

Analysis of Multi-Party Agreement in Requirements Validation

Ivan J. Jureta
FRS-FNRS & PRECISE
University of Namur
ijureta@fundp.ac.be

John Mylopoulos
Dept. of Computer Science
University of Toronto
jm@cs.toronto.edu

Stéphane Faulkner
PRECISE
University of Namur
sfaulkne@fundp.ac.be

Abstract

A requirements engineering artifact is valid relative to the stakeholders of the system-to-be if they agree on the content of that artifact. Checking relative validity involves a discussion between the stakeholders and the requirements engineer. This paper proposes (i) a language for the representation of information exchanged in a discussion about the relative validity of an artifact; (ii) the acceptability condition, which, when it verifies in a discussion captured in the proposed language, signals that the relative validity holds for the discussed artifact and for the participants in the discussion; and (iii) reasoning procedures to automatically check the acceptability condition in a discussions captured by the proposed language.

1 Introduction

A basic question for requirements engineering (RE) is how to find out what the stakeholders of a system-to-be “really need” [6]. In response, RE starts with the *elicitation* of requirements, the purpose of which is the initial investigation of the goals, functions, and constraints of the system-to-be, as they are stated by the stakeholders. Despite the difficulty of making a clear distinction between the various specific tasks that RE can involve [6], elicitation is acknowledged to be one of the three fundamental tasks in RE, in addition to *validation* and *modeling/specification* [9, 13].

Modeling/specification depends heavily on elicitation, for it is elicitation that provides the application domain- and project-specific information that is, potentially in a changed form, represented in RE artifacts (i.e., models and specifications). RE artifacts capture the elicited information in a format that lends itself to specific analysis, which the stakeholders themselves have difficulty to perform, such as, e.g., the verification of the internal consistency of requirements. For elicitation as well as modeling/specification to be effective, elicited requirements must be validated. As Goguen and Linde observe, “[t]here are very good reasons why [stake-

holders] often do not, or cannot, know exactly what they need; they may want to see models, explore alternatives, and envision new possibilities” [6]. A key purpose of requirements validation is to seek feedback from the stakeholders on RE artifacts so as to inform further iterations of elicitation and/or modeling/specification. To check the validity of an RE artifact is to determine if what it says about the system-to-be is in line with what the stakeholders “really need”.

Problem. The aim of validation is ambitious: through repeated and intertwined performance of validation together with elicitation and modeling, we would indeed hope to arrive at RE artifacts that capture *exactly* what the stakeholders *really* need. Such *absolute validity* should be distinguished from what we call *relative validity*. While the former certainly stands as an ideal to aim for, the latter is achievable in practice and is the concern of this paper.

Relative validity is concerned with whether the stakeholders agree on the content of an RE artifact. Validity is in this sense *relative to the stakeholders*. An RE artifact is therefore valid in this sense if the *stakeholders* agree that what it says about the system-to-be is acceptable to them. Stated otherwise, this form of validity will verify if all the concerns, which the stakeholders raised, are answered.

It is safe to say that we cannot know if the stakeholders agree on an artifact if we do not give them the possibility to raise their concerns. The engineer can inform them in this task by providing graphical animations of a behavior model [12], the results of checking of predefined properties on models made from parsed text [5], explicit accounts of (the inconsistencies between) different viewpoints on the system-to-be [9]. In each of these cases, the engineer will be producing information to present in a potentially summarized form to the stakeholders, and then discuss it. *Checking relative validity inevitably leads to a discussion between the stakeholders and the requirements engineer.*

Contributions. This paper focuses on the modeling and analysis of a *discussion* between the stakeholders and requirements engineers about the relative validity of an RE artifact. By building on contributions in design rationale research, argumentation research in artificial intelligence,

and graph traversal algorithms, our aims are to: (i) provide a simple but expressive propositional model of the explicit exchange of information in what is usually called a discussion; (ii) based on the model of a discussion, propose a condition, called the *acceptability condition* on an artifact (denoted **AC**), such that if it holds, then it signals that the relative validity verifies for that artifact and for the participants in that discussion; and (iii) if a concrete discussion is recorded (as is the case when discussions are realized in forum-like applications), then check automatically if **AC** holds at some point in the discussion. To meet these aims, we propose the *Acceptability Evaluation* framework, henceforth ACE. ACE can be seen as a simple propositional reasoning framework, that is independent of the RE method that produces the artifact, and of the application domain.

Organization. We introduce first the acceptability condition (§2), then illustrate the framework for the automated checking of the acceptability condition (§3), note some implementation considerations (§4), and discuss related work (§5). We end with a summary of conclusions and point to future work (§6).

2 Baseline

What is typically called a discussion is, roughly speaking, a complex exchange of information between potentially many participants. Various properties of a discussion can be studied, such as its topic, purpose, (dis)organization, and so on. We focus on discussions about RE artifacts, the purpose of which is to reach a conclusion about the relative validity of the artifacts. We are interested in the specific traits of the structure of such discussions. These are the inference, attack, and preference relationships between pieces of information offered in a discussion. Inferences connect premises to conclusions, attacks connect somehow opposing information, and preferences compare in terms of desirability the conditions described via the various pieces of information offered in the discussion.

The range of discussions we focus on is not confined to specific RE methods and artifacts. Leite and Freeman [9] observed that “the whole process of [RE] is a web of subprocesses, and it is very difficult to make a clear distinction between them”. A subprocess in that web amounts to the application of some RE method to specific inputs, in order to produce an output, itself fed into the application of another method, and so on. To remain general, we can say that the application of any RE method, i.e., any subprocess in the complex RE web, fits the abstract input-transformation-output pattern. Namely, in a given application domain D , information elicited or produced by another RE method acts as the input I_D to a domain-independent RE method, symbolized by the function T . The latter produces the domain-specific output O_D , i.e., $O_D = T(I_D)$. E.g., the refinement of a

requirement asks for an abstract requirement and domain-specific knowledge as its inputs I_D , and results in a set of less abstract requirements as its output O_D , while the transformation T establishes the relations, such as consistency, that must verify between inputs and outputs. Observe that I_D and O_D , or any part thereof is clearly an RE artifact. Moreover, we can view the application of a method to specific inputs, i.e., $T(I_D)$ as an artifact itself: there really seems to be no strong argument not to allow the participants to discuss the engineer’s choice of applying T to I_D .

Discussion performed to the aim of checking the relative validity of the application of an RE method, i.e., $O_D = T(I_D)$, or equivalently, the relative validity of individual artifacts O_D , $T(I_D)$, and I_D , consists of offering information in favor or against these, and providing opinions about the relative desirability of the offered information. If I agree with you, I can provide additional information to support your position; if we disagree, I can offer information against your positions; if I have no further information to offer in favor of or against that which has been offered, I can say which of the already present conclusions I prefer to others. $O_D = T(I_D)$ will be acceptable if and only if no information offered against any of the components of $O_D = T(I_D)$ holds by the end of the discussion. *Acceptability signals agreement*. It is reasonable to interpret agreement as relative validity. It is by analysing a discussion that we can determine if there is agreement about the artifacts being validated, and thereby if they are valid relative to the participants in the discussion. If the parties agree that the given inputs I_D transformed by the application of the method T give O_D , then they agree that $O_D = T(I_D)$ holds, so that the given method application is acceptable, denoted $\mathbf{AC}(I_D, T(I_D), O_D)$.

Definition 1. AC. *The application of the RE method T to the input I_D to produce the output O_D is acceptable, denoted $\mathbf{AC}(I_D, T(I_D), O_D)$ if and only if:*

$$\mathbf{AC}(I_D) \text{ and } \mathbf{AC}(T(I_D)) \text{ and } \mathbf{AC}(O_D) \quad (1)$$

In order to apply to *any* method, ACE sees any RE artifact, or part thereof, as a proposition. In conceptualizing a *proposition*, we follow McGrath’s [11] stipulation that propositions “are the sharable objects of the attitudes [(i.e., what is believed, desired, etc.)] and the primary bearers of truth and falsity”. Regardless then of the syntax and semantics of the RE method deployed to produce an artifact, the artifact itself is a conjunction of propositions. Symbols p , q , and r , indexed when needed, denote individual propositions. $\mathbf{In}(I_D)$, $\mathbf{In}(O_D)$, and $\mathbf{In}(T(I_D))$ denote the sets of propositions, respectively in I_D , O_D , and $T(I_D)$. We assume that all propositions in I_D , T , and O_D are visible to all participant in a discussion about the relative validity of these artifacts. A participant having information in favor or against any proposition in $\mathbf{In}(I_D)$, $\mathbf{In}(O_D)$, and $\mathbf{In}(T(I_D))$ will voice that

information. We evaluate the acceptability of the individual propositions in $In(I_D)$, $In(O_D)$, and $In(T(I_D))$ in order to verify $\mathbf{AC}(I_D, T(I_D), O_D)$:

$$\mathbf{AC}(I_D, T(I_D), O_D) \\ \text{iff } \forall p \in In(I_D) \cup In(O_D) \cup In(T(I_D)), \mathbf{AC}(p) \quad (2)$$

The acceptability of a given proposition p is automatically verified in ACE via an algorithm that analyzes the information given in favor or against p , and captured via a language defined in the following section.

3 Evaluating Acceptability

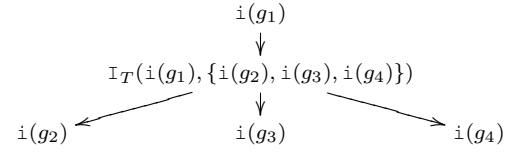
ACE has two components: (i) a language to record the information relevant to the evaluation of acceptability (§3.1), and (ii) algorithms for retrieving the recorded information and evaluating its acceptability (§3.2).

3.1 Language

All information relevant for the evaluation of acceptability is encoded into a directed labeled graph G , with the set of vertices $V(G)$ and lines $L(G)$, and the labeling functions λ_V and λ_L for vertices and lines, respectively. Any one proposition p or a conjunction of propositions in any of $In(I_D)$, $In(O_D)$, and $In(T(I_D))$ is captured by exactly one vertex $v \in V(G)$. As all lines carry the same label $\forall l \in L(G)$, $\lambda_L(l) = \mathbf{To}$, there is no need to write this label in graphs. There are four labels for vertices: $\forall v \in V(G)$, $\lambda_V(v) \in \{\mathbf{i}, \mathbf{I}, \mathbf{C}, \mathbf{P}\}$. G together with the labeling functions and the propositions forms the syntax of the language. The intended interpretation of the syntax is illustrated via the following example.

Suppose that the aim is to build a system that would deliver music on-demand: a user visits a website, chooses songs from a database, and can play them in the audio player on the website. An important goal is “Generate revenue from the audio player” (g_1), refined by the conjunction of (i) “Display text ads in the audio player” (g_2), (ii) “Target text ads according to users’ profiles” (g_3), and (iii) “Maintain the player free to all users” (g_4). We therefore have $I_D = g_1$ and $O_D = g_2 \wedge g_3 \wedge g_4$. The applied RE method is the standard AND-refinement of a goal [4]. We capture the application of AND-refinement in the example via the graph shown in Ex.1. The refined goal g_1 and the components g_2, g_3, g_4 of its refinement are assigned the label \mathbf{i} because of their role as the inputs and outputs to the application of an inference rule, denoted $\mathbb{I}_T(\mathbf{i}(g_1), \{\mathbf{i}(g_2), \mathbf{i}(g_3), \mathbf{i}(g_4)\})$.

(Ex.1)



The label \mathbf{i} is assigned to an *information* vertex, which serves as the input and/or output to the application of an *inference rule*, corresponding to the label \mathbb{I}_T . An inference rule vertex in G represents the application of some particular rule of deductive or ampliative inference to inputs in order to obtain the given outputs. An example of deductive inference is modus ponens. Inference or reasoning is ampliative when a conclusion is inferred, which includes information absent from the premises, from which the conclusion is inferred. Examples of rules of ampliative inference are induction by enumeration, reasoning with analogies, causal reasoning. Refinement, as any method is an inference rule, so that the application of AND-refinement is represented by $\mathbb{I}_T(\mathbf{i}(g_1), \{\mathbf{i}(g_2), \mathbf{i}(g_3), \mathbf{i}(g_4)\})$ in the graph. Any transformation T translates into at least one \mathbb{I} vertex; it will equate to more vertices when we are not content with evaluating the acceptability of the application of the method as a whole (i.e., the black box approach), but are interested in evaluating in detail the acceptability of the various steps or other considerations called for in the application of the method. The meaning of each line is derived from the vertices it connects, and its direction. The line from $\mathbf{i}(g_1)$ to $\mathbb{I}_T(\mathbf{i}(g_1), \{\mathbf{i}(g_2), \mathbf{i}(g_3), \mathbf{i}(g_4)\})$ is understood as stating that the former is the input to the application of the given inference rule, \mathbb{I}_T .

An information vertex (\mathbf{i}) can be involved in the application of a conflict (\mathbf{C}) or of a preference rule (\mathbf{P}). Suppose that a stakeholder indicates that “Revenue can be generated by charging subscriptions to users” (p_1). A vertex labeled \mathbf{C} indicates an application of a *conflict rule*, that is, the application of criteria giving rise to a conflict between two or more other vertices in the graph. Since it is clear that not all features of the player are free when a paid subscription is available, we add the conflict vertex $\mathbf{C}_1(\mathbf{i}(p_1), \mathbf{i}(g_4))$ to the graph. Additional information can be immediately found to elaborate on the subscription revenue model: (i) “Part of the music database can be restricted, so that the player only plays 30 seconds of some songs, until the user buys a subscription to listen full songs” (p_2); (ii) “According to competitors’ services, some users are willing to pay to choose a different graphical layout for the online audio player; users can be allowed to choose among different graphical layouts and pay for each” (p_3); and (iii) “Two versions of the player can be offered, one with basic and free features, and another with advanced features requiring subscription” (p_4). We choose to relate each of p_2, p_3, p_4 via the modus ponens

inference rule to p_1 . Say that a survey concludes that users strictly prefer a free music on-demand service to one based on subscription. We capture this strict preference by the preference vertex $P_1(i(g_4), i(p_1))$ and lines from $i(g_4)$ to $P_1(i(g_4), i(p_1))$, and from $P_1(i(g_4), i(p_1))$ to $i(p_1)$ in accordance to the direction of preference. A preference vertex represents the application of a *preference rule*, that is, the application of criteria defining a strict preference order between the conditions described in two or more other vertices. If the stakeholders agree that $P_1(i(g_4), i(p_1))$ resolves the conflict $C_1(i(p_1), i(g_4))$, then an application of a conflict rule will be added, $C_2(P_1, \{C_1, i(p_1)\})$, from $P_1(i(g_4), i(p_1))$ to $C_1(i(p_1), i(g_4))$ and $i(p_2)$. Figure 1 summarizes this discussion; applications of inference, conflict, and preference rules are given in the abbreviated form therein (i.e., P_1 is written in place of $P_1(i(g_4), i(p_1))$), and each application of the modus ponens inference rule is indexed differently as it takes different inputs, i.e.:

- $I_{MP,1}(\{i(p_2) \rightarrow i(p_1), i(p_2)\}, i(p_1))$,
- $I_{MP,2}(\{i(p_3) \rightarrow i(p_1), i(p_3)\}, i(p_1))$, and
- $I_{MP,3}(\{i(p_4) \rightarrow i(p_1), i(p_4)\}, i(p_1))$.

There are three constraints (1)–(3) imposed by the syntactic constraints on the To relationship. (1) Any two vertices in G can be connected by at most one line. (2) No two information vertices can be connected; any information vertex must be connected to an inference, conflict, or preference vertex, for it is these vertices that establish the use to which the information vertices are put in G . (3) Any inference, conflict, or preference vertex must have at least one line that enters it, and another that exits it. There are no restrictions on the label of vertices to which an inference, conflict, or preference vertex can be connected. This makes the language rather versatile, as some forms of meta-reasoning can be captured. A preference may be given between other preferences (e.g., $P_1 \rightarrow P_3(P_1, P_2) \rightarrow P_2$) to capture the priority among preferences. Inference rules can be compared in terms of preference (e.g., $I_1 \rightarrow P(I_1, I_2) \rightarrow I_2$). Conflicts between preferences can be described, along with conflicts between conflicts, and conflicts between applications of inference rules.

3.2 Algorithms

The graph in Figure 1 is a summary of the information offered in favor of and against the application of the AND-refinement method in Ex.1. Given such a graph, two tasks are relevant. The first, *retrieval* task is to search for particular subgraphs G in order to retrieve information that may be of relevance for further discussion among the participants and the evaluation of acceptability. The second, *evaluation* task

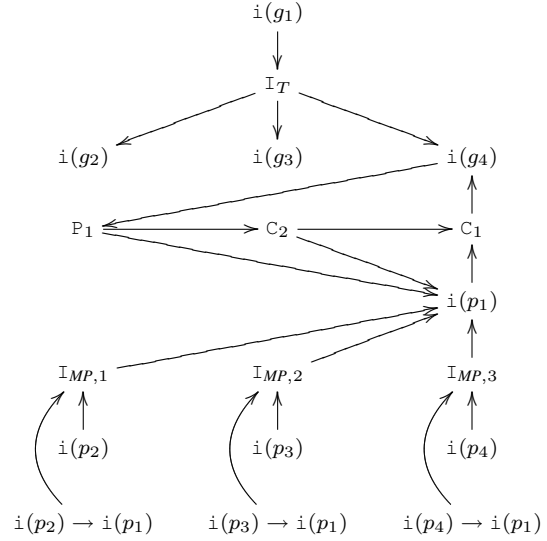


Figure 1. Discussing a refinement.

is to determine if some specific application of an RE method is acceptable.

Both the retrieval and evaluation tasks rely on the following two important notions. A vertex $v \in V(G)$ is *attacked* iff there is a line $l \in L(G)$ from a conflict vertex to v , i.e., $\exists l = v'v \in L(G)$ s.t. $\lambda_V(v') = C$. A vertex $v \in V(G)$ is *dominated* iff there is a line from a preference vertex to v , i.e., $\exists l = v'v \in L(G)$ s.t. $\lambda_V(v') = P$.

Retrieval. Suppose that we are interested in all vertices in G from Figure 1 that is directly in favor of a vertex $i(g_4)$. The result sought is the subgraph $i(g_1) \rightarrow IT \rightarrow i(g_4)$, because $i(g_4)$ is inferred from $i(g_1)$ by the application of IT . $i(g_4)$ is attacked via C_1 , so that C_1 cannot be in favor of $i(g_4)$. If we seek all vertices directly against $i(g_4)$, the result is the subgraph $i(p_1) \rightarrow C_1 \rightarrow i(g_4)$. It is, however, considerably more interesting to search for *all* (i.e., direct *and* indirect) vertices in favor of or against a given vertex, say $i(g_4)$. This equates to searching for the subgraph of G , which contains exactly all simple paths that end in $i(g_4)$. Such a subgraph is important, as it contains all vertices in G that are of interest when evaluating the acceptability of $i(g_4)$ alone. The breadth first search-like Algorithm 1 retrieves all vertices in favor of or against a given vertex in G , i.e., it retrieves a *discussion* of the given vertex.

Proposition 1. *Algorithm 1 applied to a vertex v in an ACE graph (i) does not loop indefinitely, (ii) returns all direct and indirect vertices in favor of or against the starting vertex v , and (iii) has the running time of $O(|V(D[v])| + |L(D[v])|)$.*

Proof. We first prove that the algorithm (i) does not loop indefinitely. $V(G)$ and $L(G)$ are finite; the **while** loop explores only incoming

Algorithm 1 Find Discussion

Require: ACE graph G , a starting vertex $First \in V(G)$
Ensure: Graph $D[First]$, which is a subgraph of G

```

1: procedure FINDDISCUSSION( $G, First$ )
2:   Empty the queue  $Q$ ;  $V(D[First]) \leftarrow \emptyset$ ;  $L(D[First]) \leftarrow \emptyset$ 
3:   Add  $First$  to  $Q$ 
4:   while  $Q$  is not empty do
5:     for each vertex  $v$  in  $Q$  do
6:       Add  $v$  to  $V(D[First])$ 
7:       for each  $v' \in V(G)$  s.t.  $\exists v'v \in L(G)$  do
8:         if  $v'v \notin L(D[First])$  then
9:           Add  $v'v$  to  $L(D[First])$ 
10:        end if
11:        if  $v' \notin V(D[First])$  then
12:          Add  $v'$  to  $V(D[First])$ 
13:          Add  $v'$  to  $Q$ 
14:        end if
15:       end for
16:       Delete  $v$  from  $Q$ 
17:     end for
18:   end while
19: end procedure

```

lines to a given vertex, and never adds the same line or vertex twice to $D[v]$. It follows that the algorithm will never loop indefinitely.

We prove by contradiction that the algorithm (ii) returns all direct and indirect vertices in favor of or against the starting vertex v . Suppose that there is a vertex v' that is either in favor or against the starting vertex v , and that is not visited by the algorithm. The inner **for each** loop (Lines 7–15) moves from the starting vertex along its incoming lines to its nonvisited adjacent vertices, adds these to the queue Q , and removes the starting vertex. The **while** loop guarantees that any vertex added to Q is visited, along with its outgoing lines. The **while** loop thereby ensures that any vertex in G having a path to the starting vertex is visited. If v' was not visited by the algorithm, then v' is not on a path that ends in v . It is therefore a contradiction that v' is in favor or against v , but that it has not been found by the algorithm.

Finally, we prove that the algorithm (iii) has the running time of $O(|V(D[v])| + |L(D[v])|)$. The **if-then** blocks in the inner **for each** loop (Lines 7–15) guarantee that no vertex or line in G will enter Q more than once, and that all lines and vertices visited for the first time will be added to $D[v]$. It follows that the worst case arises when $D[v] = G$, so that the algorithm will traverse all lines and vertices of G , which gives the $O(|V(G)| + |L(G)|)$ as the upper bound on the time complexity, and $O(|V(D[v])| + |L(D[v])|)$ as the time complexity for the algorithm. \square

Evaluation. The acceptability of the vertex v is evaluated by traversing and computing the labels on vertices in the discussion of v , $D[v]$. The *computed* label is a secondary label, and is different from that assigned by λ_V . The computed label of any vertex in $D[v]$ is either **A** for *accepted*, **AD** for *accepted and dominated*, or **R** for *rejected*. We illustrate the computation of labels by simple examples first,

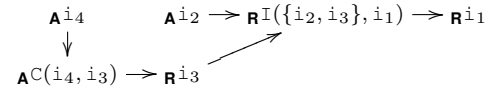
then go on to label the discussion $D[i(g_4)]$ obtained from the graph in Figure 1, and finally give an informal outline of the algorithm that computes the labels of any discussion.

Consider an ACE graph G' with only a single vertex $V(G') = \{v\}$ and $L(G') = \emptyset$, so that $D[v] = G'$. Being alone in $D[v]$, v is neither attacked nor dominated; we therefore say that v is acceptable, and label it **A**, denoted $\mathbf{A}v$. Consider now G'' with three vertices and two lines between them. To compute labels in G'' , we need to know the direction of the lines and the primary labels on the vertices:

- if G'' is $i_1 \rightarrow \mathbf{I}(i_1, i_2) \rightarrow i_2$, then all vertices are neither attacked nor dominated, and they all take the label **A**, i.e., $\mathbf{A}i_1 \rightarrow \mathbf{A}\mathbf{I}(i_1, i_2) \rightarrow \mathbf{A}i_2$;
- if G'' is $i_1 \rightarrow \mathbf{C}(i_1, i_2) \rightarrow i_2$, then i_2 is attacked; we see that i_1 and \mathbf{C} are not attacked, and conclude that i_2 is rejected: $\mathbf{A}i_1 \rightarrow \mathbf{A}\mathbf{C}(i_1, i_2) \rightarrow \mathbf{R}i_2$;
- if G'' is $i_1 \rightarrow \mathbf{P}(i_1, i_2) \rightarrow i_2$, then i_2 is dominated. To be dominated alone is not enough for rejection, so that i_2 is accepted and dominated, that is $\mathbf{A}i_1 \rightarrow \mathbf{A}\mathbf{P}(i_1, i_2) \rightarrow \mathbf{AD}i_2$.

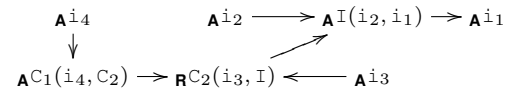
These three cases illustrate the first important principle used in computing the labels of a discussion: the label on a vertex v depends on the labels of all vertices v_1, \dots, v_n adjacent to v by lines v_1v, v_2v, \dots, v_nv . This alone is not enough, as we must know how the labels interact – consider the hypothetical discussion $D[i_1]$ in Ex.2.

(Ex.2)



The inference $\mathbf{I}(\{i_2, i_3\}, i_1)$ uses two inputs, one accepted i_2 and another i_3 , which is attacked by the accepted i_4 (hence the rejection of i_3). The inference itself cannot be accepted, since one of its inputs is rejected. Given that the application of the inference rule is rejected, the conclusion of the inference, i_1 , must be rejected as well. Ex.13 illustrates the choice that the label **R** has priority over **A**. We choose to be cautious in computing the labels, meaning that **R** has priority over **AD** and **A**, and **AD** has priority over **A**. In addition to this second principle employed in the computation of the labels, we have a third and final one, illustrated via Ex.14.

(Ex.3)



Suppose that no computed labels are given in Ex.14. We see immediately that i_4 , i_3 and i_2 should be accepted as they are not attacked, along with $C_1(i_4, C_2)$. $C_2(i_3, I)$ must then be rejected, as it is attacked via $C_1(i_4, C_2)$. Observe then that $I(i_2, i_1)$ has two incoming lines, one from the accepted i_2 and another from the rejected conflict $C_2(i_3, I)$. While it is true that **R** has priority over **A**, we conclude that $I(i_2, i_1)$ is accepted, because the rejected conflict is not an input to the inference $I(i_2, i_1)$. We have noted earlier that the meaning of a line in an ACE graph is determined from the labels that λ_V assigns to the vertices connected by the line. The conclusion that $I(i_2, i_1)$ is accepted cannot be reached without determining the meaning of each line ending in $I(i_2, i_1)$. By reading these lines, we see that $I(i_2, i_1)$ does not take the conflict $C_2(i_3, I)$ as its input, but that this conflict attacks $I(i_2, i_1)$. If the conflict is accepted, $I(i_2, i_1)$ should be rejected; however, the conflict is rejected, so that $I(i_2, i_1)$ is accepted. More generally, the third principle we use in computing the label on a vertex v is that the meaning of each line that ends in v must be determined. This leads us to define a number of deduction rules for labels, which account for the meaning of the relevant line. To see how these rules are used, consider again the vertex $I(i_2, i_1)$. Since it has lines incoming from two different vertices, we use the following two *label deduction rules*:¹

- from $\mathbf{A}i \rightarrow I(i, \cdot)$, conclude that the inference $I(i, \cdot)$ should be accepted (where $\mathbf{A}i \rightarrow I(i, \cdot)$ means that the inference $I(i, \cdot)$ uses the accepted i as its input and concludes something else, i.e., “.”); and
- from $\mathbf{R}C(\cdot, I) \rightarrow I(\cdot, \cdot)$, conclude that the inference $I(\cdot, \cdot)$ should be accepted (where $\mathbf{R}C(\cdot, I) \rightarrow I(\cdot, \cdot)$ means that the inference $I(\cdot, \cdot)$ is attacked by the rejected conflict $C(\cdot, I)$).

The application of two rules, as above, gives us two labels, both **A**; it is thus clear that $I(i_2, i_1)$ will bear the label **A**. More generally, if v has n incoming lines v_1v, v_2v, \dots, v_nv , then we will apply n rules, selected depending on the label of each $v_i \in \{v_1, v_2, \dots, v_n\}$. This will result in n labels. The one label that we will assign to v will be that of the n labels, which has the priority over others, according to the principle of how the labels interact, and given above. E.g., if we have the set of n labels, in which there is at least one label **R**, we will conclude $\mathbf{R}v$; if each label is **A**, then $\mathbf{A}v$; if there are no **R** labels, but only **A** and **AD** labels, then $\mathbf{AD}v$. Seventy two label deduction rules cover all cases allowed by the meaning of the **To** line in any ACE graph. They are all listed in the full version of this paper [8].

¹As a notational convention, we write “.” for *any*, i.e., the parameter that is not important for the application of the given rule.

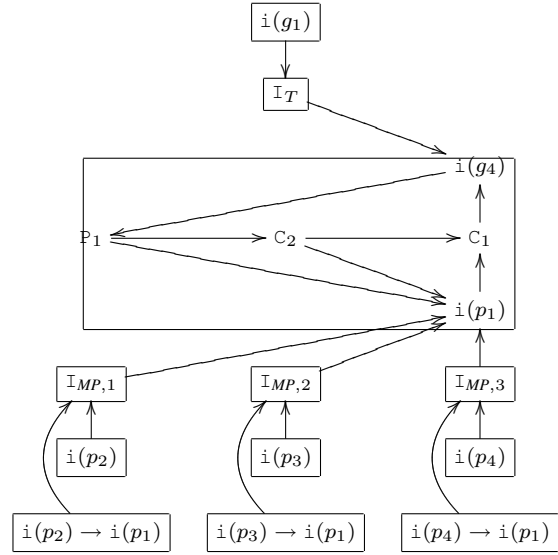


Figure 2. Strongly connected components in the discussion from the example.

We now ask if $i(g_4)$ is acceptable. The answer can be given once we compute the labels on the discussion $D[i(g_4)]$. We find the discussion $D[i(g_4)]$ via Algorithm 1 and then proceed as follows:

1. If there are preferences in the discussion that are transitive, the steps below are performed on the transitive closure of these transitive preferences on the discussion of choice. Regardless of whether P_1 is transitive, the transitive closure of P_1 on $D[i(g_4)]$ is the same as $D[i(g_4)]$.
2. We now need to find the topological sort of the strongly connected components of the discussion $D[i(g_4)]$. A discussion can contain cycles, which is why we must first identify the strongly connected components.² Figure 2 shows the discussion $D[i(g_4)]$, where each strongly connected component is delimited by a rectangle. The largest strongly connected component contains cycles, while the others contain no cycles. Once we have the topological components, we need their topological sort. It is well known that contracting each strongly connected component in a directed graph gives a directed acyclic graph, where each vertex is a contracted strongly connected component. The topological sort of that directed acyclic graph is a linear ordering of its vertices, in which a vertex comes before all vertices, to which it has outgoing lines. To see why we need the topological sort of the strongly connected

²As usual, a strongly connected component is a graph, in which there is a path from any vertex to any other vertex.

components, consider the problem of labeling $\mathbb{I}_{MP,1}$ (same applies to the problem of labeling $\mathbb{I}_{MP,2}$ and $\mathbb{I}_{MP,3}$): the label of $\mathbb{I}_{MP,1}$ depends on the labels of $\dot{i}(p_2)$ and $\dot{i}(p_2) \rightarrow \dot{i}(p_1)$. Consequently, we must label $\dot{i}(p_2)$ and $\dot{i}(p_2) \rightarrow \dot{i}(p_1)$ before we label $\mathbb{I}_{MP,1}$. In the topological sort, $\mathbb{I}_{MP,1}$ comes after both $\dot{i}(p_2)$ and $\dot{i}(p_2) \rightarrow \dot{i}(p_1)$. The topological sort therefore gives the order, in which the strongly connected components should be labeled.

3. Given the topological sort of the strongly connected components of the discussion $D[\dot{i}(g_4)]$, we label all elements in the sort that have no incoming lines. We consequently label as accepted the following vertices: $\mathbf{A}\dot{i}(g_1)$, $\mathbf{A}\dot{i}(p_2)$, $\mathbf{A}\dot{i}(p_3)$, $\mathbf{A}\dot{i}(p_4)$, $\mathbf{A}(\dot{i}(p_2) \rightarrow \dot{i}(p_1))$, $\mathbf{A}(\dot{i}(p_3) \rightarrow \dot{i}(p_1))$, and $\mathbf{A}(\dot{i}(p_4) \rightarrow \dot{i}(p_1))$. Once these are labeled, the next elements in the sort are the three applications of modus ponens and \mathbb{I}_T . They are not attacked and their inputs are accepted, so that $\mathbf{A}\mathbb{I}_{MP,1}$, $\mathbf{A}\mathbb{I}_{MP,2}$, $\mathbf{A}\mathbb{I}_{MP,3}$, and $\mathbf{A}\mathbb{I}_T$. Labeling $\dot{i}(p_1)$ is more difficult and requires a different strategy. This is because $\dot{i}(p_1)$ lies on at least one simple cycle.³ To label a strongly connected component with cycles, we proceed as follows:

- (a) The count, say C of simple cycles in the strongly connected component is computed. $C = 3$ in the strongly connected component containing $\dot{i}(p_1)$.
- (b) We add an empty sequence of labels on each vertex in the strongly connected component, and add the label \mathbf{A} to the sequence of each vertex.
- (c) We choose a vertex according to a specific heuristic (namely, we take the last added vertex in the strongly connected component; the choice of this heuristic is discussed in the full paper) and call it the *First* vertex. In $D[\dot{i}(g_4)]$, P_1 is that vertex. In doing so, we in fact merely hypothesize that P_1 is accepted. To understand intuitively what happens next, suppose that there are C walkers stationed at P_1 . Each walker obeys the following: (i) it takes equal time to traverse a vertex; (ii) it can only go forward (i.e., over lines that start in a vertex); and (iii) no two walkers will start from *First* and return to *First* along the exact same path. In the strongly connected component with $\dot{i}(p_1)$, we place three walkers at the vertex P_1 and send them along the lines outgoing from P_1 . After the first step, two walkers will reach C_2 and one will reach $\dot{i}(p_1)$. Once they reach a vertex, they compute the label on that vertex by using the label

deduction rules we explained earlier. The computed label is appended to the sequence of labels on the vertex. For $\langle \mathbf{A} \rangle C_2$, we append the sequence of labels with \mathbf{A} , and obtain $\langle \mathbf{A}, \mathbf{A} \rangle C_2$. Since all three applications of modus ponens are accepted and $\langle \mathbf{A} \rangle P_1$, we get $\langle \mathbf{A}, \mathbf{AD} \rangle \dot{i}(p_1)$. After the second step, two walkers are at C_1 , and the third is at $\dot{i}(p_1)$, so that $\langle \mathbf{A}, \mathbf{A}, \mathbf{R}, \mathbf{R} \rangle C_1$ and $\langle \mathbf{A}, \mathbf{AD}, \mathbf{R} \rangle \dot{i}(p_1)$. The fourth step results in $\langle \mathbf{A}, \mathbf{A} \rangle \dot{i}(g_4)$. After the fourth step, two walkers arrive simultaneously at the first vertex P_1 . However, the first vertex obtains its second label (i.e., $\langle \mathbf{A} \rangle P_1$ becomes $\langle \mathbf{A}, \mathbf{A} \rangle P_1$) only after the slowest walker (one on the longest path back to the first vertex) arrives at that vertex.⁴

The stopping criterion for the walkers is as follows. If the last two labels in the sequence of labels on the first vertex are identical, the walkers are not sent to traverse the strongly connected component any further. This is the case in the example, where $\langle \mathbf{A}, \mathbf{A} \rangle P_1$. The last label in the sequence of labels on any vertex is the label that indicates the acceptability of that vertex: if \mathbf{A} , the vertex is acceptable, if \mathbf{AD} , the vertex is acceptable and dominated, if \mathbf{R} , the vertex is not acceptable. In case the first two labels in the sequence of the first vertex are not identical, the walkers will be sent out in the same way as described above, until the first vertex has four labels. When the first vertex has four labels and its last two labels are not identical, then the given discussion is inconclusive with regards to acceptability: more vertices need to be added (i.e., the discussion should continue) before the discussion is evaluated again.

The sequence of steps exemplified above is the informal outline of the algorithm that labels any discussion. The algorithm, called EVALUATEDISCUSSION takes an unlabeled discussion and returns a labeled discussion. The result of the application of the algorithm on the discussion in Figure 2 is shown in Figure 3. The algorithm is formally presented in the full paper [8], and the proofs of its correctness, termination, and time complexity are given. For a given discussion $D[v]$, EVALUATEDISCUSSION has the running time in $O(C(D^c[v])(|L(D^c[v])| + 2|V(D^c[v])|))$, where $C(D^c[v])$ is the number of simple cycles in $D^c[v]$ and $D^c[v]$ is the transitive closure of the transitive preference rules in $D[v]$ on the discussion $D[v]$.

We wrote in Equation 2 (§2) that $\mathbf{AC}(O_D = T(I_D))$ holds if and only if $\forall p \in \text{In}(I_D) \cup \text{In}(O_D) \cup \text{In}(T(I_D))$, $\mathbf{AC}(p)$. We can now complete this condition with the following:

³As usual, a simple cycle is a cycle that passes once through all vertices except its starting vertex, which the cycle passes twice (as the cycle starts and ends in that vertex).

⁴Observe that the number of labels in the sequence of labels of a vertex, in a strongly connected component with cycles, equals 1 plus the number of times a walker traversed that vertex. This is valid for all vertices other than the *First* vertex.

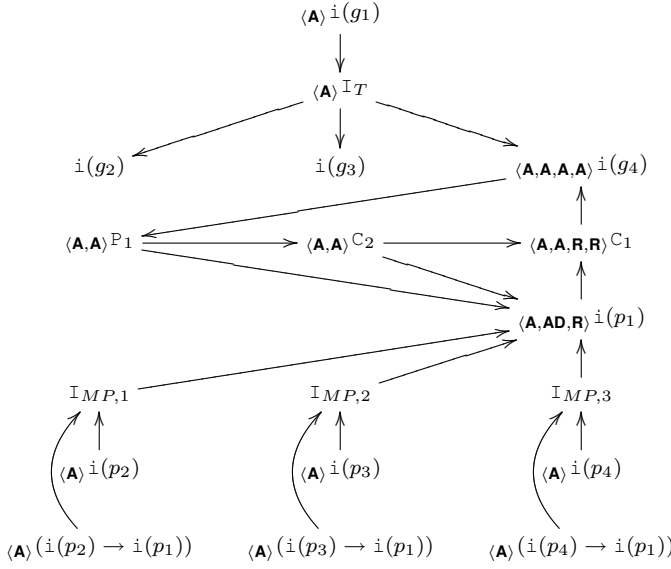


Figure 3. Example of a labeled discussion.

$$\begin{aligned} \forall p \in \text{In}(I_D) \cup \text{In}(O_D) \cup \text{In}(T(I_D)), \mathbf{AC}(p) \\ \text{iff } v(p) \text{ has } \mathbf{A} \text{ or } \mathbf{AD} \text{ as the last label} \\ \text{in its sequence of labels} \\ \text{in EVALUATEDISCUSSION}(D[v(p)]) \quad (3) \end{aligned}$$

where $v(p)$ is the vertex carrying the proposition p in the ACE graph, from which the discussion $D[v(p)]$ is obtained. A practical property in ACE is that if $v' \in V(D[v])$, then $D[v']$ is a subgraph of $D[v]$, so that it is unnecessary to compute $\text{EVALUATEDISCUSSION}(D[v'])$ if $\text{EVALUATEDISCUSSION}(D[v])$ is known: the latter gives us the evaluation of the acceptability of v' .

We observe in Figure 3 that $i(g_4)$, $i(g_1)$ and \mathbb{I}_T are acceptable. As $D[i(g_2)]$ is $\mathbf{A}i(g_1) \rightarrow_{\mathbf{A}} \mathbb{I}_T \rightarrow i(g_2)$, we conclude that $\mathbf{A}i(g_2)$. Similarly, $D[i(g_3)]$ is $\mathbf{A}i(g_1) \rightarrow_{\mathbf{A}} \mathbb{I}_T \rightarrow i(g_3)$, leading us to the conclusion that $\mathbf{A}i(g_3)$. We therefore observe that the application of the AND-refinement \mathbb{I}_T on the goal g_1 to obtain g_2 , g_3 and g_4 is acceptable, and that relative validity holds for this refinement, until new information is offered against it and the acceptability needs to be evaluated again.

4 Notes on Implementation

The implementation of ACE graphs and of the associated retrieval and evaluation algorithms is in progress at the time of writing and is based on the adaptation of standard open source internet forum software. This choice is based on two observations: (i) internet forums are popular means for discussion, and are a well known kind of software, having

evolved from bulletin board systems of the early 1970s; and (ii) a discussion in ACE amounts to a forum discussion performed according to a set of simple rules. By default, a discussion in an internet forum contains a collection of untyped posts. Any post may be a response to the original post (i.e., the root post), or a response to a later post. A forum discussion thus typically resembles an unlabeled tree, where each post is a vertex. To obtain an ACE discussion instead of a standard forum discussion, the following rules must be followed by the participants: (1) Any post is either an information, inference, conflict, or preference post. (2) Any post that is not the first post in a discussion, must be related to another post according to the meaning of the To relationship.

Compared to a classical forum, an ACE forum is thus one where a user chooses the label for a new post, and relates that post to others. The first rule above ensures that we have the label for any post, while the second rule guarantees that no post is disconnected from the others. A discussion in an ACE forum can thus be interpreted as an ACE graph (or ACE discussion), on which we can perform retrieval and evaluation operations via the algorithms discussed earlier. The evaluation algorithm provides to participants the indication on the acceptability of each post, so that they can, in case of rejection, intervene and respond in defense of their claims.

5 Discussion and Related Work

ACE is the continuation of our efforts to advance the analysis of decision making in requirements engineering. We previously discussed the problem of the acceptability of goal models [7] via their justification. The present work advances our prior results in several respects: (1) ACE allows participants to express preferences over information, and applications of inferences, conflicts, and other preferences in ACE graphs. Our prior analysis of acceptability via justification could not account for preferences, the set of inferences and conflict rules was closed (i.e., two inference rules were available and one conflict rule), and no forms of meta-reasoning could be captured (one could not express that the application of an inference rule is in conflict with the application of another inference rule, that a conflict may be in conflict with the application of an inference rule, and so on). (2) ACE offers algorithms for the retrieval of information relevant for the evaluation of acceptability; no such features were available for our justification graphs. (3) The evaluation of acceptability in ACE is automated via the evaluation algorithm outlined above; acceptability was evaluated manually in justification graphs. In conclusion, ACE is (i) more expressive (due to the presence of preferences and the support for forms of meta-reasoning), (ii) more “practical” (due to the presence of retrieval and evaluation algorithms), and (iii) more general (as argued throughout the paper) than the

goal-oriented approaches, which we suggested previously.

The language in ACE, and more precisely, the choice of the four labels – information, inference, conflict, and preference – was influenced by the initiative towards a core ontology for argumentation in artificial intelligence, within the Argument Interchange Format (AIF) [2]. AIF has recently been suggested to facilitate the representation and exchange of data between various tools and agent-based applications that rely on arguments. The basis for AIF is the AIF Argument Network, which is “the core ontology for argument entities and relations between argument entities” [2]. The notion of argument is a construct in AIF, defined as a particular subgraph in an argument network. It is by placing restrictions on, or by specializing the concepts in the argument network that classical argumentation frameworks can be obtained. ACE thereby reuses the core ontology of AIF for the labels in ACE graphs. To the best of our knowledge, no framework based on AIF and comparable to ACE in terms of the language and the retrieval and evaluation algorithms has been suggested.

Validation is an old problem in RE, and has been raised in particular in relation to the very early requirements elicited from the parties involved in RE. Leite and Freeman argued in an important paper [9] that requirements should be elicited from different viewpoints, and “that examination of the differences resulting from them can be used as a way of assisting in the early validation of requirements”. They suggested a language for capturing viewpoints, and heuristics for a syntactically oriented analysis of views to the aim of resolving their inconsistencies. This approach provides inputs to the negotiation required to reconcile different opinions. An attractive characteristic of their proposal is that it is a means of validation applicable very early in the elicitation of requirements. We are very close to their work in this motivation, although our proposal differs in several important respects. ACE has a considerably simpler language than their framework. Since automated reasoning about acceptability happens in a propositional framework in ACE, we can accommodate various forms of RE artifacts. As we treat propositions as atoms, Leite and Freeman’s approach can go into more detail, and study the structure of the propositional content. While they can point precisely the discrepancies between views, we can help by evaluating the outcomes of a discussion of these discrepancies. Nothing similar to the automated evaluation of acceptability is present in the viewpoint resolution method. Discussions are not studied in detail. ACE is complementary in this respect, as it can be used to record and evaluate discussions of views and their inconsistencies. Gervasi and Nuseibeh [5] check predefined properties on models generated from parsed text to identify nontrivial inconsistencies. While $\mathbf{AC}(I_D, T(I_D), O_D)$ indicates agreement about these artifacts, it certainly does not entail the internal consistency of the I_D , $T(I_D)$, and O_D ;

inconsistency can be present even if agreement is present: inconsistency in that case remains undetected, making ACE complementary to any approach tailored to detect nontrivial internal inconsistencies. Validation of late requirements is well illustrated in a recent paper from Uchitel et al. [12]. They establish the relation between scenarios and goals via “fluents that describe how the events of the operational description change the state of the basic propositions from which goals are expressed.” Graphical animations can be synthesized from scenarios and goal model checking over scenarios is enabled to guide animations through goal violation traces. Animations are subsequently presented to the relevant parties for discussion. ACE can complement such an approach, especially when user-centered sessions involve the asynchronous participation of geographically distributed users, so that discussion via forum-like tools becomes relevant. Boehm’s WinWin groupware supports negotiation of requirements via the general WinWin approach. It is usually defined as [1] “a set of principles, practices, and tools, which enable a set of interdependent stakeholders to work out a mutually satisfactory (winwin) set of shared commitments.” WinWin differs from ACE in that it focuses on the negotiation context, which differs from discussions on which ACE focuses. ACE is not tailored to negotiation.

Absence of validity can reflect errors in the rationale of the decisions taken when a method is applied. The primary aim of design rationale (DR) research is to capture the *why* behind decisions in a design activity. The classical IBIS method [3] starts with a participant who posts the root *issue* of an IBIS tree. Others then post *positions* (i.e., ways of resolving issues) and *arguments* (to support or object to positions). Issues, positions, and arguments are related via some of the allowed relationships: generalize, specialize, object, support, replace, question. The process stops when consensus has been reached regarding the resolution of issues. Subsequent DR methods, as reviewed by Louridas and Loucopoulos [10], share many characteristics with IBIS. The principal novelty of ACE with regards to these other DR methods lies in the way it answers the *relationship expressivity question*: What are the relationships between concepts in the DR method, and how are they defined? At stake in this question is how to build a DR approach in the face of the multitude of potentially relevant relationships; Louridas and Loucopoulos highlight this problem ([10]: p.222–223):

“A fixed set of links [i.e., relationships] limits both the expressive and the functional capabilities of the [design rationale] model. Regarding the expressive weaknesses of any such attempt, the sheer number of the proposed relationships in the various [design rationale] approaches and the differences in their semantics from approach to approach indicate that it is difficult to arrive at a widely accepted set of predefined links. Each approach commits to a certain set, but there is no reason to believe that one of these sets is innately better than the others.”

This leads Louridas and Loucopoulos to leave out the definitions of the relationships in their Reasoning Loop Model, which is their synthetic proposal for reflective design. In doing so, they adopt the so-called *free-link* approach to the construction of a DR methods. This differs from the classical *fixed-link* approach, where a closed set of relationships is defined, and their meaning set once and for all. The benefit of free-links approach is that it leaves considerable freedom in use; its main downside is that it is hard to define, even manual, techniques for the analysis of its graphs. The fixed-link approach allows for the definition of analysis techniques, although the tendency to use many relationships renders the definition of algorithms for the analysis of graphs difficult. ACE adopts a third option, which has not been explored elsewhere to the best of our knowledge. ACE has a single relationship, the meaning of which are derived from the labels of vertices that it connects. An inference, conflict, and preference label does not designate a specific inference rule, conflict rule, or preference rule: one inference vertex may capture the application of modus ponens, another the application of defensible inference; one preference vertex may capture the application of a transitive preference order, while another may capture the application of an intransitive preference. Finally, and very importantly, ACE allows one to accept or reject the application of an inference, conflict, and/or preference rule, which is clearly impossible in the fixed-links approach to the construction of DR methods. If links are fixed, no meta-reasoning on them is allowed by the DR method itself. Making the links free is no better solution: what are then the criteria to accept or reject an arbitrary link? ACE is interesting because it leaves the freedom in the actual choice of an inference, conflict, or preference, while at the same time fixing how an inference, conflict, or preference relates, in terms of acceptability, to another *i*, *I*, *C*, or *P* vertex. E.g., $i_1 \rightarrow I \rightarrow i_2$ tells us in terms of acceptability that i_1 does not make i_2 unacceptable, but is evidence to the acceptability of i_2 , and this regardless of the actual inference applied to conclude i_2 from i_1 . By writing $i_1 \rightarrow I \rightarrow i_2$, we may be abbreviating the expression “ i_1 supports i_2 ”, where “supports” has the same meaning as in IBIS. *I* in $i_1 \rightarrow I \rightarrow i_2$ is thus locally defined (as opposed to fixed-links approach), but we still have precise criteria for accepting or rejecting any local reading of *I* in $i_1 \rightarrow I \rightarrow i_2$ (as opposed to the absence of these criteria in the free-links approach): we will reject it if the evaluation algorithm labels it **R**. Overall, ACE shows a novel way to balance the tradeoff between freedom in use and the automated analysis of rationale, without falling into extremes of the usual fixed-links, or the recent free-links approach to the construction of DR methods.

6 Conclusions and Future Work

Perfectly valid RE artifacts capture *exactly* what the stakeholders *really* need. We distinguished this absolute validity from relative validity. The latter asks if the stakeholders agree on the content of an RE artifact, being thereby relative to the stakeholders. Checking relative validity inevitably leads to a discussion between the stakeholders and the requirements engineer. This paper offered the *acceptability condition* (§2–3.2) on an artifact as a proxy for relative validity, and the ACE framework for the evaluation of the acceptability condition via the analysis of discussions. If the acceptability condition holds, then this signals that the relative validity verifies for the given artifact and for the participants in a given discussion. The ACE framework incorporates a simple, but expressive language for the representation of the pieces of information exchanged in a discussion, and the inference, conflict, and preference relationships between these pieces of information. Discussions are represented via directed labeled graphs. We suggested an algorithm to retrieve subgraphs in order to inform the discussions between the participants. ACE incorporates another algorithm to evaluate the acceptability condition in these graphs. Data from actual use will expectedly open many questions regarding usability and relevance in practice.

References

- [1] B. Boehm, P. Grunbacher, and R. O. Briggs. Developing groupware for requirements negotiation: Lessons learned. *IEEE Software*, 2001.
- [2] C. Chesnevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *Knowl. Eng. Rev.*, 21(4):293–316, 2006.
- [3] J. Conklin and M. L. Begeman. gibus: a hypertext tool for exploratory policy discussion. *ACM Trans. Inf. Syst.*, 6(4):303–331, 1988.
- [4] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT FSE*, pages 179–190, 1996.
- [5] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. *Software—Practice & Exp.*, 32:113–133, 2002.
- [6] J. A. Goguen and C. Linde. Techniques for requirements elicitation. In *Proc. Int. Symp. Req. Eng.*, pages 152–164, 1993.
- [7] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Requir. Eng.*, 13(2):87–115, 2008.
- [8] I. J. Jureta, J. Mylopoulos, and S. Faulkner. Towards a theory of requirements elicitation: Acceptability condition for the relative validity of requirements, 2009. <http://arxiv.org/abs/0902.0924v1>.
- [9] J. C. S. P. Leite and P. A. Freeman. Requirements validation through viewpoint resolution. *IEEE T. Softw. Eng.*, 17(12):1253–1269, 1991.
- [10] P. Louridas and P. Loucopoulos. A generic model for reflective design. *ACM Trans. Softw. Eng. Methodol.*, 9(2):199–237, 2000.
- [11] M. McGrath. Propositions. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008.
- [12] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. Goal and scenario validation: a fluent combination. *Req. Eng.*, 11:123–137, 2006.
- [13] P. Zave. Classification of research efforts in requirements engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.