Paper accepted at RE'18

CaRE: A Refinement Calculus for Requirements Engineering Based on Argumentation Semantics

Yehia Elrakaiby*, Alessio Ferrari†, John Mylopoulos‡
* Lero - University of Limerick, Email: yehia.elrakaiby@lero.ie
† CNR-ISTI, Email: alessio.ferrari@isti.cnr.it
‡ University of Toronto and University of Trento, Email: jm@cs.toronto.edu

***Abstract*—The requirements problem consists of transforming stakeholder requirements - however informal, ambiguous, conflicting, unattainable, imprecise and incomplete – into a consistent, complete and realizable specification through a systematic process. We propose a refinement calculus for requirements engineering (CaRE) for solving this problem, which takes into account the typically dialectic nature of requirements activities. The calculus casts the requirement problem as an iterative argument between stakeholders and requirements engineers, where posited requirements are attacked for being ambiguous, incomplete, etc. and refined into new requirements that address the defect pointed out by the attack. Refinements are carried out by operators provided by CaRE that refine (e.g., strengthen, weaken, decompose) existing requirements, to build a refinement graph. The semantics of the operators is provided by means of argumentation theory. Examples are given to illustrate the elements of our proposal.**

I. Introduction

The core problem in requirements engineering (RE) consists of transforming the requirements elicited from stakeholders - however informal, ambiguous, conflicting, unattainable, imprecise and incomplete - through a systematic refinement process into a specification that is consistent, complete and realizable. Variants of this problem have been addressed by many contributions that constitute the backbone of RE research and go as far back as seminal contributions by Douglas Ross [1] and Michael Jackson [2]. Various IEEE/ISO Standards on Software Requirements Specifications, such as [3] and [4], have detailed the "defects" that need to be eliminated during the refinement process.

The state-of-the-art in RE for dealing with this core problem consists of approaches that offer refinement operators for eliminating specific types of defects. For example, Goal-Oriented RE (GORE) techniques offer AND/OR refinements to move requirements towards operationalizability (making requirements operationalizable through functions that the system-to-be can perform) [5]. Others propose refinements for eliminating various forms of conflict, including inconsistencies [6], [7] and obstacles [8]. Still others focus on recognizing ambiguity, also by means of natural language processing (NLP) [9].

Unfortunately, existing proposals are limited in that they focus on particular types of defect, e.g., inoperationalizability in the case of GORE, inconsistencies and obstacles in the specific case of KAOS [7], [8]. In this work-in-progress, we propose a refinement calculus intended to addresses all defects cited by the IEEE/ISO Standards through a small number of refinement operators. Our calculus includes operators for strengthening a requirement r into $r+$, i.e., $r+$ entails r; weakening a requirement r into $r-$, i.e., r entails $r-$; rejecting a requirement; introducing a new requirement; decomposing r into $r_1, \ldots, r_n$; and justifying r by introducing a "higher-level" requirement r' that explains the motivation for r[1].

The key contribution of this work is to formally define what does it mean for a specification to *address* (deal with) an initial set of requirements. Addressing a requirement constitutes a weaker notion than satisfying a requirement, because it allows for a requirement to be weakened or altogether dropped, as long as there is an acceptable argument for this.

Towards this end, we adopt argumentation semantics from Dung [10], a state-of-the-art argumentation framework for formal reasoning about arguments. The refinement process is viewed as an argument between stakeholders and requirements engineers where posited requirements are attacked by pointing out defects (e.g., "r is too strong/weak", "r is incomplete", "r is unattainable", "$r_1, \ldots, r_m$ are conflicting", *etc.*) and new requirements are proposed that remove the defect pointed out by the attack. Participants in the argument can counter-propose, or outright reject what has been proposed and so on. This argumentation process supports arguments of the form "The set of requirements given so far is incomplete because it doesn't say anything about privacy, whereas the system-to-be will be handling personal data", or "This requirements is not needed, drop it". The process leads to a refinement graph with many potential specifications S for an initial set of requirements R, each consisting of leaf nodes of the graph. Each S addresses R if there is an acceptable argument that derives S from R through refinement. This makes the derivation of S from R a Hegelian dialectic process of thesis-antithesis-synthesis [11], also similar in spirit to the inquiry cycle [12], though our proposal aims to include more structure, technical details and reasoning support for the RE process.

The envisioned contributions of this ongoing work, some of which are included in this report, are:

- A refinement calculus for RE that extends goal-oriented approaches and addresses a full set of defects, inspired by the IEEE/ISO standards;
- An argumentation semantics of what it means for a specification to satisfy a set of stakeholder requirements;

[1] In GORE terminology, r' is a higher-level goal than r justifying r.

- Reasoning support that, given an initial set of requirements R and a constructed refinement graph, returns all specifications S that satisfy R.

The remainder of the paper is structured as follows. In Sect. II, we present an informal overview of the calculus. In Sect. III, we show how we propose to use argumentation to provide semantics for the calculus. Finally, Sect. IV provides our research roadmap, together with final remarks.

II. Calculus Description

CaRE consists of a systematic process and a calculus for requirements elicitation, negotiation and refinement that supports documentation of the process, agreement between stakeholders[2] and management of change. The calculus consists of a collection of attacks types and refinements. The attack types we adopt have been inspired by the IEEE/ISO standards and represent defects that could be identified by stakeholders in one or more requirements. Refinements, on the other hand, are the means for fixing defects.

A. Attack Types

Attacks on requirements consist of pointing out defects that make them unacceptable according to some stakeholders. Requirements may be attacked individually or collectively. In the former case, we refer to an attack as Single-Target Attack (or STA) and in the latter as Multi-Target Attack (MTA).

a) Single-Target Attacks:

- NonAtomic: the requirement under attack is not operationalisable. For example, $\langle r_1$:"System shall schedule a meeting upon request"$\rangle$ is non-atomic since there is no single action the system-to-be can perform to fulfill it.
- Ambiguous: the requirement admits multiple interpretations because it is vague, imprecise or ambiguous. For example, $\langle r_2$:"The authentication process shall be easy"$\rangle$ is ambiguous since the term *easy* is vague.
- Unattainable: the requirement is not feasible, i.e. doesn't have a realistic solution. For example, $\langle r_3$:"The system shall be available at all times"$\rangle$ is unattainable because it assumes eternal availability of power and other resources.
- Unjustified: the requirement does not have an explicit motivation. For example, $\langle r_4$:"The system shall run on Windows operating system"$\rangle$ may be attacked for missing an explicit justification of why other operating systems are not considered.
- Incomplete: the requirement is missing information. For example, $\langle r_5$:"In case of fault, the system shall send an error message"$\rangle$ is incomplete because it does not specify a recipient of the message.
- TooStrong: the requirement is unnecessarily strong or unnecessary. For example, $\langle r_6$:"The website shall use HTTPS protocol"$\rangle$, may be too strong and unnecessary for a website that does not handle sensitive data.
- TooWeak: the requirement is too weak. For example, $\langle r_7$:"The DB system shall process 500 transactions/sec"$\rangle$ is too weak if the expected workload for the system-to-be is 1,000 transactions/sec.
- Rejected: the requirement is rejected. For example, in the context of an app recommending nearby restaurants to users, a requirement $\langle r_8$:"The app shall support chatting between the user and a recommended restaurant"$\rangle$ may be deemed unacceptable.

b) Multi-Target Attacks:

- mConflict: the collection of requirements under attack is not satisfiable, even though subsets are. For example, the requirements $\langle r_9$:"The train control system shall stop the train if a red signal is missed"$\rangle$ and $\langle r_{10}$:"The train control system shall not apply brakes if the speed is below 30 km/h"$\rangle$ (assuming that the driver is in charge for speeds <30km/h) are conflicting.
- mIncomplete: the set of requirements is missing needed requirements. For example, a set of requirements for a social network platform is mIncomplete if it does not include any privacy requirement.
- mTooStrong: here a collection of requirements is too strong or redundant. This is the case of requirements such as $\langle r_{11}$:"The system shall support authentication through fingerprint recognition"$\rangle$ and $\langle r_{12}$:"The system shall support authentication through iris recognition"$\rangle$.

Attacks have a type and take as arguments a single requirement or a set, as well as a statement that offers a rationale for the attack. For instance, TooWeak(r_7, "System is supposed to handle 1,000 tps"), or mConflict($\{r_9, r_{10}\}$, "train control can't be in manual and automatic mode at the same time").

B. Refinement Operators

Refinement operators are intended to remove defects pointed out by attacks. Each operator operates on a single requirement or a set under attack, and it is applicable for attacks of one or more attack types. One exception are attacks of type Rejected – if a requirement has been rejected. In this case, there is no possible fix and this requirement represents a dead end. The proposed operators are intended to address all attack types. They are as follows:

- **weaken**: produces a weaker requirement than its operand. For example, the unattainable requirement r_3 may be weakened into $\langle r_{13}$:"The system shall be available at all times, with interruptions of $\leq$2 hours"$\rangle$. As indicated earlier, **weaken** is applicable for attacks of type Unattainable and TooStrong.
- **strengthen**: produces a stronger requirement than its operand. For instance, r_7 may be strengthened into $\langle r_{14}$:"The system shall process 1,200 tps"$\rangle$. **strengthen** is applicable for attacks of type Incomplete, TooWeak and Ambiguous.
- **reduce**: decomposes its operand into a set $r_1, ..., r_n$ using AND-, OR-, or XOR-refinement. In the case of AND-refinement, the operand is satisfied if all requirements are satisfied, i.e. every r_i $(1 \leq i \leq n)$ is satisfied. In the case of OR-refinement, the operand is satisfied if any r_i

[2]In the body of the paper, the term "stakeholders" denotes also requirements engineers.

is satisfied. Finally, the operand of an XOR-refinement is satisfied whenever exactly one r_i is satisfied. **reduce** is applicable for attacks of type NonAtomic.

- **add**: introduces a new requirement and it is applicable for attacks of type Unjustified, Incomplete and mIncomplete.
- **resolve**: replaces a set of conflicting requirements $r_1, ..., r_n$ with a new set of non-conflicting requirements $r'_1, ...r'_m$. Conflict resolution typically consists of weakening or dropping some of the requirements in $r_1, ..., r_n$ such that the new set $r'_1, ...r'_m$ is not conflicting. **resolve** is applicable for attacks of type mConflict.

C. Incremental Construction of a Refinement Graph

The process of building a refinement (hyper)graph $G = \langle \mathbf{R}, \mathbf{E} \rangle$ from an initial set of stakeholder requirements R_0 proceeds as illustrated by the informal pseudo-code reported below. Each node $r \in \mathbf{R}$ in the graph is associated to a requirement. Each hyper-edge $e \in \mathbf{E}$, where $\mathbf{E} \subseteq \mathbb{P}(\mathbf{R}) \times \mathbb{P}(\mathbf{R})$ represents the application of a refinement operator. In the following, R denotes a set of requirements.

BUILD-REFINEMENT-GRAPH(R_0)
  $\mathbf{E} \leftarrow \emptyset$
  $\mathbf{R} \leftarrow R_0$
    do
      – Select $R \subseteq \mathbf{R}$, such that an attack type A applies on R
      – Define an attack of type A on R
      – Select a refinement operator o that is applicable for A
      – Apply o on R resulting in $r_1, \ldots, r_n$
      $\mathbf{R} \leftarrow \mathbf{R} \cup \{r_1, \ldots, r_n\}$
      $e \leftarrow \langle R, \{r_1, \ldots, r_n\} \rangle$
      $\mathbf{E} \leftarrow \mathbf{E} \cup e$
      while there are applicable attacks on any R
  return $G \leftarrow \langle \mathbf{R}, \mathbf{E} \rangle$

This process creates a refinement (hyper)graph with one (hyper)edge for every refinement that has as source(s) all operands of the refinement and as destinations all the new requirements resulting from the refinement. Leaf nodes in that graph are those that haven't participated in any refinement because there are no applicable attacks.

A specification consists of a set of leaf nodes of the refinement graph that together address all requirements in R_0. Note that, since no attack is applicable to leaf nodes, these must be atomic, and that is why they can be included in the specification. Specifications constitute solutions to the requirements problem defined by the initial set of stakeholder requirements R_0, by addressing all requirements in it. The semantics of addressability is founded on the notion of acceptability from argumentation theory [10]: a set of statements is acceptable if none of them is attacked, or none of the attacks against them are acceptable. Establishing acceptability constitutes the key problem in reasoning with arguments.

III. A SEMANTICS OF CARE BASED ON ARGUMENTATION

Argumentation theory studies the fundamental mechanism that humans use in argumentation and develops formal models and reasoning support to implement this mechanism on computers [10]. This section shows how refinement graphs can be mapped into arguments in ASPIC+ [13], a framework for structured argumentation that extends Dung's argumentation theory. Such a mapping gives formal semantics to CaRE refinement graphs and enables reasoning that determines whether a given set of stakeholder requirements is addressed by a specification. In the following, we consider a simplified version of the description of $ASPIC^+$ presented by Caminada and Amgoud [14]. An $ASPIC^+$ argumentation theory is a tuple $\langle \mathcal{L}, \mathcal{IR}, \mathcal{KB}, n \rangle$ where

- $\mathcal{L}$ is a logical language (closed under negation).
- $\mathcal{IR}$ is a set of *defeasible* inference rules of the form $\phi_1, ..., \phi_n \Rightarrow \phi$, where $\phi, \phi_1, ..., \phi_n$ are meta-variables ranging over the set of well-formed formulas of $\mathcal{L}$. A defeasible rule means that if one accepts all antecedents, then one must accept the consequent unless there is a sufficient reason to reject it. Defeasible rules with empty premises, i.e. rules of the form $\Rightarrow \phi$, are *assumptions*.
- $n : R_d \rightarrow \mathcal{L}$ is a partial function that gives names to (some) defeasible rules. Informally, $n(r)$ is a proposition which means that r is applicable.

We assume in this section that $\mathcal{L}$ is a propositional language. Moreover, two formulas ϕ and ψ are contradictory iff $\phi = \neg\psi$ and $\neg\phi = \psi$. Note that $\neg\neg\phi = \phi$.

A. Construction of Arguments & Attacks

a) Construction of Arguments: Consider an initial argumentation theory $\mathcal{AT}$. Let $\mathcal{L}$ of $\mathcal{AT}$ be a propositional language that includes the propositions {a,b,c}. Those propositions, put forward by stakeholder s_1, mean the following:

- a: trains are using the latest fail-safe air brake system (FABS),
- b: therefore, trains need 500m to stop,
- c: $\langle r_1$:"there shall be a minimal distance of 500m between trains"$\rangle$

Figure 1 depicts how the above statements are represented as an $ASPIC^+$ argumentation theory $\mathcal{AT}$ and the arguments that are constructed on the basis of it. In particular, $\mathcal{AT}$ includes $\{\Rightarrow a, a \Rightarrow b, arg_{\text{re}} : b \Rightarrow c\}$. The arguments constructed on the basis of $\mathcal{AT}$ are $\{A_1, A_2, A_3\}$[3]. Those arguments can be written as $A_1 = [\Rightarrow a]$, $A_2 = [A_1 \Rightarrow b]$ and $A_3 = [A_2 \Rightarrow c]$. The conclusion and premises of an argument A are denoted by $conc(A)$ and $prem(A)$ respectively. For example, $conc(A_3)$ and $conc(A_2)$ are c and b respectively. On the other hand, $prem(A_3)$ and $prem(A_2)$ are $\{a, b\}$ and $\{a\}$ respectively.

[3]Due to space limitations, we do present formally the construction of arguments on the basis of argumentation theories. Interested readers are referred to [13], [14].

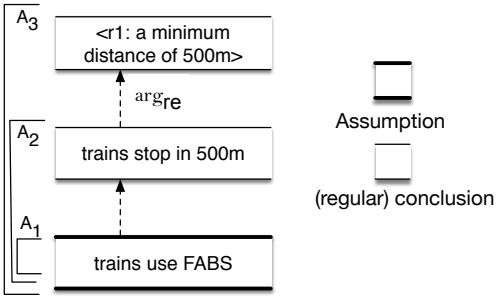

Fig. 1: Argument Construction

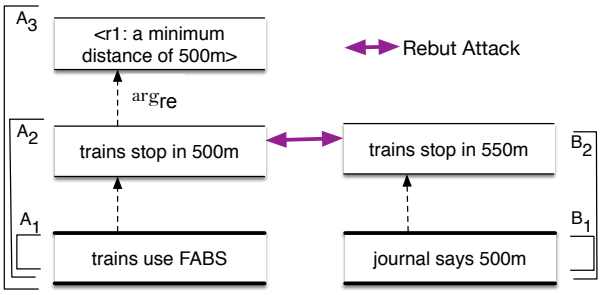

Fig. 2: Rebuttal Attack

b) Attacking Requirements: After identification of requirements, they are examined by stakeholders to determine whether they contain defects. In $ASPIC^+$, there are two basic types of attacks: *rebuttal* attacks which denote a conflict relation between the conclusions of two arguments and *undercut* attacks which represent an attack on the validity of some inference. The attacks identified in the previous section are mapped into the form of one of those two basic types.

For example, consider that stakeholder s_2 argues that "I read recently in a scientific journal that latest tests revealed that FABS requires 550m to stop". Therefore, "trains need 550m to stop and not 500m". This conflict situation is presented by adding the following propositions to the argumentation theory.

- d: I read recently that FABS requires 550m to stop,
- e: trains need 550m and not 500m to stop.

In $\mathcal{AT}$, those arguments will be represented by $\{\Rightarrow d, d \Rightarrow e\}$, leading to the construction of the arguments of B_1 and B_2 shown in Figure 2. Since the conclusion of B_2 is contradictory to the conclusion of A_2, B_2 is said to (rebut) attack A_2. B_2 also rebut attacks A_3 since A_3 is a continuation of A_2[4]. In the former case, the attack is called a *direct* attack, whereas in the latter case the attack is *indirect*. Notice that rebuttal attacks are *symmetric*, i.e. A_2 and A_3 also attack B_2.

c) Dung's Argumentation Framework: Argumentation allows reasoning about conflicts between arguments and determining which arguments are justified, i.e. should be accepted. In CaRE, this notion of acceptability determines the status of requirements, more precisely a requirement is said to be *acceptable* only it is justified. Notice that a requirement is acceptable only if does not have defects. $ASPIC^+$ relies on Dung's Argumentation Framework (DAF) [10]. A DAF is a pair $\langle \mathcal{A}, \mathcal{D} \rangle$, where $\mathcal{A}$ is a set of arguments and $\mathcal{D}$ is a set of attacks among those arguments. Based on DAFs, the justification of arguments is determined through the calculation of the so-called *extensions*. In this paper, we consider only the *grounded* extension, which represents the *skeptical* semantics of argumentation frameworks, i.e. the arguments that should "always" be accepted.

d) Construction of DAF: Based on the arguments and attacks in Figure 2, a DAF_{exmp} is constructed as follows:

$$\begin{aligned} \mathcal{A} &= \{A_1, A_2, A_3, B_1, B_2\} \\ \mathcal{D} &= \{(A_2, B_2), (B_2, A_2), (A_3, B_2), (B_2, A_3)\} \end{aligned}$$

e) Computation of Grounded Extensions: determines the arguments that are "always" *justified* or *acceptable*. The computation of the grounded extension for DAF_{exmp} produces the arguments $\{A_1, B_1\}$. This is interpreted as "only arguments A_1 and B_1 are (always) justified".

f) Justified Conclusions: are the statements or propositions that are justified. They are simply the conclusions of the arguments in the grounded extension. For example, the justified conclusions of $\mathcal{AT}$ before and after stakeholder s_2 pointed out the defect in r_1 are as follows:

- before: $\{a, b, c\}$
- after: $\{a, d\}$

Notice that the requirement r_1 was *acceptable* (c was acceptable) until the intervention of stakeholder s_2. To restore the acceptability of r_1, refinements are needed. Before the description of refinements, we first present the second basic attack type in $ASPIC^+$.

g) Undercut Attacks: Imagine that stakeholder s_3 argues that r_1 does not have a justification:

- g: r_1 is not justified.

In this case, s_3 does not attack the premises of the requirement argument arg_{re} but rather it points out a problem with the requirement itself. We represent this case in the form of an (undercut) attack as shown in Figure 3. The DAF computed for Figure 3 is as follows:

$$\begin{aligned} \mathcal{A} &= \{A_1, A_2, A_3, C_1\} \\ \mathcal{D} &= \{(C_1, A_3)\} \end{aligned}$$

Notice that, as opposed to the previous rebuttal attack, the undercut attack is asymmetric, i.e. C_1 attacks A_3 but not vice versa. The grounded extension and justified conclusions are:

- Extension: $\{A_1, A_2, C_1\}$
- Conclusions: $\{a, b, g\}$

Thus, this undercut attack renders the requirement, denoted by c, unjustified.

B. Refinements and Refinement Graph

a) Refinements: The computation of extensions reveals the state of requirements according to the negotiation between

[4]We refer interested readers to [13] for more detailed definitions of attacks and their identification.

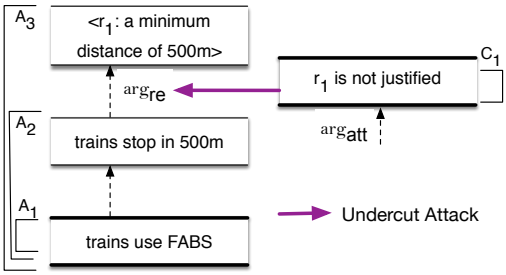

Fig. 3: Argument Construction

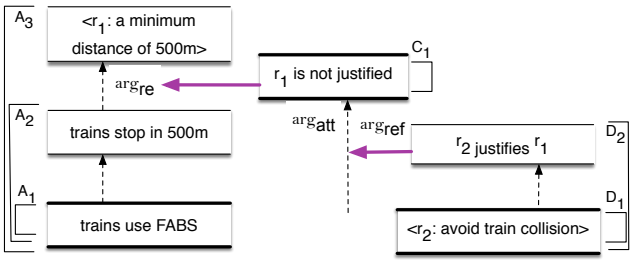

Fig. 4: Refinement

stakeholders, i.e. whether they are acceptable or not. This enables stakeholders to identify which requirements have defects and therefore should be refined. To address the defect in r_1, stakeholder s_2 proposes the following:

- i: $\langle r_2$:"avoid train collisions"$\rangle$,
- h: r_2 provides a justification for r_1.

Figure 4 depicts the statements above and how they affect the argumentation theory. In particular, by providing a justification for r_1, s_2 is able to fix the issue identified by stakeholder s_3 (i.e. that r_1 is not justified). This results in the construction of the arguments D_1 and D_2. By providing a justification for r_1, this refinement argument undercuts C_1. Consequently, the DAF computed for Figure 4 becomes as follows:

$$\begin{array}{lcl} \mathcal{A} & = & \{A_1, , A_2, A_3, C_1, D_1, D_2\} \\ \mathcal{D} & = & \{(C_1, A_3), (D_2, C_1)\} \end{array}$$

The grounded extension and justified conclusions are:

- Extension: $\{A_1, A_2, A_3, D_1, D_2\}$
- Conclusions: $\{a, b, c, i, h\}$

Thus, stakeholder s_3, by proposing the refinement above, is able restore the *acceptability* of r_1.

Note that defects that correspond to rebuttal attacks (Sect. III-A.b) are addressed through the specification of *preferences*. More precisely, we allow stakeholders to choose which, among the conflicting propositions, is more likely to be true. $ASPIC^+$ supports the expression of such preferences and the computation of extensions based on them. Due to space limitations, preferences will not be further discussed here but interested readers are referred to [13] for details on how preferences affect the computation of extensions.

b) Refinement Graph: Throughout the evolution of the negotiation process above, we construct an *argumentation theory* that tracks the evolution of requirements, attacks towards them and refinements that succeed in restoring their acceptability. On the other hand, the refinement graph shows the requirements and their update through refinements. Refinements shown in the graph are the ones that are *justified* through argumentation. Figure 5 shows an example of a fragment of a refinement graph, coupled with its corresponding argumentation graph. The figure depicts the progression of the refinement process: r_1 is found to be unjustified, since it does not have a clear motivation. This leads to a refinement that fixes the problem through the addition of a new requirement (**add** operator), r_2, which provides a motivation for r_1. If r_1 and r_2 are not attacked, then the refinement process ends. However, r_2 may be considered non-atomic, since it is too abstract. Therefore, the **reduce** operator is applied, and a novel requirement r_3 is produced, with the same content of r_1 – notice that other requirements with different content than r_1 could also be produced at this stage. By iterating this refinement process, leaf nodes will be all the nodes for which no attack is applicable, as specified in Sect. II-C.

IV. Roadmap and Conclusion

This paper presents work-in-progress on CaRE, a requirements engineering methodology equipped with an expressive calculus for requirements elicitation, negotiation and refinement that supports documentation of the requirements process, convergence towards consensus among stakeholders, rationalization and management of requirements change as follows:

- Negotiation and agreement: thanks to its argumentation structure and the dialectical nature of argumentation, CaRE is well suited for capturing the arguments exchanged between stakeholders and requirements engineers during the requirements engineering process;
- Refinement and elicitation: is supported through the systematic identification of defects (attacks) and refinements. In particular, this process leads to identification of missing requirements, rejection of unnecessary ones and refinement of those that are too strong or too weak;
- Documentation of the process: is supported by the refinement graph that captures arguments (requirements, attacks and refinements) exchanged between stakeholders in the requirements process;
- Rationalization of requirements change: is supported since every change is motivated by an attack and there is room for stakeholders to counter-attack during the refinement process.
- Management of change: is supported since requirements and attacks can be easily added, revised or removed, leading automatically to the computation of new extensions that show whether requirements are still acceptable or not. Refinement graphs are also updated accordingly.

The proposed calculus extends GORE approaches by offering a rich set of attack types as well as powerful refinement operators for addressing the defects pointed out in attacks.

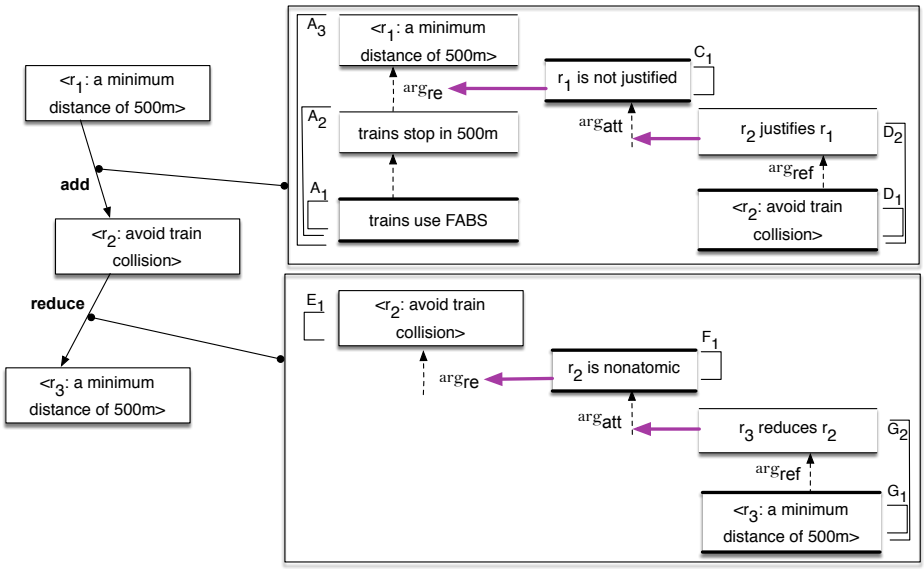

Fig. 5: A fragment of refinement graph, mapped with the argumentation graph

The closest work to our proposal is the thesis of Li [15] where a rich refinement calculus is proposed, but is not cast in argumentation terms and consequently lacks proper semantics. In [16], the Goal Structuring Notation (GSN) is used to explicitly represent the elements of safety arguments and their relationships. The main objective of this work is to present an argument that a system is acceptably safe to operate in a given context. In contrast, our work focuses on requirements refinements, negotiation and elicitation. There have been proposals in Software Engineering for capturing design rationale during the design process [17]. Such work differs from our proposal in that it is not intended to offer a calculus for design. Rather, it strives to model the rationale behind design decisions.

Moving forward, we plan to complete this research by providing a full formalization of CaRE in ASPIC+. Secondly, we will consider a more detailed and formal requirements language that includes assumptions, functional, and quality requirements, in the spirit of Li *et al.* [15], [18]. Thirdly, we propose to build a reasoner for refinement graphs that determines whether there is a specification for a given set of initial requirements. Finally, we propose to integrate into our framework existing NLP tools – as, e.g., SREE [9] – that support automated identification of requirements defects. Our goal is to come to a tool-supported method in which formal argumentation is transparent to the user, making CaRE easy to use by common requirements analysts. A cost-benefit evaluation on the practical applicability and scalability of CaRE is also foreseen as part of our future work.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and ERC Advanced Grant 291652.

References

[1] D. T. Ross, "Structured analysis (sa): A language for communicating ideas," *IEEE TSE*, no. 1, pp. 16–34, 1977.
[2] M. Jackson, "Information systems: Modelling, sequencing and transformations," in *ICSE'78*. IEEE Press, 1978, pp. 72–81.
[3] "IEEE Recommended Practice for Software Requirements Specifications," *IEEE Std 830-1998*, pp. 1–40, Oct 1998.
[4] "Iso/iec/ieee international standard - systems and software engineering – life cycle processes –requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, Dec 2011.
[5] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1-2, pp. 3–50, 1993.
[6] A. Hunter and B. Nuseibeh, "Managing inconsistent specifications: reasoning, analysis, and action," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 4, pp. 335–367, 1998.
[7] A. V. Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 10th ed. Chichester, UK: John Wiley & Sons, 2009.
[8] A. Van Lamsweerde, "Handling obstacles in goal-oriented requirements engineering," *IEEE TSE*, vol. 26, no. 10, pp. 978–1005, 2000.
[9] S. F. Tjong and D. M. Berry, "The design of SREE: a prototype potential ambiguity finder for requirements specifications and lessons learned," in *REFSQ'13*. Springer, 2013, pp. 80–95.
[10] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial intelligence*, vol. 77, no. 2, pp. 321–357, 1995.
[11] G. W. F. Hegel, *Ph´anomenologie des Geistes*, 1807.
[12] C. Potts, K. Takahashi, and A. I. Anton, "Inquiry-based requirements analysis," *IEEE software*, vol. 11, no. 2, pp. 21–32, 1994.
[13] S. Modgil and H. Prakken, "The ASPIC+ framework for structured argumentation: a tutorial," *Argument Comput.*, vol. 5, pp. 31–62, 2014.
[14] M. Caminada and L. Amgoud, "On the evaluation of argumentation formalisms," *Artif. Intell.*, vol. 171, no. 5-6, pp. 286–310, 2007.
[15] F.-L. Li and J. Mylopoulos, "Desiree-a refinement calculus for requirements engineering," *arXiv preprint arXiv:1604.03184*, 2016.
[16] T. Kelly and R. Weaver, "The goal structuring notation a safety argument notation," in *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.
[17] J. Conklin and M. L. Begeman, "gibis: A hypertext tool for exploratory policy discussion," *ACM Trans. Inf. Syst.*, vol. 6, no. 4, pp. 303–331.
[18] F.-L. Li, J. Horkoff, A. Borgida, G. Guizzardi, L. Liu, and J. Mylopoulos, "From stakeholder requirements to formal specifications through refinement," in *REFSQ'15*. Springer, 2015, pp. 164–180.

View publication stats