Consolidating Multiple Requirement Specifications through Argumentation

Ebrahim Bagheri
Athabasca University and National Research Council Canada

Faezeh Ensan
University of New Brunswick

ABSTRACT
The process of handling inconsistencies in software requirements is an important and challenging task. Most often in cases where multiple stakeholders interact with the requirement analysts, inconsistent, discrepant and conflicting information is gathered that needs to be understood and interpreted properly. Recent research has suggested that despite the fact that inconsistencies are not desirable by nature, they can be tolerated in order to better understand the nature of the problem domain and the stakeholders’ line of thought. With this in mind, we propose an argumentative approach towards handling inconsistent requirement specifications. In our semi-formal approach, we build on Dung’s abstract argumentation framework and represent requirement statements as arguments. This way we are able to model the interaction of the requirement statements in terms of their inconsistencies and also provide a decision support process for the resolution of inconsistencies. We discuss our approach in detail through a widely used case study and introduce our Eclipse plugin tool supporting the proposed work.

Keywords
inconsistency resolution, argumentation, requirement engineering

1. INTRODUCTION
Requirements analysis is one of the first stages in the systems engineering and software development processes [1]. It is concerned with understanding, determining and formalizing the needs and restrictions of an intended software system, which is often based on possibly conflicting requests and information provided by the various stakeholders of the product-to-be. Requirements analysis or as termed more broadly, requirement engineering, covers three main activities, namely 1) requirements elicitation by means of which the analysts will try to understand and gather the needs of the end users and stakeholders; 2) requirements compilation which brings all the gathered information together under a unified framework and ensures that they are complete, clear, non-ambiguous, and non-conflicting. Several authors have referred to such a requirements collection as being canonical [2, 3]; 3) requirements documentation that formulates the gathered information in an agreed upon formalism, which can be referred to in later software design activities.

Each of these three main activities has been the area of many fruitful studies [4, 5]. Here, we are interested in looking at the requirements compilation process. Clearly, as the complexity of the target domain for which the software system is being designed rises, the wealth and amount of collected requirements information also substantially increases. For this reason, requirements analysts have become interested in studying approaches that would allow them to understand, compile and consolidate information from multiple sources. The sources of requirements information ranging from legacy system documentation to the results of domain expert interviews are commonly known as viewpoints [6]. Finkelstein et al. define viewpoints as ‘loosely coupled, locally managed distributed objects which encapsulate partial knowledge about a system and its domain, specified in a particular, suitable representation scheme, and partial knowledge of the process of development [7].

In spite of the great advantages of a viewpoint-based requirement engineering process, there are several obstacles that impede the best performance of such an approach. First, since most often the sources of information are humans, and given the fact that human beings usually make conception errors due to causes such as risk aversion, short-term memory or even framing and perceptual problems, it can be perceived that not all of the information that has been gathered in the requirement elicitation process are correct or equally reliable. The second is related to the fact that even if we assume that all information sources are reliable and non-biased, still there is no guarantee that the gathered information are mutually consistent. Consider a situation where the areas of interest of two experts with different understanding and perception of the domain of discourse overlap. Here, it is likely that the expressed information by each of these parties is different, inconsistent or even conflicting.

In order to address the above-mentioned issues, it is important to consider and analyze them in close relation with the underlying formalism through which the requirement specifications of the viewpoints are expressed. A recent survey on industrial requirement engineering methods showed that a great portion (95%) of requirement specifications are written in common or semi-structured natural language, and only a very small fraction of requirement analysts employ...
some sort of formal requirement specification language or formalism [8]. This is corroborative of the fact that requirement engineering has essentially some flavor of social sciences work in which strict formal structures may fail to capture all of the desired details. Despite the fact that formal representation of requirements specifications can be very useful for their automated analysis such as finding inconsistencies, conflicts or ambiguities, it can be cumbersome for the requirements analysts to define and also interpret all of the requirements information in a formal language (logic), which is not native to humans.

1.1 Approach Overview

In our approach, we will suppose that software requirements have been gathered from different sources of information. Our assumption is that all of the information sources are equally reliable and information that they are providing is correct from their own perspective. The issue of reliability has been extensively studied in other work including [9, 10]. For cases in which sources of software requirements cannot be considered reliable, our proposed approach in this paper can be easily extended to provide support for explicit reliability evaluation. The equal-reliability assumption allows us to exclusively focus on the issue of inconsistency and conflict in requirement specifications. In our work, although information sources are reliable, the information that they are providing might be inconsistent with each other. This is perceivable due to the fact that each information source has his/her own perspective and limited knowledge of the domain of discourse. The assumption of reliability also permits us to view inconsistency as different reflections of knowledge about a specific aspect of the domain of discourse, and not necessarily wrong information.

In this context of thought, inconsistencies do not entail that among the discrepant set of requirements one is correct and the others are incorrect. We believe that rather than correctness, inconsistencies should be dealt with in light of their desirability. So, to resolve inconsistency the most desirable requirement needs to be selected and the rest of the conflicting ones be removed. It is also important to notice that desirability is subjective and depends on the stakeholders’ opinion; therefore, while one requirement is more desirable for one group of stakeholders and hence should be kept and the others be removed to resolve inconsistency, this may not be the case for another group of stakeholders. In our opinion, one of the major drawbacks of pure logical formalism for dealing with inconsistency in requirement specifications is that they attempt to identify and resolve inconsistency in a pure syntactic form (e.g., removing the formula with the most number of syntactic conflicts), failing to take note of the more important semantical information that is required to resolve inconsistency.

For this reason, we view handling inconsistency in software requirements as an argumentation process through which the reasons for and against the discrepant requirement specifications are given (and visualized), based on which the requirement analysts can select their most desired requirements. We base our work on Dung’s abstract argumentation framework [11]. Unlike other argumentation frameworks [12], the abstract argumentation framework does not require the details of the argumentative information to be represented in formal logic and only needs the argumentative relations between the arguments to be explicit. As was mentioned before, most requirement analysts document their requirements in natural language; therefore, the use of the abstract argumentation framework is reasonable because it does not need the requirement to be formally defined and only needs the relationships between the requirements to be defined. For instance suppose we have the following two requirements: Req. 1: Cheap cars also need to be equipped with a remote starter and Req. 2: Remote starters are expensive. As opposed to some other argumentation frameworks that formalize these two requirements as: Req. 1: Cheap Cars ∧ Remote Starter, Req. 2: Remote Starter → Expensive, and Cheap Cars ∧ Expensive = ⊥, showing the inconsistency through the third formula, Dung’s framework shows this by simply stating: Req. 2 ∼ Req. 1, depicting that these two requirements are inconsistent without going into the details of their inconsistency. This allows requirements analysts to use informal natural language formalisms to express the captured requirements and at the same time depict the possible discrepancies in a semi-formal manner.

The additional benefit of Dung’s abstract argumentation framework is that it provides suitable dialectical proof procedures for computing the preferred arguments in an argumentation structure [13]. We exploit these procedures to help requirements analysts in finding their most desirable set of requirements. In summary, the contributions of this paper are as follows:

- Requirement specifications are represented in terms of their interactions and mutual implications rather than their internal semantics. This approach has the advantage of not requiring the requirement analysts to understand a formal language for expressing the captured requirements, and at the same time allows them to address inconsistencies and discrepancies;
- A distinction between correctness and desirability has been made, which allows us to benefit from an argumentative approach for handling and resolving inconsistencies. The argumentative approach respects the fact that different viewpoints may have disagreements on similar aspects of the domain of discourse and attempts to explain how these disagreements link together, through dialectical proof procedures, for the requirement analysts to be able to make the most appropriate selection decision;

The process benefits from tool support that enhances decision making in face of inconsistencies in software requirement specifications.

2. OVERVIEW OF ARGUMENTATION

Argumentation is an important aspect of human cognition [12]. Humans are constantly faced with a stream of information that needs to be understood and responded to, among which conflicting and inconsistent information is not a rare commodity. Therefore, in order to reach a rational conclusion, the relevant assumptions and possible conclusions relating to the perceived information should be analyzed based on their pros and cons. This process might require a chain of reasoning, where arguments and counterarguments for a certain conclusion are evaluated. Decision making and its required argument building process is mostly performed semi-consciously in the daily activities. However in more complicated circumstances with many parameters to
consider, argumentation needs to be cogently performed in a more structured way. This would require formal tools for comparing arguments, evaluating them, and finally judging whether arguments are warranted or not [14].

Three main argumentation systems have been proposed in the literature that have given rise to many interesting argumentation methods and techniques: 1) abstract argumentation systems assume that argumentative information and their structure can be captured through interaction as modeled by a binary attacks relation between pairs of arguments; 2) defeasible argumentation systems are fundamentally based on the idea of incorporating defeasible implication in the formal language used for representing arguments. In these systems, arguments can be defined as chains of reasons against their counterarguments; 3) coherence argumentation systems adopt a reasonable approach towards reasoning with inconsistent body of knowledge, which is reasoning over coherent subsets of the information.

Unlike coherence and defeasible system that require all information to be represented in a formal language, e.g., classical logic, abstract systems only require the capture of the interaction between the arguments; therefore, for cases such as our problem, where the representation of the body of knowledge in a formal language is a very tough task, argumentation based on abstract systems seems to be a rational choice. We adopt and customize the abstract argumentation system originally proposed by Dung [11] for the purpose of handling inconsistencies in requirement specifications.

3. SOFTWARE REQUIREMENT SPECIFICATIONS AS ARGUMENTS

In this section, we define how requirements specifications can be represented in the abstract argumentation framework. Further we extend it to accommodate the required tasks for handling inconsistencies in requirements specifications.

**Definition 1.** A requirement constellation is a pair \((\mathcal{S}, \sim)\) where \(\mathcal{S}\) is a set of requirement specifications, and \(\sim\) is a binary relationship over \(\mathcal{S}\) such that \(\sim \subseteq \mathcal{S} \times \mathcal{S}\).

The set \(\mathcal{S}\) consists of all the gathered requirements statements from the viewpoints; therefore, \(\forall r_i \in \mathcal{S},\ r_i\) is a requirement statement. Furthermore, \(r_i\) disputes \(r_j\) iff \(r_i \sim r_j\). In the example given earlier, \(\text{Req. 2}\) disputes the information provided by \(\text{Req. 1}\) by arguing that remote starters are expensive and are hence not suitable for cheap cars; therefore, \(\text{Req. 2} \sim \text{Req. 1}\). This definition shows that regardless of the information in each requirement statement, a requirements constellation can be developed from the set of requirements specifications only based on the dispute relationships between them. A requirements constellation can be represented through a graph formalism.

**Definition 2.** A requirement constellation graph is an ordered pair \((\mathcal{S}, \sim)\) with requirement statements in the requirements specification set \(\mathcal{S}\) as its vertices and the binary relationship \(\sim\) as its arcs, which are irreflexive, antisymmetric, and nontransitive.

It is easy to see that an edgeless requirement constellation graph represents a completely consistent requirements specifications set. This is because an edgeless graph implies that none of the requirement statements disputes the information provided by the other statements and therefore, the set of requirements specifications \(\mathcal{S}\) is consistent \((\mathcal{S} \models \top)\). Furthermore, the three properties imposed on the disputes relationship \(\sim\) ensures that implicit implications that are not intended by the requirement statements are not derivable from the constellation.

**Example 1.** Let us suppose that three requirement statements have been provided as follows: 1) \(\text{Req. 1: Plug and play components should be developed because they are easy to sell}\), 2) \(\text{Req. 2: Plug and play components often fail to perform properly, and Req. 3: our experience shows that systems based on COTS are often reliable}\). Given these statements, it can be seen that \(\text{Req. 2}\) disputes \(\text{Req. 1}\) by saying that plug and play components do not work properly and hence people are not interested in buying them; therefore, we have \(\text{Req. 2} \sim \text{Req. 1}\). However, \(\text{Req. 1} \sim \text{Req. 2}\) does not hold since \(\text{Req. 1}\) is not providing any information that is inconsistent with \(\text{Req. 2}\). This shows that the disputes relation is intrinsically antisymmetric.

Furthermore, \(\text{Req. 3}\) disputes \(\text{Req. 2}\), and also \(\text{Req. 2}\) disputes \(\text{Req. 3}\), which depicts that while the disputes relation is not antisymmetric by nature, still some disputes might be symmetric; therefore, symmetric dispute of two requirements statements needs to be explicitly defined as: \(\text{Req. 3} \sim \text{Req. 2}\) and \(\text{Req. 2} \sim \text{Req. 3}\). Another important observation from this example is that although we have \(\text{Req. 3} \sim \text{Req. 2}\) and \(\text{Req. 2} \sim \text{Req. 1}\), it cannot be inferred that \(\text{Req. 3} \sim \text{Req. 1}\); hence, it is important to note that the disputes relation is nontransitive as well. Similar to cases of symmetry in the disputes relation, transitive disputes relations need to be explicitly defined. Also, the assumption is that a requirement statement does not dispute and contradict its own claim, therefore, disputes are irreflexive.

Now, assume that an additional requirement such as \(\text{Req. 4: Selling products more easily, isn’t our goal at the current time}\). This requirement disputes the underlying reason for \(\text{Req. 1}\); therefore, \(\text{Req. 4} \sim \text{Req. 1}\) and also \(\text{Req. 1} \sim \text{Req. 4}\). An important observation here is that if two requirements dispute one similar requirement, they are not necessarily expressing information related to the same issue. This can be seen by viewing \(\text{Req. 4} \sim \text{Req. 2}\) that both dispute \(\text{Req. 1}\), but are not addressing the same issue. Figure 1 shows the requirements constellation graph for these examples. Possible links that could have been inferred from the disputes relation that were not permitted to be automatically inferred due to the properties of the requirements constellation graph are shown with dashed line.

Now that requirements specifications are modeled through the concept of requirements constellation and the disputes relationship, it is important to reason over the develop structure in order to find the acceptable requirement statements.

![Figure 1: The requirements constellation graphs for the requirements in Example 1.](Image 353x688 to 520x738)
Trivially, if the requirements constellation is an edgeless graph then all the requirement statements are acceptable. However, if one or more disputes relationships exist, then the acceptable requirement statements need to be identified such that inconsistencies can be resolved. Let us extend the disputes relation from single requirement statements to requirement specification sets.

Definition 3. A requirement specification set \( S_s \subseteq S \) disputes a requirement statement \( r_i \in S \) if for some requirement statement such as \( r_j \in S_s \) then \( r_j \rightarrow r_i \).

Given this definition, it is possible to define acceptability.

Definition 4. A requirement statement \( r_i \in S \) is acceptable in the context of \( S_s \subseteq S \) iff for all requirement statements \( r_j \in S \) that \( r_j \rightarrow r_i \), then \( S_s \rightarrow r_j \). In this context, \( S_s \) justifies \( r_j \) against \( r_i \).

An interesting observation from this definition is that only inconsistency-free requirement statements, i.e., those that do not have any conflict with the other requirement statements in the requirements constellation are acceptable in the context of \( S \). The other requirement statements are simply acceptable by smaller subsets of \( S \).

Definition 5. A requirements specification set \( S_s \subseteq S \) is dispute-free iff no two requirement statements such as \( r_i, r_j \) can be found in \( S_s \) such that \( r_i \rightarrow r_j \).

A dispute-free requirements specification set is one that has no inconsistencies among its requirement statements. A dispute-free requirement specification set is one that is most desirable for the requirements analysts, and is also the most reliable to base later software design decision upon.

Example 2. In Example 1, the dispute-free requirement specification sets are \{Req. 1\}, \{Req. 2\}, \{Req. 3\}, \{Req. 4\}, \{Req. 1, Req. 3\}, \{Req. 3, Req. 4\}, \{Req. 2, Req. 4\}. There are no requirement specification sets of cardinality 3 or 4 that are dispute-free. Furthermore, in this example requirement statements are only acceptable by the dispute-free requirement specification sets where they are a member. For instance, Req. 4 is acceptable by \{Req. 3, Req. 4\}, \{Req. 2, Req. 4\}, and \{Req. 4\}.

Definition 6. A requirements specification set \( S_s \subseteq S \) is admissible iff \( S_s \) is dispute-free and justifies all of its constituent requirement statements.

In the above example, all of the dispute-free sets are also admissible. Given these sets, it is possible to select the one that is more desirable and base future design decisions on this consistent subset of the requirement specifications. For instance, one might choose \{Req. 1, Req. 3\}, and infer that plug and play components in the form of COTS need to be developed and employed because they are easy to sell and reliable.

Specifically while resolving inconsistency in a requirements constellation, we are interested in solutions that lose the least amount of information, i.e., it is important that the inconsistency resolution process preserves the information provided by the viewpoints as much as possible. Therefore, we employ the concept of preferred distillations of the requirements constellation.

Definition 7. A preferred distillation of a requirements constellation is a \( \subseteq \) -maximal admissible subset of its requirements specification set \( S \).

A preferred distillation of a requirements constellation only considers those admissible subsets of \( S \) that are maximal with regards to set inclusion. In other words, preferred distillations discard the least possible amount of requirements statements in order to achieve consistency. In the example, \{Req. 1, Req. 3\}, \{Req. 3, Req. 4\}, and \{Req. 2, Req. 4\} are the preferred distillations of the requirements constellation; therefore, these three subsets are the maximally preserving consistent subsets of \( S \) that are candidates for being employed as a consistent requirements specification set used in future design decision making processes.

Given the preferred distillations, the important issue is to find an appropriate mechanism to evaluate the relative fitness of each preferred distillation, and hence create a priority relationship over the preferred distillations. This priority relation will facilitate decision making with regards to choosing the most suitable from amongst the preferred distillations.

Definition 8. The fitness measure of a preferred distillation \( S_p \) is a function \( fm : \mathbb{S}_p \rightarrow \mathbb{R} \). Given two preferred distillations \( S_p^1 \) and \( S_p^2 \), \( S_p^1 \) is considered to have more or equal priority over \( S_p^2 \), denoted \( S_p^1 \geq S_p^2 \), if and only if \( fm(S_p^1) \geq fm(S_p^2) \).

According to Definition 8, if under some fitness function we have \{Req. 1, Req. 3\} \( \succ \) \{Req. 3, Req. 4\} \( \succ \) \{Req. 2, Req. 4\}, then \{Req. 1, Req. 3\} has the highest priority to be selected as the most preferred consistent requirements specification subset. Now we introduce the building blocks of the fitness measure.

Definition 9. Let \( r_i \) be a requirement statement in \( S \). The support for \( r_i \), denoted \( Sup(r_i) \), is defined as the cardinality of \( r_i \) in all preferred distillations of \( S \).

For instance in the preferred distillations above, \( Sup(Req. 1) = 1 \) and \( Sup(Req. 4) = 2 \). The support for a requirement statement shows how consistent the requirement statement is. If the support for a requirement statement is high, it shows that the requirement has appeared in more preferred distillations, which is an indication that it is consistent with a lot of different requirement statements in \( S \). It is desirable that such requirement statements are preserved in the final most preferred consistent requirements specification subset. It is possible to extend the notion of support to preferred distillations.

Definition 10. Let \( \psi = \{\psi_1, ..., \psi_p\} \) be the set of preferred distillations of \( S \), and \( r^k = \{r^k_1, ..., r^k_n\} \) be the set of requirement statements in \( \psi_k \). The support for \( \psi_k \) is defined as follows:

\[
Sup(\psi_k) = \frac{\sum_{i=1}^{n_k} Sup(r^k_i)}{\sum_{j=1}^{p} \sum_{i=1}^{n_j} Sup(r^j_i)}.
\]

The support for a preferred distillation is defined structurally similar to the degree of conflict in [3]. The support for \{Req. 3, Req. 4\} is \( Sup(Req. 3, Req. 4) = \frac{1}{0.4} = 0.4 \).

Simply stated, the support for preferred distillations structurally analyzes the requirements constellation and evaluates the value of each preferred distillation by measuring how
Figure 2: A requirements specification constellation augmented with relevant concerns information.

generally consistent its constituent requirement statements are. In other words, the concept of support captures to what extent the information provided by one preferred distillation is corroborated by the other preferred distillations. Intuitively, support allows us to form the fitness function based on the structure of the requirements constellation, i.e., it will provide means for finding the maximally consistent preferred distillation; however, a fitness function based only on support may fail to capture the intentions of the viewpoints completely. To illustrate this point consider a scenario where after the analysis of the requirements constellation, two preferred distillations have been extracted. By looking at these two we see that the requirement statements in the first favor aspects related to the reliability of the system-to-be and its implementation costs, while the latter focuses just on performance.

To address this issue, Sommerville and Sawyer have introduced the notion of concerns in their viewpoint-based requirement engineering framework called Preview [6]. Concerns are high-level strategic goals and aspects of the intended software entity such as reliability, performance, safety, fault tolerance, etc. that need to be closely observed in the development of the requirement specifications. Concerns crosscut viewpoints and can be addressed by any of the viewpoints. So, it is important to see how individual requirement crosscut viewpoints and can be addressed by any of the viewpoints and must be seen that the resonance for each preferred distillation respects the important viewpoints’ goals through the concept of resonance. It is important to notice that the assumption is that all concerns are equally important; however, in cases where concerns have different importances, the scheme proposed below can be extended as investigated in [9, 10].

**Definition 12.** Let \( r_i \) be a requirement statement in \( \mathcal{S} \). The resonance of \( r_i \), denoted \( \text{Res}(r_i) \), is defined as number of concerns that \( r_i \) respects. Further, let \( \varphi = \{\varphi_1, ..., \varphi_p\} \) be the set of preferred distillations of \( \mathcal{S} \), and \( r^* = \{r^*_1, ..., r^*_n\} \) be the set of requirement statements in \( \varphi_k \). The resonance of \( \varphi_k \) is defined as follows:

\[
\text{Res}(\varphi_k) = \frac{\sum_{i=1}^{n_{\varphi_k}} \text{Res}(r^*_i)}{\sum_{j=1}^{p} \sum_{i=1}^{n_{\varphi_j}} \text{Res}(r^*_i)}.
\]

Since concerns depict the important goals of the viewpoints and the stakeholders, Res measures how well a preferred distillation resonates with the viewpoints, or in other words, how likely it is for the stakeholders to find the information in that preferred distillation relevant and important. Continuing on the information provided in Example 3, it can be seen that the resonance for \( \{\text{Req. 3, Req. 4}\} \) is \( \text{Res}(\{\text{Req. 3, Req. 4}\}) = \frac{4}{4+3} = 0.36 \).

Now, we have two measures, namely support and resonance that evaluate the fitness of a preferred distillation from both structural-syntactic and conceptual-semantic perspectives. A hybrid combination of these two measures will enable us to create the desired fitness measure to evaluate the suitability of any preferred distillation.

**Definition 13.** (extends Definition 8). The fitness measure of a preferred distillation \( \mathcal{S}_p \) is a function \( f_m : \mathcal{S}_p \rightarrow [0, \sqrt{2}] \) such that:

\[
f_m(\mathcal{S}_p) = \sqrt{\text{Sup}(\mathcal{S}_p) \times \text{Res}(\mathcal{S}_p)}
\]

where \( f_m \) is the length of a two dimensional vector whose dimensions are the support and resonance of \( \mathcal{S}_p \).

This fitness measure can facilitate the rank-ordering of the preferred distillations and hence help in choosing a more relevant (both structurally and conceptually) consistent subset of the original inconsistent requirements specifications.

**Example 4.** Given the augmented requirement constellation in Example 3, it is possible to calculate both support and resonance for the preferred distillations of the requirements specifications, which would be \( \text{Sup}(\{\text{Req. 1, Req. 3}\}) = 0.3, \text{Sup}(\{\text{Req. 3, Req. 4}\}) = 0.4, \text{Sup}(\{\text{Req. 2, Req. 4}\}) = 0.3, \text{Res}(\{\text{Req. 1, Req. 3}\}) = 0.36, \text{Res}(\{\text{Req. 3, Req. 4}\}) = 0.36 \).
3. Req. 4)) = 0.36, Res((Req. 2, Req. 4)) = 0.27. Therefore, by creating a two dimensional vector from these two properties the fitness measure of each of these preferred distillations can be computed: \( fm\{(\text{Req. 1, Req. 3})\} = 0.46, \)
\( fm\{(\text{Req. 3, Req. 4})\} = 0.53, \) \( fm\{(\text{Req. 2, Req. 4})\} = 0.4; \)
therefore, we would have the following order relationships between the preferred distillations that can be used to select the most relevant maximally preserving consistent subset of the requirements specifications: \( \{\text{Req. 3, Req. 4}\} \succ \{\text{Req. 1, Req. 3}\} \succ \{\text{Req. 2, Req. 4}\}. \) Given this outcome, the best consistent interpretation of the requirements specifications would be:

Experience shows that systems based on COTS are often reliable and can be used for development. Also, selling products more easily isn’t the primary goal at the current stage of the project. Figure 3 shows the two dimensional representation of the fitness vector of the preferred distillations.

As it can be seen, the fittest preferred distillation can be considered as possibly the most suitable consistent subset of the requirements specifications for making future software design decisions.

4. A CASE STUDY

In this section, we will borrow a case study scenario from [15] to show a step-by-step process of the argumentation process over an inconsistent requirements constellation, and show how inconsistencies are handled in its context. This scenario is also used in other relevant papers such as [3, 2] to describe how each of their proposed techniques handle inconsistency in requirement constellations. The scenario studies a subset of a residential management system from three viewpoints. We will consider three main concerns in this scenario which are Safety, Security, and Responsiveness. It is possible that any of the requirement statements given below are related to these concerns:

- **Viewpoint 1 (Vehicle Entrance Manager):**
  - (a) The vehicles without special authorization are not allowed to enter the residential area - Safety;
  - (b) The system should trigger a warning alarm if an unauthorized vehicle enters the area - Security;

- **Viewpoint 2 (Emergency Manager):**
  - (c) The fire engine should be viewed as a special vehicle for emergency - Safety;

- **Viewpoint 3 (Authorization Manager):**
  - (d) Special vehicles of emergency can enter the residential area - Safety, Responsiveness, Security;
  - (e) Special vehicles of emergency do not need to be pre-authorized - Safety, Responsiveness;
  - (f) The system should trigger an alarm when any vehicle without pre-authorization enters the area - Security, Responsiveness;
  - (g) The fire engine should be considered a special kind of vehicle - Safety;
  - (h) Special vehicles can enter the area - Safety, Responsiveness, Security;
  - (i) Special vehicles do not need to be pre-authorized - Safety, Responsiveness;
  - (j) The system should not trigger an alarm for special vehicles - Security, Responsiveness

The concerns related to each requirement statement are shown besides each of the requirements. In addition, by looking at the requirement statements that each of the viewpoints have provided, it can be seen that requirements \{ (a), (f) \}, \{ (g), (c) \}, \{ (d), (h) \}, and \{ (i), (e) \} are similar; therefore, we can easily remove one of the similar requirement statements after which the requirement specification set would be: \{ (a), (b), (c), (d), (e), (j) \}.

Now, it is possible to create the requirements constellation graph based on the disputes relationships that exist between the requirement statements. The requirements constellation graph is shown in Figure 4. An interesting observation from this graph is that requirement (c) neither disputes nor is disputed by any other requirement statement; therefore, it will be present in all possible preferred distillations.

The next step after developing the requirements constellation graph is to derive the preferred distillations. This can be done by performing the process introduced in [13]. The preferred distillation derivation process will entail two preferred distillations: \( D_1=\{\text{(e), (d), (c), (j)}\}, \) and \( D_2=\{\text{(a), (c), (b)}\}. \)

In order to be able to select from these two preferred distillations, their degree of support and resonance are calculated that are the building blocks of the fitness measure. The support and resonance for each of the preferred distillations is as follows: \( \text{Sup}(D_1)=\frac{5}{5+4}=0.55, \) \( \text{Sup}(D_2)=\frac{1}{5+4}=0.45, \)
\( \text{Res}(D_1)=\frac{4}{5+4}=0.73, \) and \( \text{Res}(D_2)=\frac{5}{5+4}=0.27. \) Given these values, the fitness measure of the preferred distillations, which is the length of a two-dimensional vector formed based on support and resonance, can be calculated: \( fm(D_1) = \sqrt{\text{Sup}^2(D_1) + \text{Res}^2(D_1)} = 0.91, \) and \( fm(D_2) = \sqrt{\text{Sup}^2(D_2) + \text{Res}^2(D_2)} = 0.524; \) therefore, \( D_1 \succ D_2. \)
The interpretation of this is that although both of the preferred distillations are consistent subsets of the original requirements constellations, the preferred distillation developed in \( D_1 \) has a higher chance of being desired by the viewpoints. As a result the compilation of the requirements in \( D_1 \) can be understood as: The fire engine should be viewed as a special vehicle for emergency. Special vehicles of emergency can enter the residential area. Special vehicles of emergency do not need to be pre-authorized. The system should not trigger an alarm for special vehicles., which is more preferred than that expressed by \( D_2 \) as The vehicles without special authorization are not allowed to enter the residential area. The system should trigger a warning alarm if an unauthorized vehicle enters the area. The fire engine should be viewed as a special vehicle for emergency.

Although our argumentation framework provides a ranking and shows more preferences towards \( D_1 \) compared to \( D_2 \), the ordering of the preferred distillations can only serve as a decision support process and not a decision making algorithm due to the very subjective nature of the problem.

5. DISCUSSIONS

There have been various approaches towards resolving and handling inconsistencies in software requirements ranging from informal techniques to quite strict and formal methods [2, 3, 16, 17, 7].

In the informal inconsistency resolution methods, since the analysts are not equipped with supportive tools and methods to resolve inconsistency, the final outcome of inconsistency handling and resolution might not be the most desirable. Also since the process is informal, the negotiation process for understanding the sources of inconsistency and resolving them may take a long time. However, the requirement analysts and the stakeholders are often more comfortable with informal approaches due to the fact that they do not require extra formulation of the requirements or the learning of some kind of formal language/logic. While informal approaches trade performance for ease of use, formal approaches benefit from the formalization of requirement statements in an accepted logical representations and hence try to carefully resolve inconsistency in requirement statements. For instance, in previous line of work, the authors benefited from annotated Propositional logic to create a belief integration game that gradually resolve inconsistencies in software models and requirements [3].

Proposals such as [9, 3] and others alike [2, 16, 15] are rooted mainly in belief revision and inconsistency resolution in formal logics. While they are theoretically sound, they lack two main characteristics. First, it is usually hard for the requirement analysts to define all of their requirements information in logic-based formalizations. Second, although the formal inconsistency resolution methods resolve discrepancies in knowledge bases, they do not necessarily preserve the most desirable information, i.e., they will remove some of the information to retain consistency based on some syntactical criteria; however, in requirement engineering, the subjective point of view of the stakeholders and the requirement analysts is very important in the decision making process. For these reasons, in our approach we try to strike a balance between formal and informal approaches and their benefits. Our work allows the requirement analysts to define their information in their preferred way, and only specifies the interaction of these requirements in a semi-formal way using the disputes relationship. The benefit of this approach is that it does not enforce the analysts to learn and use a new formalization for representing their requirements, and provides them with a simple to use mechanism to identify and resolve inconsistency. The other advantage of our approach is that it does not take initiative in resolving inconsistency based on syntactical conflict information. It only identifies the inconsistencies and equips the requirement analysts with her possible options towards handling and resolving inconsistency in requirement specifications by providing and rank ordering these options. In this way, the final decision is based on the discretion of the requirement analysts.

Easterbrook distinguishes three broad strategies for resolving requirement specification conflicts, namely: Cooperative method, which includes negotiation; Competitive method, which comprises of coercion and competition; and Third-party method, which consists of arbitration and appeal to higher level authorities [18]. Many of the work proposed in the literature benefit from some form of these strategies. A major advantage of our proposed approach is that the involved participants can utilize all of these three strategies at the appropriate time: Cooperation and competition are facilitated in our model through the use of the disputes relationship. So, in order for two viewpoints to compete they can dispute the requirement statements of the other. Similarly, for being cooperative they can provide information or additional requirement statements that dispute those requirements that dispute the other viewpoints requirement statements. In other words, given two viewpoints \( vp_1 \) and \( vp_2 \), they can compete over some requirement statements described as \( ∃ r_1 \in vp_1, ∃ r_2 \in vp_2 : r_2 ⇝ r_1 \). On the same basis, cooperation between two viewpoints such as \( vp_1 \) and \( vp_2 \) is defined as \( ∃ r_1 \in vp_1, ∃ r_2 \in vp_2, ∃ r_i \in vp_i : r_i ⇝ r_1 \). Now if \( ∀ r_1 \in vp_1, ∃ r_2 \in vp_2 : r_2 ⇝ r_1 \), then the viewpoints are completely inconsistent and fully competitive. Similarly, if \( ∀ r_1 \in vp_1, ∃ r_2 \in vp_2, ∃ r_i \in vp_i : r_i ⇝ r_1 \), then \( r_i ⇝ r_1 \) then \( r_i ⇝ r_1 \), then the viewpoints are completely cooperative and supportive. In real-world cases viewpoints are competitive on some of the requirement statements and cooperative on the others, which is why the resolution of inconsistencies is possible (e.g. through concession and trade-off [3]). Finally, our approach supports for the third-party intervention for inconsistency resolution by only providing recommendations and decision support such that a third party can benefit from them and make the final inconsistency resolution strategy decision.

In order to provide the requirement analysts with a tool to perform the proposed technique for handling inconsistent requirement specifications, we have provided support for representing requirement statements in Eclipse shown in Figure 5. The developed tool allows the requirement analysts to carry out the following tasks: 1) Define and edit their requirement specifications in natural language and formulate the interaction between the requirements using the disputes relationship; 2) Compute the preferred distillations of the requirement constellations, which are the maximally preserving consistent subsets of the originally inconsistent requirement specification set; 3) Rank-order and compare the possible preferred distillations based on their strategic goals, i.e. concerns, and the syntactical properties of the requirements specifications; 4) Choose the appropriate preferred distillation and export it as the requirement specification set that is going to be used in further design work.
6. CONCLUDING REMARKS

Requirement specifications can be viewed as the foundations of a software development process. The quality of software requirements has direct effect on the final software product that is ultimately developed; therefore, it is very important to ensure that software design decisions are based on a consistent set of requirements. In large complex systems, requirements inconsistencies and conflicts are often sourced from the disagreement between the participating stakeholders and differences in their opinions.

To provide a solution to this issue in requirement engineering, our proposed approach in this paper contributes in the following ways: 1) It provides a semi-formal mechanism for representing the interactions between the available requirement statements in such a way that the requirement analysts are not obliged to learn and use a new complicated formalism to represent their requirements, but at the same time be able to depict and understand the possible inconsistencies in a structured way; 2) Our approach acknowledges the difference between correctness and desirability in software requirements and views the source of inconsistency in differences of subjective judgement. This approach to inconsistency prevents from making judgements about the correctness of the requirements, instead it focuses on finding the degree of desirability of the requirements. For this purpose, our approach benefits from an argumentative approach that allows inconsistencies to be resolved through argumentation. Furthermore, it only serves as decision support through which the requirement analysts have the decision making authority. the argumentation process only provides rank-ordering preference relations over the preferred distillations, which can be used by the requirement analysts to make final decisions; 3) The process is supported by an Eclipse plugin that allows requirement analysts to easily represent the interaction of the requirement statements, performs reasoning on the requirements and provides recommendations on how to handle inconsistencies to the analysts.

7. REFERENCES