A Measurement-Driven Process Model For Managing Inconsistent Software Requirements

Kedian Mu
School of Mathematical Sciences
Peking University, Beijing 100871, P.R.China

Zhi Jin
School of Electronics Engineering and Computer Science
Peking University, Beijing 100871, P.R.China

Didar Zowghi
Department of Software Engineering
University of Technology, Sydney, NSW2007, Australia

Abstract

Inconsistency is a pervasive issue in software engineering. Both general rules of inconsistency management and special case-based approaches to handling inconsistency have recently been considered. In this paper, we present a process model for handling requirements inconsistency within the Viewpoints framework. In this process model, when an inconsistency among viewpoints is detected, a set of candidate proposals for handling inconsistency will be generated using techniques from Multi-agent automated negotiations. The proposals are then prioritized using an integrated measurement of inconsistencies. The viewpoints involved in the inconsistency will then enter the negotiations by being presented with the candidate proposals and thus selecting an acceptable proposal based on the priorities associated with each candidate proposal. To facilitate usability, in our process, we assume that the natural language requirements statements are first translated into corresponding logical formulas using a translator software. Moreover, the candidate proposals for handling inconsistency are also translated back from formal login into natural language before being presented to the analyst for selection.

Keywords: *Inconsistency handling, automated negotiation, measure of inconsistency.*

1. Introduction

Support for evolutionary processes is needed at all stages of software development but especially during Requirements Engineering (RE), since it is the requirements modifications that typically triggers other changes in the development life cycle. Requirements often evolve because stakeholders cannot possibly envision all the ways in which a system can be utilized. The environment where the software is situated frequently changes and so do the software boundaries and business rules governing the utilization of that software. Correspondingly, designs change because the requirements change. Implementation has to be changed because designs evolve and defects have to be fixed.

Software Requirements Specification (SRS) plays an important role in the development process because it provides a baseline for the development of all the subsequent artifacts such as design, code and test cases. Eliciting and documenting high quality requirements as well as an effective and efficient management of requirements process are considered crucial for software development projects. This statement is supported by claims in the literature that ineffective requirements management is identified as a major source of problems [9] and that errors made during requirements stage account for 40-60% of all defects found in software projects [13]. It is generally accepted that it is much more expensive to correct these types of errors if they are left undetected and leaked into the subsequent phases of software development projects.

One of the main classes of defects in requirements specifications is that of inconsistency. Inconsistency occurs when a specification contains conflicting, contradictory descriptions of the expected behavior of the system. Such contradictory descriptions could be attributed to conflicting goals between the various participants that contribute to the development of the specification, namely the stakeholders. Inconsistencies could also result from uncoordinated changes

introduced in the specification during the usual evolution of the requirements.

Inconsistency is considered as a serious problem and a major source of risk that permeates all aspects of software development. It makes it very hard to design and implement a system that fully satisfies its specification. In some cases, the programmer may arbitrarily resolve inconsistencies during the implementation of the system. This means that one of the conflicting requirements will be preferred over the others, usually without performing an in-depth analysis of the consequences, and without notifying the relevant stakeholders. In worst-case scenarios, undetected inconsistency may lead to incorrect and unreliable systems, whose faults are only discovered after deployment when it is too late.

In requirements elicitation, it is often advocated to consult all relevant stakeholders so that each can express his/her own needs for the system under construction. In viewpoint-based approaches to RE [2, 5, 11], it is recognized that all the necessary requirements information cannot be elicited by viewing the system from a single perspective and that requirements have to be gathered and organized from a number of different viewpoints. A viewpoint is an encapsulation (from a certain perspective), of partial information about a system's requirements. Information from different viewpoints then has to be integrated into the final product of RE, the SRS.

Inconsistency has been recognized as one of the important issues in the Viewpoints framework[4, 8]. Handling inconsistency arising in overlapped viewpoints is not just a technical issue. It is also associated with a process of negotiation over actions of inconsistency handling between related stakeholders. For each stakeholder, many factors such as his expected benefits from the system-to-be, relative value of each requirement, cost of implementation, and subjective preference and so on will have influence on his trade-off decision on inconsistency handling. It is not easy to persuade stakeholders to change their needs that have been expressed. Thus, the final proposal could often become an unsuccessful compromise between stakeholders. Moreover, lack of appropriate principles for evaluating negotiation proposals prevents participants from reaching reasonable agreement.

If a proposal for inconsistency handling is acceptable to a particular viewpoint, then acceptance of the proposal should intuitively result in decreasing the degree and significance of inconsistency in overlapped viewpoints from the perspective of the viewpoint. Previously [14], we have argued that the relative importance of requirements with regard to the viewpoint always affects the evaluation of significance of inconsistent viewpoints. Therefore, the appropriate principles for evaluating proposals of negotiation over inconsistent viewpoints should also take the priority of requirements statement into account.

Generally speaking, inconsistency management may be divided into two parts, i.e. consistency checking and inconsistency handling. Consistency checking focuses on techniques for detecting inconsistencies in a set of requirements. Typical techniques include logic-based approaches [7, 8, 22] and consistency-rule-based approaches [15, 17]. For logic-based approaches, the term *inconsistency* is defined as any situation in which some fact and its negation can be simultaneously derived from the same requirements collection. That is, inconsistency is referred to as logical contradiction. Detecting this type of inconsistency entails that the SRS should be represented in formal notation such as logic. Writing and analyzing a specification in formal logic requires very specialized knowledge and expertise. Often, such expertise is not readily available, and this contributes to the limited use of formal specification languages in software industry. Translation between formal and informal languages is thus needed if we are to benefit from the power of formal logic in management of inconsistencies in requirements specifications and make automated inconsistency management accessible to requirements engineers.

"Inconsistency Implies Actions" is recognized as a meta-rule for inconsistency handling [6, 8]. That is, when inconsistencies are detected, some action should be performed to manage these inconsistencies. However, identifying appropriate actions still remains a difficult, but important challenge [8] in software development. The choice of an inconsistency handling action always depends on the nature and context of these inconsistencies [6, 4]. Thus it is difficult to provide a universal approach to handling all types of inconsistencies in requirements engineering.

In this paper we build upon our previous work [14] on an integrated measure-driven approach to handling inconsistency in Viewpoints framework. Firstly, we present our measure-driven framework for handling inconsistency among multiple viewpoints using an integrated priority relations among requirements that combines the notions of *degree* and *significance* of inconsistencies. When an inconsistency among viewpoints is detected, a set of candidate proposals for handling this inconsistency will be generated using techniques from Multi-agent automated negotiations. The viewpoints involved in the inconsistency will then enter the negotiation by being presented with the candidate proposals and thus selecting an acceptable proposal based on the priorities of each candidate proposal.

In our recognition of the need for a software tool that can fully automate the process of detection and handling of requirements inconsistencies in an evolutionary viewpoints framework, and motivated by our desire to make this tool accessible to any requirements engineer, we thus propose a process model for the management of requirements inconsistencies. This process model will also include the details of generating a set of candidate proposals for handling

inconsistency and identifying an acceptable common proposal. Our proposed process model will envisage the use of a translator application (e.g. Carl in [7]), that would allow requirements engineers interact in natural language with an automated inconsistency handler application.

This paper is organized as follows. Section 2 presents a measurement-driven process model for managing requirements inconsistencies. Section 3 gives some preliminaries for formulating this process model. Section 4 describes components of the process model. Section 5 gives a case study to illustrate the use of the process model during the requirements stage. Section 6 compares our process model and related works. Finally, we conclude the paper in section 7.

2. A Process Model for Handling Requirements Inconsistencies

We propose here a measurement-driven process for handling requirements inconsistencies, as shown in fig. 1.

The input and the output of this process model are requirements in natural language with priority and the consistent requirements specification respectively. The activities in this process model are as follows:

- requirement translation: translating the requirements with priority in natural language submitted by the stakeholders into requirements in logic with priority
- specification revision: revise the requirement set into an ordered requirement specification
- inconsistence management: the inconsistence manager contains two parts. One is the detector and the other is the measurer. The detector is designed for detecting the inconsistency in requirements specification. Once there are inconsistent requirements statements, it produces the minimum inconsistent set and triggers the measurement of the inconsistent degree. After measurement, an integrated inconsistence degree will be obtained. If no inconsistence can be detected, a consistent requirement specification can be obtained and returned to the stakeholders
- negotiation: the negotiation mechanism is designed for generating the proposals from each individual viewpoint for resolving the inconsistences in the minimum inconsistence set. And a group decision maker is also provided for getting a sorted list of proposals by adopting some priority-based strategies.
- proposal translation: translating the proposals into natural langauge sentences so that the stakehoders can understand them and then can make their choices by their own preference.

Among these activities, the inconsistence management and the negotiation are two key activities in this process. The other activities are just included for communicating with the stakeholders. They will not within the scope of this paper.

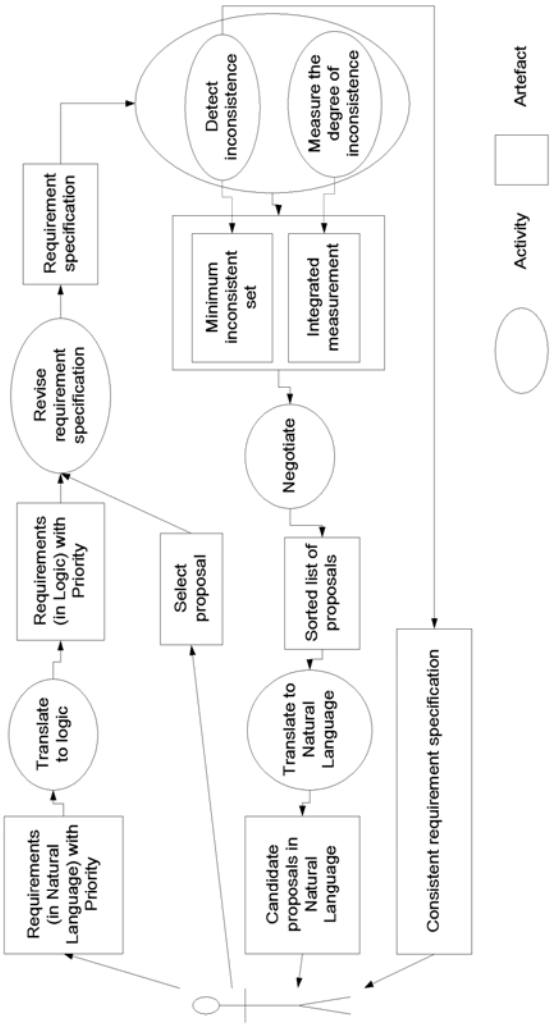

Figure 1. A Process Model for Handling Requirements Inconsistencies

3. Preliminaries for Formulating the Process

The principle of heterogeneity of representation allows different viewpoints use different notations and tools to represent their requirements during the requirements stage [16]. However, classical logic is appealing for formal representation of requirements [8]. Moreover, in logic-based framework for representing requirements, reasoning about requirements is always based on some facts that describe a certain scenario [8]. That im-

plies that checking consistency of requirements collections considers ground formulas rather than unground formulas. These reasons make the classical logic the most convenient to illustrate our approach, as will be shown in the rest of the paper.

Let $\mathcal{L}_{\Phi_0}$ be the language composed from a set of classical atoms Φ_0 and logical connectives $\{\vee, \wedge, \neg, \rightarrow\}$ and let $\vdash$ be the classical consequence relation. Let $\alpha \in \mathcal{L}_{\Phi_0}$ be a classical formula and $\Delta \subseteq \mathcal{L}_{\Phi_0}$ a set of formulas in $\mathcal{L}_{\Phi_0}$. In this paper, we call Δ *a set of requirements* while each formula $\alpha \in \Delta$ represents a requirements statement.

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of viewpoints. For each i $(1 \leq i \leq n)$, S_i is a set of requirements of v_i.

Generally, the requirements specification could be represented by a n tuple $\langle \Delta_1, \cdots, \Delta_n \rangle$.

The term of *inconsistency* has different definitions in requirements engineering [22]. Most logic-based works such as [8, 22, 7] concentrated on a particular kind of inconsistency, i.e. *the logical contradiction*: any situation in which some fact α and its negation $\neg\alpha$ can be simultaneously derived from the same requirements collection. In this paper, we shall be also concerned with the logical contradiction. If there is a formula α such that $\Delta \vdash \alpha$ and $\Delta \vdash \neg\alpha$, then we consider Δ to be *inconsistent* and abbreviate $\alpha \wedge \neg\alpha$ by $\perp$.

It has been recognized that the relative priority of requirements can help stakeholders resolve conflicts and make some necessary trade-off decisions [20, 3]. A common approach to prioritizing requirements specification is to group requirements into several priority categories, such as the most frequent three-level scale of {*High*, *Medium*, *Low*} [20] and the IEEE recommended scale of {*Essential*, *Conditional*, *Optional*} [10].

Let m, a natural number, be the scale of the priority and L be $\{l_0, \cdots, l_{m-1}\}$, a totally ordered finite set of m symbolic values of the priorities, i.e. $l_i < l_j$ iff $i < j$. Furthermore, each symbolic value in L could associate with a linguistic value. For example, we have a totally ordered set L as $L = \{l_0, l_1, l_2\}$ where l_0 : *Low*, l_1 : *Medium*, l_2 : *High*. For example, if we assign l_2 to a requirements statement α, it means that α is one of the most important requirements statements. Prioritization over Δ_i is in essence to establish a prioritization function $P_i : \Delta_i \rightarrow L$ by balancing the business value of each requirements statement against its cost and technique risk. Then in the *Viewpoints* framework, the requirements specification with preference may be represented by $\langle (\Delta_1, P_1), \cdots, (\Delta_n, P_n) \rangle$, where P_i is the prioritizing function of viewpoint v_i.

We consider each individual viewpoint as a negotiation participant during the process of inconsistency handling. And assume that they are negotiating over a finite set Ω of possible modifications of requirements (which is referred to as outcomes in [21]).

Negotiation proceeds in a series of rounds, where at each round, every participant presents a proposal which is a subset of Ω. And it is assumed that any outcome of the proposal is acceptable to the offerer. As we use the classical logic-based negotiation language $\mathcal{L}$ proposed in [21], the proposal a participant makes at each round is a formula of $\mathcal{L}$. Then each round could be described by a tuple $\langle \varphi_1, \cdots, \varphi_n \rangle$, where φ_i is the proposal presented by participant i [21]. A negotiation history is a finite sequence of rounds. Let φ_i^k be the proposal made by participant i in the kth round, then a negotiation history could be viewed as the following matrix of formulas of $\mathcal{L}$ [21]:

$$\begin{array}{cccc} \varphi_1^0 & \varphi_1^1 & \cdots & \varphi_1^m \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_n^0 & \varphi_n^1 & \cdots & \varphi_n^m. \end{array}$$

In this paper, we use the combination of the two above agreement conditions mentioned in [21], that is, the agreement will be reached if and only if $(\varphi_1^m \wedge \cdots \wedge \varphi_n^m) \wedge (\varphi_1^m \Leftrightarrow \cdots \Leftrightarrow \varphi_n^m)$ is satisfiable. This agreement condition guarantees that if an agreement is reached in negotiation over inconsistency handling, then all the viewpoints involved will undertake a feasible global proposal of inconsistency handling and they should agree on the meanings of the proposal.

4. Components of Tool Set

We discuss components of tool set for supporting application of the process model to requirements engineering practice. A translator, an inconsistency manager, and a negotiation controller may be considered as the most important components of the process. As mentioned earlier, the process model would allow requirements engineers interact in natural language with an automated inconsistency handler. Consequently, a translator for mapping requirements in natural language into logic formulas (e.g. Carl in [7]) would be built in the process. In this paper, we focus on the inconsistency manager and the negotiation controller.

4.1. Inconsistency manager

The inconsistency manager consists of an inconsistency detector and an inconsistency measurer.

Inconsistency detector. An inconsistency detector is designed to detect inconsistencies in a set of requirements. Generally, most logic-based approaches [8, 22, 7] design some paraconsistent logics to represent the requirements, moreover, reasoning engine is considered as inconsistency detector. In contrast, consistency rule bases plays an important role in consistency rule-based approaches. In this paper, the term of inconsistency is defined as *logical contradictions*. Consequently, inconsistency detector should be a type of paraconsistent reasoning engine.

Inconsistency Measurer. Inconsistency measurer plays an important role during the process for handling inconsistent requirements. The degree and the significance of inconsistency are two essential aspects for articulating an individual inconsistent requirements set. The degree of inconsistency describes how inconsistent a requirements collection is, whilst the significance of inconsistency measures how important the inconsistency of a requirements set is. Then they may be viewed as two distinct dimensions of measurement of an inconsistency.

- **Degree of inconsistency**. Suppose that Δ is a set of requirements. Let

$$\mathsf{MI}(\Delta) = \{\Phi \subseteq \Delta | \Phi \vdash \bot, \forall \phi \in \Phi, \Phi - \{\phi\} \nvdash \bot\},$$
$$\mathsf{CORE}(\Delta) = \bigcup_{\Phi \in \mathsf{MI}(\Delta)} \Phi.$$

  Note that $\mathsf{MI}(\Delta)$ is the set of *minimal inconsistent subsets* of Δ; and $\mathsf{CORE}(\Delta)$ is the union of all its minimal inconsistent subsets of Δ. For any set of requirements, we consider the union of all its minimal inconsistent subsets as the collection of all the *problematical* requirements.

  Hunter's approach [1] to measuring the degree of inconsistency of a formulas set captures contribution of each subset to the inconsistency. Given Δ, a scoring function S is defined from $\mathcal{P}(\Delta)$ (the power set of Δ) into the natural numbers so that for any $\Gamma \in \mathcal{P}(\Delta)$, $S(\Gamma)$ gives the number of minimal inconsistent subsets of Δ that would be eliminated if the subset Γ was removed from Δ [1]. That is, for $\Gamma \subseteq \Delta$,

$$S(\Gamma) = |\mathsf{MI}(\Delta)| - |\mathsf{MI}(\Delta - \Gamma)|.$$

  As such, sets of formulas could be compared using their scoring functions so that an ordering relation, which means "*more inconsistent than*", over these sets can be defined [1]. That is, assume that Γ_1 and Γ_2 are of the same cardinality, S_1 is the scoring function for Γ_1, and S_2 the scoring function for Γ_2. $S_1 \leq S_2$ holds iff there is a bijection $f : \mathcal{P}(\Gamma_1) \mapsto \mathcal{P}(\Gamma_2)$ such that the following condition can be satisfied:

$$\forall \Theta \in \mathcal{P}(\Gamma_1),\ S_1(\Theta) \leq S_2(f(\Theta))$$

  We say Γ_2 is *more inconsistent than* Γ_1 iff $S_1 \leq S_2$.

- **Significance of inconsistency**. Intuitively, the significance of an inconsistency depends on the relative importance of requirements statements involved in the inconsistency. In our previous paper [14], we provided the following intuitive assumptions about significance of a requirements set in a given viewpoint:
  - the requirements statements with the same priority have the same significance;
  - any requirements statement with a higher priority is more significant than all of those with lower priorities;
  - those requirements statements with higher priorities play dominant roles in measuring the significance of a requirements set.

  Then given a requirements set Γ in viewpoint v_i, we use the priority-based cardinality vector, denoted $\overrightarrow{C}_i(\Gamma)$, to measure the significane of Γ with regard to v_i, where $\overrightarrow{C}_i(\Gamma) = (|\Gamma^0|, \cdots, |\Gamma^{m-1}|)$, and for each k, $\Gamma^k = \{\gamma \in \Gamma, P_i(\gamma) = l_k\}$ [14]. Informally, vector $\overrightarrow{C}_i(\Gamma)$ provides the number of requirements with each priority level as well as the preferences of v_i. In this sense, it gives a measure of priority-based significance of Δ from the perspective of v_i. Moreover, in a given viewpoint, the sets of requirements of the viewpoints could be compared using the lexicographical vector ordering relationship $\preceq_P$ provided in [14].

  Based on the cardinality vector ordering, we can compare two minimal inconsistent subsets Φ_1 and Φ_2 in the sense of significance with regard to v_i using their cardinality vectors. If $\overrightarrow{C}_i(\Phi_1) \preceq_P \overrightarrow{C}_i(\Phi_2)$, then Φ_2 is more important(or significant) than Φ_1 with regard to v_i.

4.2. Negotiation Controller

Based on the measurement of a given inconsistency, each viewpoint involved in the inconsistency need to provide a proposal for handling the inconsistency from its own perspective. Also, all the viewpoints involved in the inconsistency need to reach an agreement on the final proposal for handling the inconsistency. Measurement of inconsistency provides support for putting forward a desirable proposal for handling inconsistency as well as reaching a successful compromise among viewpoints. Negotiation controller will simulate the behavior of viewpoints(or stakeholders) during this stage. It consists of *proposal generators* and *an automated group decision maker*.

- **Proposal Generators.** For each viewpoint, the corresponding proposal generator is responsible for providing appropriate proposals from its own perspective during the negotiation process. Just for convenience, we focus our attention on negotiation between two viewpoints in this paper. The negotiation among multiple viewpoints could be discussed based on negotiation in multi-agents system in the way similar to that presented in this paper. Let $\Delta_{i,j} = \Delta_i \cup \Delta_j, 1 \leq i, j \leq n, i \neq j$. If $\Delta_{i,j}$ is inconsistent, then v_i and v_j are trying to reach an agreement on the modifications of

$\mathsf{CORE}(\Delta_{i,j})$. For each individual viewpoint, we consider two kinds of atomic actions appropriate for modifying $\mathsf{CORE}(\Delta_{i,j})$:

- $delete(v_i, \alpha, \Delta_i)$: v_i deletes the requirement α from Δ_i;
- $add(v_i, (\beta, l_k), \Delta_i)$: If neither β nor $\neg\beta$ appears in Δ_i, then v_i add the requirements statement β with priority of l_k to Δ_i.

A proposal is a series of such actions under $\wedge$. In particular, we abbreviate $delete(v_i, \alpha, \Delta_i) \wedge add(v_i, (\gamma, l_t), \Delta_i)$ by $replace(v_i, \alpha, \Delta_i, (\gamma, l_t))$. A desirable proposal should be able to decrease the degree of inconsistency or decrease the significance of inconsistency. Moreover, change of the requirements with lower priorities should be considered firstly. Generally, if viewpoint v_i wants to put forward a proposal φ_i^k at k-th step, the following strategies should be considered:

- the degree of inconsistency in the union of the viewpoints should be decreased;
- from the perspective of v_i, a desirable proposal should help to eliminate the most significant minimal inconsistent subsets of the current requirements set;
- change of the requirements with lower priorities should be considered firstly.

However, these proposals presented by different viewpoints reflect different concerns and intentions. Viewpoints need to identify a common acceptable proposal by proceeding negotiation. It is a process of group decision making.

- **Group Decision Maker.** When a proposal for handling inconsistency is presented by a viewpoint, the negotiation for a common acceptable proposal start. For proceeding the negotiation process, we may use a negotiation meta-language defined in [21]. This language is richer for taking about proposals than negotiation languages designed for special scenarios [21], since it includes the following illocutions for describing the speech acts of conveying intentions:
  - $request(i, j, \varphi)$: a request from agent i to agent j for proposal φ;
  - $offer(i, j, \varphi)$: a proposal of φ from agent i to agent j;
  - $accept(i, j, \varphi)$: agent i accepts proposal φ made by agent j;
  - $reject(i, j, \varphi)$: agent i rejects proposal φ made by agent j;
  - $withdraw(i, j)$: agent i withdraws from negotiation with agent j.

  Here φ is a formula of the negotiation language. Generally, a negotiation begins when one agent makes an $offer$ to another, or when one makes a $request$ to another. Negotiation ceases when one agent $accept$ an $offer$ or $withdraw$ from negotiation [21].

Viewpoint v_i says	Viewpoint v_j replies
$request(v_i, v_j, \varphi_i^k)$	$offer(v_j, v_i, \varphi_j^k)$
$offer(v_i, v_j, \varphi_i^k)$	$offer(v_j, v_i, \varphi_j^k)$, or $accept(v_j, v_i, \varphi_j^k)$, or $reject(v_j, v_i, \varphi_j^k)$, or $withdraw(v_j, v_i)$
$reject(v_i, v_j, \varphi_i^{k-1})$	$offer(v_j, v_i, \varphi_j^k)$, or $withdraw(v_j, v_i)$
$accept(v_i, v_j, \varphi_i^{k-1})$	end of negotiation
$withdraw(v_i, v_j)$	end of negotiation

Table 1. The protocol at the k-th step of the negotiation

We adopt the protocol used in [18] in this paper, which is list in Table 1. It has been shown that the protocol guarantees success in [21].

To respond to proposal φ_i^k presented by v_i, viewpoint v_j need to evaluate the proposal. The following strategies should be taken into account:

- If there are some *essential* requirements of Δ_j involved in φ_i^k, then v_j may reject the proposal φ_i^k, or offer a new proposal to v_i, or withdraw from negotiation with v_i. Generally, viewpoint v_j cannot accept the proposal.
- For v_j, if the most significant minimal inconsistent subsets of the current requirements set will be eliminated by performing the proposal, and the importance of the set of requirements involved in φ_i^k is less than that of the set of others requirements, then v_j may accept the proposal φ_i^k, or offer an adaptation of φ_i^k to v_i in general case.

Based on these strategies, all the proposals may be sorted by each viewpoint from its own perspective. Let $MI(\Delta) = \{\Phi_1, \cdots, \Phi_n\}$. And assume that $\overrightarrow{C}_i(\Phi_1) \preceq_P \cdots \preceq_P \overrightarrow{C}_i(\Phi_n)$. Given a proposal φ, $u(\varphi) = (a_1, \cdots, a_n)$, where $a_i = 1$ if φ can eliminate Φ_i and $a_i = 0$ otherwise.

Definition 1 (Ordering relation over proposals) *Given two proposals φ_1 and φ_2, $\varphi_1 \leq \varphi_2$ iff $u(\varphi_1) \preceq_P u(\varphi_1)$. We say that viewpoint v_i prefers φ_2 to φ_1.*

However, these priority-based strategies do not provide an universal solution. Rather, the priority-based strategy emphasizes that the priority of requirements should play an important role in making or evaluating a proposal, since it is viewed as some kind of abstraction of the statement's significance.

5. Case Study

We give a small example to illustrate the use of the process model to handling inconsistency in software requirements development.

Example 1 *Consider the requirements for updating a software system. Stakeholder A, who will be in charge of sale of system-to-be, provides three demands as follows:*

- *Essential requirements: The system-to-be should be open (*OPEN*) , that is, the system-to-be could be extended easily;*
- *Conditional requirements: The user interface of system-to-be should be fashionably designed(*FASH*);*
- *Optional requirements: The system-to-be should be developed based on the newest development techniques (*NEWT*).*

Stakeholder B, who is a delegate of the users of the existing system, provides the following requirements:

- *Essential requirements: The system-to-be should be developed based on the techniques used in the existing system;*
- *Essential requirements: The user interface of system-to-be should keep the style of that of existing system;*
- *Optional requirements: The system-to-be should not be open for the sake of security.*

Let $\langle(\Delta_A, P_A), (\Delta_B, P_B)\rangle$ *denote the partial specification comprising the two viewpoints* $\{v_A, v_B\}$*, where* $\Delta_A = \{\text{OPEN}, \text{FASH}, \text{NEWT}\}$, $\Delta_B = \{\neg\text{OPEN}, \neg\text{FASH}, \neg\text{NEWT}\}$, *and* $P_A(\text{OPEN}) = l_2, P_A(\text{FASH}) = l_1,\ P_A(\text{NEWT}) = l_0$*;*

$P_B(\neg\text{NEWT}) = l_2;\ P_B(\neg\text{OPEN}) = l_0, P_B(\neg\text{FASH}) = l_2.$

Then $\Delta_{A,B} \vdash \bot$*, and* $MI(\Delta_{A,B}) = \{\Gamma_1, \Gamma_2, \Gamma_3\}$, $CORE(\Delta_{A,B}) = \bigcup_{i=1}^{3} \Gamma_i$,

$\Gamma_1 = \{\text{OPEN}, \neg\text{OPEN}\}$, $\Gamma_2 = \{\text{FASH}, \neg\text{FASH}\}$, $\Gamma_3 = \{\text{NEWT}, \neg\text{NEWT}\}$, $\overrightarrow{C}_A(CORE(\Delta_{A,B}) \cap \Delta_A) = (1,1,1)$, $\overrightarrow{C}_B(CORE(\Delta_{A,B}) \cap \Delta_B) = (1,0,2)$.

Let φ_i^k *be a proposal offered by* v_i *at k-th step and* $\Delta_l^{\varphi_i^k}$*(*$l = A, B, \{A, B\}$*) the modification of* Δ_l *by accepting* φ_i^k*. Since* ¬FASH *and* ¬NEWT *are the requirements statement with highest priority in* v_B *involved in inconsistency, at the beginning of negotiation,* v_B *puts forward a proposal* φ_B^0 *as follows:*

$$offer(v_B, v_A, (delete(v_A, \text{OPEN}, \Delta_A) \wedge delete(v_A, \text{FASH}, \Delta_A) \wedge delete(v_A, \text{NEWT}, \Delta_A)));$$

For viewpoint v_A*,* $P_A(open) = l_2$*, that is, open is essential to viewpoint* v_A*, then* v_A *rejects the proposal* φ_B^0*:* $reject(v_A, v_B, \varphi_B^0)$*. According*

to the strategies mentioned above, then v_B *puts forward the proposal* φ_B^1 *as follows:*

$$offer(v_B, v_A, (delete(v_B, \neg\text{OPEN}, \Delta_B) \wedge replace(v_A, \text{NEWT}, \Delta_A, (\neg\text{NEWT}, l_0))));$$

If v_A *accepts the proposal* φ_B^1*, then* Γ_1 *and* Γ_3 *disappear, and* $CORE(\Delta_{A,B}^{\varphi_B^1}) = \Gamma_2$*,*

$$\overrightarrow{C}_A(\Delta_A \cup \Gamma_2) = (0,1,0) \preceq_P \overrightarrow{C}_A(\Gamma_1 \cup \Delta_A),\quad \overrightarrow{C}_A(\{\text{NEWT}\}) \prec_P \overrightarrow{C}_A(\{\text{OPEN}\}).$$

Then v_A *may accept* φ_B^1*:* $accept(v_A, v_B, \varphi_B^1)$*. The first negotiation ends. And we get a modification of viewpoints as follows:*

$$\Delta_A^{\varphi_B^1} = \{\text{OPEN}, \text{FASH}, \neg\text{NEWT}\},$$
$$P_A^{\varphi_B^1}(\text{OPEN}) = l_2, P_A^{\varphi_B^1}(\text{FASH}) = l_1,\ P_A^{\varphi_B^1}(\neg\text{NEWT}) = l_0;$$
$$\Delta_B^{\varphi_B^1} = \{\neg\text{FASH}, \neg\text{NEWT}\}, CORE(\Delta_{A,B}^{\varphi_B^1}) = \Gamma_2.$$
$$P_B^{\varphi_B^1}(\neg\text{FASH}) = l_2, P_B^{\varphi_B^1}(\neg\text{NEWT}) = l_2.$$

However, $\overrightarrow{C}_2(\Gamma_2 \cup \Delta_2) = (0,0,1), \overrightarrow{C}_1(\Gamma_2 \cup \Delta_1) = (0,1,0)$*. The second negotiation begins when viewpoint* v_B *puts forward a proposal* φ_B^2 *as follows:*

$$offer(v_B, v_A, (delete(v_A, \text{FASH}, \Delta_A^{\varphi_B^1})));$$

If viewpoint v_A *accepts it:* $accept(v_A, v_B, \varphi_B^2)$*. The second negotiation ends. And we get a consistent modification of viewpoints as follows:*

$$\Delta_A^{\varphi_B^2} = \{\text{OPEN}, \neg\text{NEWT}\},$$
$$P_A^{\varphi_B^2}(\text{OPEN}) = l_2, P_A^{\varphi_B^2}(\neg\text{NEWT}) = l_0;$$
$$\Delta_B^{\varphi_B^2} = \{\neg\text{FASH}, \neg\text{NEWT}\},$$
$$P_B^{\varphi_B^2}(\neg\text{FASH}) = l_2, P_B^{\varphi_B^2}(\neg\text{NEWT}) = l_2.$$

6. Related Works

Inconsistency handling is a pervasive issue during the software development. A similar framework for managing inconsistency has been proposed in [15, 17]. Both the framework in [15, 17] and the process model presented in this paper emphasize the role of measurement of inconsistency in managing inconsistency. However, consistency rule rather than logical reasoning engine is considered as a central role to the framework presented in [15, 17]. In contrast,the process model presented in this paper considers logical reasoning engine as inconsistency detector. A detail rather than general approach to measuring requirements inconsistency is also considered in the process model. Moreover, the process model introduces the automated negotiation mechanism to resolve disagreement among different stakeholders in identifying final modification of requirements. In particular, the significance and the degree of inconsistency play a crucial role in putting forward or evaluating a proposal.

On the other hand, identifying appropriate actions for handling inconsistency is still a difficult issue in inconsistency management [8, 6, 4]. We do not claim that negotiation should be considered as a silver bullet for identifying an

appropriate proposal for inconsistency handling. For example, when a negotiation round ends with a withdraw, some other strategies such as combinatorial vote [12] and game theory should be considered.

7. Conclusions

We have presented a measure-driven process model for handling requirements inconsistency in the *Viewpoints* framework. In this process model, developers first translate the requirements in natural language into corresponding logical formulas by a translator such as developed in [7]. And an appropriate logical reasoning engine serves as inconsistency detector. If some inconsistencies are detected, then the inconsistent requirements set is measured in terms of the significance as well as the degree of inconsistency. Then negotiation mechanism from Multi-Agents is introduced to find an agreement among different viewpoints on inconsistency handling. In particular, the measurement of inconsistency is crucial to reaching a successful compromise during the negotiation.

The approach presented in this paper is an innovative integration of negotiation with the priority-based inconsistency handling for viewpoints conflicts. This work presents a first attempt in overcoming one of the weaknesses of the viewpoints framework, namely "limited support for requirements negotiation" as identified by Sommerville and Sawyer [19].

References

[1] A.Hunter. Logical comparison of inconsistent perspectives using scoring functions. *Knowledge and Information Systems Journal*, 6(5):528–543, 2004.

[2] J. Andrade, J. Ares, R. Garcia, J. Pazos, S. Rodriguez, and A. Silva. A methodological framework for viewpoint-oriented conceptual modeling. *IEEE Trans. Softw. Eng.*, 30(5):282–294, 2004.

[3] A. Davis. *Just Enough Requirements Management:Where Software Development Meets Marking*. Dorset House, 2005.

[4] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Trans. on Software Engineering*, 20:569–578, 1994.

[5] A. Finkelstein, J.Kramer, B.Nuseibeh, L.Finkelstein, and M.Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58, 1992.

[6] D. Gabbay and A. Hunter. Making inconsistency respectable 2:meta-level handling of inconsistent data. In *ECSQARU93,LNCS*, volume 747, pages 129–136. Springer, 1993.

[7] V. Gervasi and D.Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Transaction on Software Engineering and Methodologies*, 14(3):277–330, 2005.

[8] A. Hunter and B.Nuseibeh. Managing inconsistent specification. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367, 1998.

[9] M. Ibanez. *European user survey analysis. Tech. rep. ESI report TR95104.* European Software Institute, Zamudio, Spain. Web site: www.esi.es, 1996.

[10] IEEE.1998. *IEEE Std 830-1998:IEEE Recommended Practice for Software Requirements Specifications*. Los Alamitos, CA:IEEE Computer Society Press, 1998.

[11] G. Kotonya and I.Sommerville. Viewpoints for requirements definition. *IEE Software Eng.Journal*, 7:375–387, November 1992.

[12] J. Lang. From logical preference representation to combinatorial vote. In *Proceedings of 8th International Conference on Principles of Knowledge Representation and Reasoning*, pages 277–288. Morgan Kaufmann, 2002.

[13] D. Leffingwell. Calculating the return on investment from more effective requirements management. *American Programmer*, 10(4), 1997.

[14] K. Mu, Z. Jin, R. Lu, and W. Liu. Measuring inconsistency in requirements specifications. In L.Godo, editor, *ECSQARU2005,LNCS*, volume 3571, pages 440–451. Springer-Verlag Berlin Heidelberg, 2005.

[15] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. *IEEE Computer*, 33(4):24–29, 2000.

[16] B. Nuseibeh, J. Kramer, and A. Finkelstein. Viewpoints: meaningful relationships are difficult! In *Proceedings of the 25th International Conference on Software Engineering*, pages 676–681. IEEE CS Press, 2003.

[17] B. Nuseibeh, S.Easterbrook, and A.Russo. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2):171–180, 2001.

[18] C. Sierra, N. Jennings, P. Noriega, and S. Parsons. A framework for argumentation-based negotiation. In *Intelligent Agents IV (LNAI Volume 1365)*, pages 177–192. 1998.

[19] I. Sommerville and P. Sawyer. Viewpoints: Principles, problems and a practical approach to requirements engineering. *Ann. Software Eng.*, 3:101–130, 1997.

[20] K. Wiegers. First things first:prioritizing requirements. *Software Development*, 7(9):48–53, 1999.

[21] M. Wooldridge and S. Parsons. Languages for negotiation. In *Proceedings of ECAI2000)*, pages 393–397. 2000.

[22] D. Zowghi and V. Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993–1009, 2003.