineer (See [43] for an overview), can be seen as incorporating argumentation and justification, in that sub-goals could be understood as arguments supporting

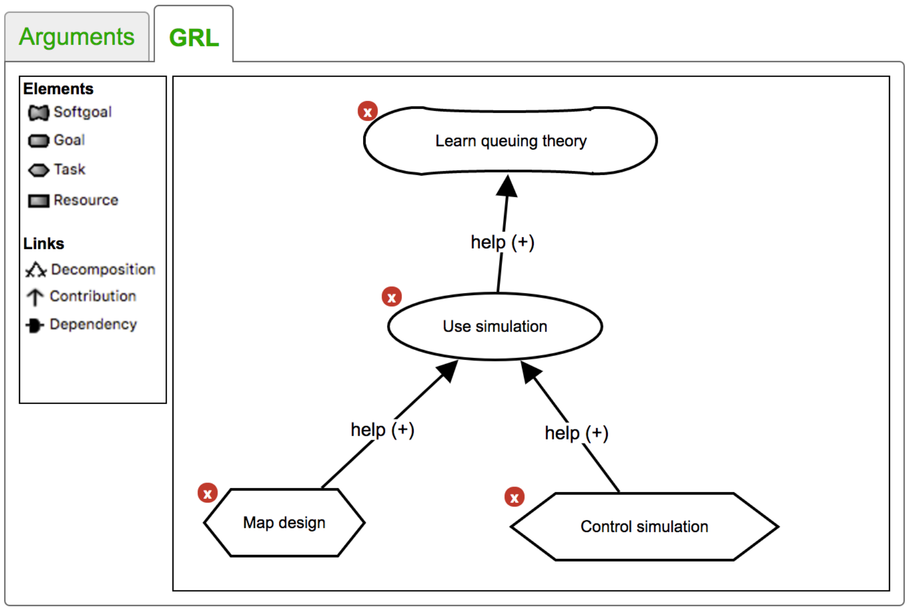


Fig. 17: Screenshot of the prototype tool

parent goals. In that case, a refinement alternative is justified if there are no conflicts between sub-goals (i.e., it is consistent), as few obstacles as possible sub-goals harm sub-goal achievement, there are no superfluous sub-goals (the refinement is minimal), and the achievement of sub-goals can be verified to lead to achieving the parent goal (if refinement is formal [12]). This interpretation is one of the founding ideas of goal modeling. However, while this interpretation may seem satisfactory, argumentation and justification processes differ from and are complementary to refinement in several respects, such as limited possibilities for rationalization and lack of semantics (see Jureta [20] for more details).

There are several contributions that relate argumentation-based techniques with goal modeling. The contribution most closely related to ours is the work by Jureta *et al.* [20]. Jureta *et al.* propose "Goal Argumentation Method (GAM)" to guide argumentation and justification of modeling choices during the construction of goal models. One of the elements of GAM is the translation of formal argument models to goal models (similar to ours). In this sense, our RationalGRL framework can be seen as an instantiation and implementation of part of the GAM. The main difference between our approach and GAM is that we integrate arguments and goal models using argument schemes, and that we develop these argument schemes by analyzing transcripts. GAM instead uses structured argumentation.

The RationalGRL framework is also closely related to frameworks that aim to provide a design rationale (DR) [37], 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.* [17], 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 [20], 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.* [30] propose Arg-ACH, an approach to capture inconsistencies between stakeholders' beliefs and goals, and resolve goal conflicts using argumentation techniques.

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

- **RE17 paper: merging argumentation theory with law "Using Argumentation to Explain Ambiguity in Requirements Elicitation Interviews" Yehia**

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

8.2 Open issues

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

Architecture principles

One aspect of enterprise architecture that we did not touch upon in this article are *(enterprise) architecture principles*. Architecture principles are general rules and guidelines, intended to be enduring and seldom amended, that inform and support the way in which an organization sets about fulfilling its mission [23, 32, 38]. They reflect a level of consensus among the various elements of the enterprise, and form the basis for making future IT decisions. Two characteristics of architecture principles are:

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

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

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

In a previous paper, we [26] propose an extension to GRL based on enterprise architecture principles. We present a set of requirements for improving the clarity of definitions and develop a framework to formalize architecture principles in GRL. We introduce an extension of the language with the required constructs and establish modeling rules and constraints. This allows one to automatically reason about the soundness, completeness and consistency of a set of architecture principles. Moreover, principles can be traced back to high-level goals.

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

Extensions for argumentation

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

For instance, until now we have assumed that every argument put forward by a critical questions always defeats the argument it questions, but this is a rather strong assumption. In some cases, it is more difficult to determine whether or not an argument is defeated. Take, for example, the argumentation framework in Figure 18 with just A1 and A2. These two arguments attack each other, they are alternatives and without any explicit preference, and it is impossible to choose between the two. It is, however, possible to include explicit preferences between arguments when determining argument acceptability [1]. If we say that we prefer the action `Create new cars` (A2) over the action `Keep same cars` (A1), we remove the attack from A1 to A2. This makes A2 the only undefeated argument, whereas A1 is now defeated. It is also possible to give explicit arguments for preferences [28]. These arguments are then essentially attacks on attacks. For example, say we prefer A3 over A1 because 'it is important to have realistic traffic flows' (A4). This can be rendered as a separate argument that attacks the attack from A1 to A3, removing this attack and making {A3, A4} the undefeated set of arguments.

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

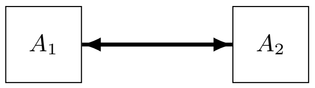


Fig. 18: Preferences between arguments

Finally, in this article we have only explored one single argument scheme, but there are many other around. In his famous book "Argumentation schemes", Walton describes a total of 96 schemes. Murukannaiah *et al.* [29] already explain how some of these schemes may be use for resolving goal conflicts, and it is worth studying what this would look like in our framework as well.

Empirical study

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

Formal framework

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

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

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

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

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

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

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

8.3 Conclusion

TODO for all(by Marc)**: Finish this**

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

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

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

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

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

Acknowledgments

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

References

1. L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3):197–215, 2002.
2. D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. S. K. Yu. Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 25:841–877, August 2010.
3. K. Atkinson and T. Bench-Capon. Legal case-based reasoning as practical reasoning. *Artificial Intelligence and Law*, 13(1):93–131, 2005.
4. K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171(10):855–874, 2007.
5. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.
6. M. E. Bratman. *Intention, plans, and practical reason*. Harvard University Press, Cambridge, MA, 1987.
7. D. Cartwright and K. Atkinson. Using computational argumentation to support e-participation. *IEEE Intelligent Systems*, 24(5):42–52, 2009.
8. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.
9. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.
10. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.
11. A. Dardenne, A. Van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.
12. R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 179–190, New York, NY, USA, 1996. ACM.
13. P. Donzelli. A goal-driven and agent-based requirements engineering framework. *Requirements Engineering*, 9(1):16–39, 2004.
14. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.
15. S. Ghanavati. *Legal-urn framework for legal compliance of business processes*. PhD thesis, University of Ottawa, 2013.
16. P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.
17. C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh. Arguing security: Validating security requirements using structured argumentation. In *in Proc. of the Third Symposium on RE for Information Security (SREIS'05)*, 2005.
18. J. Horkoff, D. Barone, L. Jiang, E. Yu, D. Amyot, A. Borgida, and J. Mylopoulos. Strategic business modeling: representation and reasoning. *Software & Systems Modeling*, 13(3):1015–1041, 2014.
19. ITU-T. Recommendation Z.151 (11/08): User Requirements Notation (URN) – Language Definition. `http://www.itu.int/rec/T-REC-Z.151/en`, 2008.

20. I. Jureta, S. Faulkner, and P. Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Requirements Engineering*, 13(2):87–115, May 2008.
21. E. Kavakli and P. Loucopoulos. Goal modeling in requirements engineering: Analysis and critique of current methods. In J. Krogstie, T. A. Halpin, and K. Siau, editors, *Information Modeling Methods and Methodologies*, pages 102–124. Idea Group, 2005.
22. S. D. kenniscentrum. DEMO: The KLM case. `http://www.demo.nl/attachments/article/21/080610_Klantcase_KLM.pdf`, Last accessed on 22 August, 2012.
23. M. Lankhorst. *Enterprise architecture at work: modelling, communication, and analysis*. Springer, 2005.
24. L. Liu and E. Yu. Designing information systems in social context: a goal and scenario modelling approach. *Information systems*, 29(2):187–203, 2004.
25. J. G. March. Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, pages 587–608, 1978.
26. D. Marosin, M. van Zee, and S. Ghanavati. Formalizing and modeling enterprise architecture (ea) principles with goal-oriented requirements language (grl). In *Proceedings of the 28th International Conference on Advanced Information System Engineering (CAiSE16)*, June 2016.
27. R. Medellin-Gasque, K. Atkinson, T. Bench-Capon, and P. McBurney. Strategies for question selection in argumentative dialogues about plans. *Argument & Computation*, 4(2):151–179, 2013.
28. S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9):901–934, 2009.
29. P. K. Murukannaiah, A. K. Kalia, P. R. Telangy, and M. P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 156–165. IEEE, 2015.
30. P. K. Murukannaiah, A. K. Kalia, P. R. Telangy, and M. P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *23rd Int. Requirements Engineering Conf.*, pages 156–165. IEEE, 2015.
31. G. Mussbacher and D. Amyot. Goal and scenario modeling, analysis, and transformation with jUCMNav. In *ICSE Companion*, pages 431–432, 2009.
32. M. Op 't Land and E. Proper. Impact of principles on enterprise engineering. In H. sterle, J. Schelp, and R. Winter, editors, *ECIS*, pages 1965–1976. University of St. Gallen, 2007.
33. M. Petre and A. V. D. Hoek. *Software Designers in Action: A Human-Centric Look at Design Work*. Chapman & Hall/CRC, 1st edition, 2013.
34. J. Raz. *Practical Reasoning*. Oxford University Press, 1978.
35. Rizkiyanto. Better Design Rationale to Improve Software Design Quality. Master's thesis, Utrecht University, the Netherlands, 2016.
36. C. Schriek. How a Simple Card Game Influences Design Reasoning: a Reflective Method. Master's thesis, Utrecht University, the Netherlands, 2016.
37. S. J. B. Shum, A. M. Selvin, M. Sierhuis, J. Conklin, C. B. Haley, and B. Nuseibeh. Hypermedia support for argumentation-based rationale. In *Rationale management in software engineering*, pages 111–132. Springer, 2006.
38. The Open Group. *The Open Group – TOGAF Version 9*. Van Haren Publishing, Zaltbommel, The Netherlands, 2009.
39. P. Tolchinsky, S. Modgil, K. Atkinson, P. McBurney, and U. Cortés. Deliberation dialogues for reasoning about safety critical actions. *Autonomous Agents and Multi-Agent Systems*, 25(2):209–259, 2012.
40. UCI. Design Prompt: Traffic Signal Simulator. `http://www.ics.uci.edu/design-workshop/files/UCI_Design_Workshop_Prompt.pdf`. Accessed: 2016-12-27.
41. F. H. van Eemeren, B. Garssen, C. Krabbe, A. F. Snoeck Henkemans, B. Verheij, and J. H. Wagemans. *Handbook of Argumentation Theory. A Comprehensive Overview of the State of the Art*. Springer, Berlin, 2014.
42. A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. 5th IEEE Int. Symposium on RE*, pages 249–262, 2001.
43. A. Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. RE'01: 5th Intl. Symp. Req. Eng.*, pages 249–262, 2001.
44. A. van Lamsweerde. Requirements engineering: from craft to discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 238–249. ACM, 2008.
45. M. van Zee, F. Bex, and S. Ghanavati. Rationalization of Goal Models in GRL using Formal Argumentation. In *Proceedings of RE: Next! track at the Requirements Engineering Conference 2015 (RE'15)*, August 2015.
46. M. van Zee and S. Ghanavati. Capturing Evidence and Rationales with Requirements Engineering and Argumentation-Based Techniques. In *Proc. of the 26th Benelux Conf. on Artificial Intelligence (BNAIC2014)*, November 2014.
47. M. van Zee, D. Marosin, F. Bex, and S. Ghanavati. The rationalgrl toolset for goal models and argument diagrams. In *Proceedings of the 6th International Conference on Computational Models of Argument (COMMA'16), Demo abstract*, September 2016.
48. M. van Zee, D. Marosin, S. Ghanavati, and F. Bex. Rationalgrl: A framework for rationalizing goal models using argument diagrams. In *Proceedings of the 35th International Conference on Conceptual Modeling (ER'2016), Short paper*, November 2016.
49. D. Walton, C. Reed, and F. Macagno. *Argumentation schemes*. Cambridge University Press, 2008.
50. D. N. Walton. *Practical reasoning: goal-driven, knowledge-based, action-guiding argumentation*, volume 2. Rowman & Littlefield, 1990.
51. E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proc. of the 3rd IEEE Int. Symposium on RE*, pages 226–235, 1997.
52. E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements*

Engineering, RE ’97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society.

A UCI Design Workshop Prompt

Design Prompt: Traffic Signal Simulator

Problem Description

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

Requirements

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

1. Students must be able to create a visual map of an area, laying out roads in a pattern of their choosing. The resulting map need not be complex, but should allow for roads of varying length to be placed, and different arrangements of intersections to be created. Your approach should readily accommodate at least six intersections, if not more.
2. Students must be able to describe the behavior of the traffic lights at each of the intersections. It is up to you to determine what the exact interaction will be, but a variety of sequences and timing schemes should be allowed. Your approach should also be able to accommodate left-hand turns protected by left-hand green arrow lights. In addition:
   (a) Combinations of individual signals that would result in crashes should not be allowed.
   (b) Every intersection on the map must have traffic lights (there are not any stop signs, over- passes, or other variations). All intersections will be 4-way: there are no "T" intersections, nor one-way roads.
   (c) Students must be able to design each intersection with or without the option to have sensors that detect whether any cars are present in a given lane. The intersection's lights' behavior should be able to change based on the input from these sensors, though the exact behavior of this feature is up to you.
3. Based on the map created, and the intersection timing schemes, the students must be able to simulate traffic flows on the map. The traffic levels should be conveyed visually to the user in a real-time manner, as they emerge in the simulation. The current state of the intersections' traffic lights should also be depicted visually, and updated when they change. It is up to you how to present this information to the students using your program. For example, you may choose to depict individual cars, or to use a more abstract representation.
4. Students should be able to change the traffic density that enters the map on a given road. For ex- ample, it should be possible to create a busy road, or a seldom used one, and any variation in between. How exactly this is de- clared 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. Stu- dents 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.

Your design will primarily be evaluated based on its elegance and clarity both in its overall solution and envisioned implementation structure.

Desired Outcomes

Your work on this design should focus on two main issues:

1. You must design the interaction that the students will have with the system. You should design the basic appearance of the program, as well as the means by which the user creates a map, sets traffic timing schemes, and views traffic simulations.
2. You must design the basic structure of the code that will be used to implement this system. You should focus on the important design decisions that form the foundation of the implementation, and work those out to the depth you believe is needed.

The result of this session should be: *the ability to present your design to a team of software developers who will be tasked with actually implementing it.* The level of competency you can expect is that of students who just completed a basic computer science or software engineering undergraduate degree. You do not need to create a complete, final diagram to be handed off to an implementation team. But you should have an understanding that is sufficient to explain how to implement the system to competent developers, without requiring them to make many high-level design decisions on their own.

To simulate this hand-off, you will be asked to briefly explain the above two aspects of your design after the design session is over.

Timeline

- 1 hour and 50 minutes: Design session
- 10 minutes: Break / collect thoughts
- 10 minutes: Explanation of your design
- 10 minutes: Exit questionnaire

B Transcripts excerpts

C 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).
```

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? **[22 answer to 22]** Not useful, because according to the specification all intersections have traffic lights.
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	
0:15:51.4 (P2)	They need-	

Table 5: 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. **[32a REPLACE]** “Create road” becomes “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	
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 6: 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 7: 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 8: Discussion about the relevance of an actor (transcript t_3)

```
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,_).
```