Identifying Acceptable Common Proposals for Handling Inconsistent Software Requirements

Kedian Mu[1] and Zhi Jin[2]

[1] School of Mathematical Sciences
Peking University, Beijing 100871, P.R. China
[2] Academy of Mathematics and System Sciences
Chinese Academy of Sciences, Beijing 100080, P.R. China

Abstract. The requirements specifications of complex systems are increasingly developed in a distributed fashion. It makes inconsistency management necessary during the requirements stage. However, identifying appropriate inconsistency handling proposals is still an important challenge. In particular, for inconsistencies involving many different stakeholders with different concerns, it is difficult to reach an agreement on inconsistency handling. To address this, this paper presents a vote-based approach to choosing acceptable common proposals for handling inconsistency. This approach focuses on the inconsistency in requirements that results from conflicting intentions of stakeholders. Informally speaking, we consider each distinct stakeholder (or a distributed artifact) involved in the inconsistency as a voter. Then we transform identification of an acceptable common proposal into a problem of combinatorial vote. Based on each stakeholder's preferences on the set of proposals, an acceptable common proposal is identified in an automated way according to a given social vote rule.

1 Introduction

It is widely recognized that inconsistency management is one of the important issues in requirements engineering. For any complex software system, the development of requirements typically involves many different stakeholders with different concerns. Then the requirements specifications are increasingly developed in a distributed fashion, Viewpoints-based approaches [1,2,3] being a notable example. It makes inconsistency management necessary during the requirements stage. Generally speaking, inconsistency management may be divided into two parts, i.e. consistency checking and inconsistency handling. Consistency checking is a pervasive issue in requirements validation and verification. It focuses on techniques for detecting inconsistencies in a collection of requirements, including logic-based approaches [4,5,6] and consistency rule-based approaches [7,8]. In contrast, inconsistency handling focuses on how to identify an appropriate proposal for handling given inconsistencies and to evaluate the impact it has on other aspects of requirements stage [7,8].

Identifying appropriate inconsistency handling actions is still a difficult, but important challenge [5]. Generally, the choice of an inconsistency-handling action

J. Derrick and J. Vain (Eds.): FORTE 2007, LNCS 4574, pp. 296–308, 2007.

should depend on the nature and context of these inconsistencies [9,10]. But the context of the inconsistencies in requirements is always rather complex. Many factors such as misunderstanding between customers and analysts, inappropriate statements of requirements, and conflicting intentions of stakeholders can cause inconsistencies during requirements stage. It is not easy to provide a universal methodology to handle all the inconsistencies in requirements engineering.

In this paper, we concentrate on a particular kind of inconsistency that results from conflicting intentions of stakedholers. We would assume that there is no shortcoming in the ways that developers elicit and restate the requirements. That is, the inconsistent requirements are correctly elicited, stated, and represented from the perspective of corresponding stakeholders. There is no cause other than conflicting intentions of different stakeholders for the inconsistency. When an inconsistency resulting from conflicting intentions of stakeholders are detected, the different stakeholders involved in the inconsistency often present different proposals for handling the inconsistency from their own perspectives. These proposals reflect different concerns and intentions, and it is difficult to reach agreement on choice of proposals. Then the final proposal for handling the inconsistency is often an unsuccessful compromise among these different stakeholders. However, for this kind of inconsistency handling , there are two key problems associated with the identification of acceptable common proposals. One is how to evaluate an individual proposal from the perspective of each distinct stakeholder. That is, each stakeholder involved in the inconsistency needs to express his/her preferences on a set of proposals. It will provide a basis for identifying an acceptable common proposal. Another one is how to identify an acceptable common proposal from the set of these different proposals. Clearly, the latter problem is concerned with the mechanism of choosing the proposals such as negotiation and vote.

To address this, we present a combinatorial vote-based approach to identifying an acceptable common proposal for handling the inconsistency resulting from conflicting intentions of stakeholders in Viewpoints framework [1] in this paper. Combinatorial vote is located within the larger class of group decision making problems. Each one of a set of voters initially expresses his/her preferences on a set of candidates, these preferences are then aggregated so as to identify an acceptable common candidate in an automated way [11]. Informally speaking, for the inconsistency resulting from conflicting intentions of viewpoints (or stakeholders), we transform a set of proposals into a set of candidates with combinatorial structure. Then we consider each distinct viewpoint (or stakeholder) as a voter with different preferences on the set of candidates. Then an acceptable common proposal will be identified according some social vote rules in an automated way.

The rest of this paper is organized as follows. Section 2 gives some preliminaries about inconsistency handling in Viewpoints framework. Section 3 presents the combinatorial vote-based approach to identifying an acceptable common proposal for handling inconsistency. Section 4 gives some comparison and discussion about the vote-based approach. Finally, we conclude this paper in section 5.

2 Preliminaries

2.1 Viewpoints

The *Viewpoints* approach [1] has been developed to provide a framework in which the different perspectives and their relationships could be represented and analyzed. Viewpoint-oriented approaches to requirements engineering have been used for requirements elicitation [2], modeling [3], validation[12], and elaboration [13]. In the Viewpoints framework, a viewpoint is a description of system-to-be from the perspective of a particular stakeholder, or a group of stakeholders. It reflects the concerns of a particular stakeholder. The requirements specification of the system-to-be comprises a structured collection of loosely coupled, locally managed, distributable viewpoints, with explicit relationships between them to represent their overlaps [14].

The Viewpoints may allow different viewpoints use different notations and tools to represent their requirements during the requirements stage. However, the first order predicate calculus is appealing for formal representation of requirements statements since most tools and notations for representing requirements could be translated into formulas of the first order predicate calculus [5]. That is, predicate calculus may be considered as a promising tool to represent requirements from multiple sources. Moreover, we focus on the inconsistency handling rather than inconsistency checking in this paper. Then we need not consider reasoning with inconsistency in this paper.[1] For these reasons, we use the predicate calculus to illustrate our approach in this paper.

Let $\mathcal{L}$ be a first order language and let $\vdash$ be the consequence relation in the predicate calculus. Let $\alpha \in \mathcal{L}$ be a well-formed formula and $\Delta \subseteq \mathcal{L}$ a set of formulas in $\mathcal{L}$. In this paper, we call Δ *a set of requirements statements* or *a partial requirements specification* while each formula $\alpha \in \Delta$ represents a requirements statement.

As mentioned earlier, in this paper, we are concerned with the problem of handling inconsistency that involves multiple viewpoints. We would assume that $\mathsf{V} = \{v_1, \cdots, v_n\}(n \geq 2)$ is the set of distinct viewpoints. Let Δ_i be the set of requirements of viewpoint v_i. Then the partial requirements specification is represented by a n tuple $< \Delta_1, \cdots, \Delta_n >$. For any $\Gamma_i \subseteq \Delta_i (1 \leq i \leq n)$, we call $\bigcup_i \Gamma_i$ an integrated requirements collection, which could be viewed as a combination of requirements of multiple viewpoints. For example $\Gamma_1 \cup \Gamma_2$ and $\Gamma_1 \cup \Gamma_3 \cup \Gamma_n$ are two integrated requirements collections.

Further, for each $i (1 \leq i \leq n)$, G_i denotes the goal of viewpoint v_i. Intuitively, for each i, if the set of requirements Δ_i is sound with regard to v_i, then $\Delta_i \vdash G_i$. Generally, we call $< G_1, \cdots, G_n >$ a goal base.

2.2 Inconsistency in Viewpoints

The term of *inconsistency* has different definitions in requirements engineering [6]. In this paper we will be concerned with the logical contradiction: any

[1] This assumption is just for convenience. If not, we may use a paraconsistent adaptation of predicate calculus, such as Annotated Predicate Calculus [15], to represent requirements statements [16].

situation in which some fact α and its negation $\neg\alpha$ can be simultaneously derived from the same requirements collection [4]. Moreover, we focus on the inconsistency arising from multiple viewpoints.

Definition 1 (Inconsistency). *Let* $< \Delta_1, \cdots, \Delta_n >$ *be the requirements specification comprising n viewpoints. Let* Δ *be an integrated requirements collection. If there exists a formula* α *such that* $\Delta \vdash \alpha$ *and* $\Delta \vdash \neg\alpha$*, then* Δ *is inconsistent; otherwise,* Δ *is consistent. We abbreviate* $\alpha \wedge \neg\alpha$ *by* $\perp$*, which we read as "inconsistency". Further, if* $\Delta = \bigcup_{j=1}^{k} \Gamma_{i_j} (1 \leq i_j \leq n)$*, then we say that viewpoints* $v_{i_1}, \cdots, v_{i_k}$ *are involved in the inconsistency.*

If Δ is inconsistent, then Δ may be partitioned into two collections. One is the set of requirements statements being free from inconsistency, and another is the set of requirements statements involved in inconsistency. Actions for handling inconsistencies are always concerned with the set of requirements statements involved in inconsistency. Let

$$\begin{aligned} \mathsf{INC}(\Delta) &= \{\Gamma \subseteq \Delta | \Gamma \vdash \perp\}, \\ \mathsf{MI}(\Delta) &= \{\Phi \in \mathsf{INC}(\Delta) | \forall \Psi \in \mathsf{INC}(\Delta), \Psi \not\subset \Phi\}, \\ \mathsf{CORE}(\Delta) &= \bigcup_{\Phi \in \mathsf{MI}(\Delta)} \Phi, \\ \mathsf{FREE}(\Delta) &= \Delta - \mathsf{CORE}(\Delta). \end{aligned}$$

Essentially, $\mathsf{INC}(\Delta)$ is the set of inconsistent subsets of Δ; $\mathsf{MI}(\Delta)$ is the set of minimal inconsistent subsets of Δ; $\mathsf{CORE}(\Delta)$ is the union of all minimal subsets of Δ; and $\mathsf{FREE}(\Delta)$ is the set of requirements that don't appear in any minimal inconsistent subset of Δ, that is, it is a set of requirements statements being free from inconsistency. In contrast, $\mathsf{CORE}(\Delta)$ could be considered as a collection of all the requirements statements involved in inconsistency of Δ. It is this set that is of concern in the inconsistency handling.

Now we give an example to illustrate these concepts.

Example 1. Consider the following setting in development of residential area management system, which deals with the maintenance of fixed garages for vehicles. *Alice*, a manager who is in charge of maintenance, supplies the following demands:

- *The damaged garages should be maintained;*
- *An individual free garage, Garage* 1 *is damaged.*

Alice's goal is

- *Garage* 1 *should be maintained.*

Bob, a manager who is in charge of distribution of garages, gives the following demands:

– *Each garage on the expiration of utilization should be routinely maintained;*
– *All the free garages should not be maintained;*
– *Another individual garage, Garage* 2 *is on the expiration.*

Bob's goal is

– *Garage* 2 *should be maintained.*

Then the requirements of viewpoint v_A, denoted Δ_A, is

$$\Delta_A = \{\ (\forall x)(Damaged(x) \rightarrow Maintain(x)),\\ Damaged(Garage\ 1) \wedge Free(Garage\ 1)\}.$$

The requirements of viewpoint v_B, denoted Δ_B, is

$$\Delta_B = \{(\forall x)(Expire(x) \rightarrow Maintain(x)),\\ (\forall x)(Free(x) \rightarrow \neg Maintain(x)),\\ Expire(Garage\ 2)\}.$$

The goal base $< G_A, G_B >$is

$$< \{Maintain(Garage\ 1)\}, \{Maintain(Garage\ 2)\} > .$$

Let $\Delta = \Delta_A \cup \Delta_B$.We can conclude that

$$\Delta_A \vdash G_A,\\ \Delta_B \vdash G_B,\\ \Delta \vdash Maintain(Garage\ 1) \wedge \neg Maintain(Garage\ 1).$$

Then Δ is inconsistent. In this case,

$$\mathsf{MI}(\Delta) = \{\Phi_1\}\\ \mathsf{FREE}(\Delta) = \{(\forall x)(Expire(x) \rightarrow Maintain(x)),\\ Expire(Garage\ 2)\},\\ \mathsf{CORE}(\Delta) = \Phi_1, \text{ where}\\ \Phi_1 = \{(\forall x)(Damaged(x) \rightarrow Maintain(x)),\\ (\forall x)(Free(x) \rightarrow \neg Maintain(x)),\\ Damaged(Garage\ 1) \wedge Free(Garage\ 1)\}.$$

As mentioned above, the repair actions should be performed on $\mathsf{CORE}(\Delta)$. Generally, just for the simplicity of reasoning, the requirements set Δ_i contains both preliminary requirements statements and relevant facts. For example, *Damaged* (*Garage* 1) $\wedge$ *Free*(*Garage* 1) and *Expire*(*Garage* 2) are facts in Δ_A and Δ_B, respectively. These facts are used to model the certain scenario associated with each viewpoint's goal. This paper focuses on how to elect an acceptable common proposal for modifying the preliminary requirements specification , then we will

view the facts as being correct and not subject to the modification of preliminary requirements. This will allow us to focus our attention on choice of actions performed for modifying the preliminary requirements. Thus, we are concerned with proposals for modifying a set of problematical preliminary requirements, denoted $\mathsf{CORE}(\Delta)_P$, which is a subset of $\mathsf{CORE}(\Delta)$. In the example above,

$$\mathsf{CORE}(\Delta)_P = \{(\forall x)(Damaged(x) \rightarrow Maintain(x)), \\ (\forall x)(Free(x) \rightarrow \neg Maintain(x))\}.$$

2.3 Combinatorial Vote

Combinatorial vote has been presented by Lang in [11], where a group of voters (or agents) is supposed to express their preferences and come to a common decision concerning a set of non-independent variables to assign. Of course, the set of candidates $\mathcal{X}$ has combinatorial structure [11]. A combinatorial vote problem consists of two steps:

(1) the voters express their preference on a set of candidates within a fixed representation language;
(2) one or several optimal candidate(s) is (are) determined automatically, using a fixed vote rule.

A preference profile consists of a preference structure for each of the voters. A relational preference structure consists of a binary relationship $\geq$ on $\mathcal{X}$. A vote rule V is defined as a function mapping every preference profile P to an elected candidate, or a subset of candidates. Given a preference profile P and a vote rule V, the set of elected candidates is denoted by $Select_V(P)$. Scoring rules consists in translating the preference relation $\geq_i$ of voters into scoring function $s_i(\mathbf{x})$, such that the score $s_i(\mathbf{x})$ of a candidate $\mathbf{x}$ with respect to voter i is a function of its position in the relation $\geq_i$. The plurality and the veto rules are appropriate for combinatorial vote [11]. In this paper, we adopt the plurality rule as the vote rule to illustrate our approach. Actually, the choice of social vote rules used in the practice should depend on the specific circumstances.

The plurality rule is the scoring rule obtained by taking $s_i(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ is non-dominated for $\geq_i$, i.e., iff there is no $\mathbf{y}$ such that $\mathbf{y} >_i \mathbf{x}$. $Select_{plurality}(P)$ is the set of candidates maximizing the number of voters for whom $\mathbf{x}$ is non-dominated.

3 Identifying an Acceptable Common Proposal of Inconsistency Handling

In this section, we will transform the problem of identifying an appropriate proposal for handling inconsistency into combinatorial vote. It consists of four key steps:

(1) We define a 1-1 mapping from the set of proposals to a set of candidates that has combinatorial structure;

(2) We transform the evaluation of proposals from the perspective of an individual viewpoint into a voter's preference representation on the set of candidates;
(3) Given a social vote rule, the set of elected candidates is identified automatically.
(4) The set of elected candidates is transformed into the set of acceptable common proposals.

3.1 Proposals of Inconsistency Handling

Generally, handling inconsistency in an integrated requirements collection Δ means that stakeholders or viewpoints involved in the inconsistency are trying to reach an agreement on the modification of $\mathsf{CORE}(\Delta)_P$.

Informally, proposal for handling inconsistency should be a series of actions performed to modify $\mathsf{CORE}(\Delta)_P$. For each requirements $\alpha \in \mathsf{CORE}(\Delta)_P$, an individual proposal for handling the inconsistency will delete it from $\mathsf{CORE}(\Delta)_P$ or retain it.

Now we try to transform the problem of identifying appropriate proposals for inconsistency handling into a problem of combinatorial vote. Let $|\mathsf{CORE}(\Delta)_P|$ be the number of requirements in $\mathsf{CORE}(\Delta)_P$. Suppose that $|\mathsf{CORE}(\Delta)_P| = m$ and $A = \{a_1, \cdots, a_m\}$ be a set of propositional variables that don't appear in Δ. Then we can define a 1-1 mapping f from $\mathsf{CORE}(\Delta)_P$ to A. Further, let Π be a set of possible proposals for handling the inconsistency in $\mathsf{CORE}(\Delta)_P$, then $|\Pi| = 2^m$.

Definition 2 (Transformation Mapping). *Let* $\mathcal{X} = \{a_1, \neg a_1\} \times \cdots \times \{a_m, \neg a_m\}$. *Let* Π *be the set of possible proposals for handling inconsistency. Transformation mapping* t *is a 1-1 mapping from* Π *to* $\mathcal{X}$ *such that for every* $\pi \in \Pi$, $t(\pi) = (t_1, \cdots, t_n)$, *where for each* i $(1 \leq i \leq n)$

- $t_i = a_i$, *if* π *retains requirements* $f^{-1}(a_i)$ *in* $\mathsf{CORE}(\Delta)_P$;
- $t_i = \neg a_i$, *if* π *deletes requirements* $f^{-1}(a_i)$ *from* $\mathsf{CORE}(\Delta)_P$;

Essentially, by transformation mapping t, we transform the set of possible proposals into a set of candidates with combinatorial structure. Suppose that $v_{i_1}, \cdots, v_{i_k}$ involved in the inconsistency of Δ, the the problem of identifying appropriate proposal is transformed into the following problem:

- *Voters* $v_{i_1}, \cdots, v_{i_k}$ *to elect a winner in* $\mathcal{X}$.

Since $\mathcal{X}$ has combinatorial structure, then the latter is a problem of combinatorial vote [11]. Now we give an example to illustrate this transformation.

Example 2. Consider *Example 1.* again. *Alice* and *Bob* are involved in the inconsistency and

$$\mathsf{CORE}(\Delta)_P = \{(\forall x)(Damaged(x) \rightarrow Maintain(x)), \\ (\forall x)(Free(x) \rightarrow \neg Maintain(x))\}.$$

Now we define mapping f from $\mathsf{CORE}(\Delta)_P$ to $\{a_1, a_2\}$ as follows:

$$f((\forall x)(Damaged(x) \rightarrow Maintain(x))) = a_1,$$
$$f((\forall x)(Free(x) \rightarrow \neg Maintain(x))) = a_2.$$

There are 4 possible proposals for handling inconsistency:

- π_1: to delete $(\forall x)(Damaged(x) \rightarrow Maintain(x))$ from $\mathsf{CORE}(\Delta)_P$;
- π_2: to delete $(\forall x)(Free(x) \rightarrow \neg Maintain(x))$ from $\mathsf{CORE}(\Delta)_P$;
- π_3: to delete all the requirements in $\mathsf{CORE}(\Delta)_P$;
- π_4: to retain all the requirements in $\mathsf{CORE}(\Delta)_P$.

Then $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ and

$$t(\pi_1) = (\neg a_1, a_2);$$
$$t(\pi_2) = (a_1, \neg a_2);$$
$$t(\pi_3) = (\neg a_1, \neg a_2);$$
$$t(\pi_4) = (a_1, a_2).$$

Now we transform inconsistency handling problem into a combinatorial vote problem:

- Two voters (stand for Alice and Bob, respectively) to elect a winner in $\{(\neg a_1, a_2), (a_1, \neg a_2), (\neg a_1, \neg a_2), (a_1, a_2)\}$.

3.2 Voting for a Common Proposal

As mentioned earlier, voters' preferences on the set of candidates play an important role in combinatorial vote. In this paper, for a particular voter, we focus on the relational preference structure on $\mathcal{X}$. It should be associated with the viewpoint's preference on the set of proposals.

Intuitively, the viewpoint's preferences on the set of proposals are always associated with the degree of satisfaction of his/her goal by performing each proposal. For each proposal $\pi_i \in \Pi$, let $\pi_i(\Delta)$ denote the modification of Δ by performing the proposal π_i. Let $\lfloor G_j^i \rfloor$ denote the number of formulas of goal G_j that can be derived from $\pi_i(\Delta_j)$ consistently. Then $\lfloor G_j^i \rfloor$ may be viewed as a measure of the degree of satisfaction of the goal.

Definition 3 (Preference on Π). *Let Π be the set of possible proposals. For each i $(1 \leq i \leq n)$, a binary relationship with regard to viewpoint v_i on Π, denoted $\geq_i$, is defined as follows:*

$$\forall\ \pi_l, \pi_j \in \Pi, \pi_l \geq_i \pi_j \text{ if and only if } \lfloor G_i^l \rfloor \geq \lfloor G_i^j \rfloor.$$

Note that $\pi_l >_i \pi_j$ if and only if $\pi_l \geq_i \pi_j$ and $\pi_j \ngeq_i \pi_l$.

Definition 4 (Preference on $\mathcal{X}$). *Let Π be the set of possible proposals. For each i $(1 \leq i \leq n)$, $\geq_i$ is a binary relationship with regard to viewpoint v_i on Π. Let t is a transformation mapping from Π to $\mathcal{X}$. Then $\forall\ t(\pi_l),\ t(\pi_j) \in \mathcal{X}$,*

$$t(\pi_l) \geq_i t(\pi_j) \text{ if and only if } \pi_l \geq_i \pi_j.$$

Now we give an example to illustrate the preferences of voters.

Example 3. Consider the proposals mentioned in *Example 2.* For viewpoint v_A,

$$\begin{aligned} \lfloor G_A^1 \rfloor &= 0; \\ \lfloor G_A^2 \rfloor &= 1; \\ \lfloor G_A^3 \rfloor &= 0; \\ \lfloor G_A^4 \rfloor &= 0. \end{aligned}$$

And for v_B,

$$\begin{aligned} \lfloor G_B^1 \rfloor &= 1; \\ \lfloor G_B^2 \rfloor &= 1; \\ \lfloor G_B^3 \rfloor &= 1; \\ \lfloor G_A^4 \rfloor &= 0. \end{aligned}$$

Then

$$\begin{aligned} \pi_2 &\geq_A \pi_1, \pi_3, \pi_4 \\ \pi_1, \pi_2, \pi_3 &\geq_B \pi_4. \end{aligned}$$

and

$$\begin{aligned} & (a_1, \neg a_2) \\ \geq_A\ & (\neg a_1, a_2), (\neg a_1, \neg a_2), (a_1, a_2); \\ & (\neg a_1, a_2), (a_1, \neg a_2), (\neg a_1, \neg a_2) \\ \geq_B\ & (a_1, a_2). \end{aligned}$$

Note that candidates written on a same line are equally preferred.

In this paper, we adopt the plurality rule as the vote rule. As mentioned earlier, the plurality rule is the scoring rule obtained by taking $s_i(\mathbf{x}) = 1$ if and only if $\mathbf{x} \in \mathcal{X}$ is non-dominated for $\geq_i$, i.e., iff there is no $\mathbf{y}$ such that $\mathbf{y} >_i \mathbf{x}$. Given preferences profile P, the set of acceptable common candidates, denoted $Select_{plurality}(P)$, is the set of candidates maximizing the number of voters for whom $\mathbf{x}$ is non-dominated.

Example 4. Consider the example above again. In this case, there are two voters v_A and v_B. The preference ordering $\geq_A$, and $\geq_B$ are:

$$\begin{aligned} & (a_1, \neg a_2) \\ \geq_A\ & (\neg a_1, a_2), (\neg a_1, \neg a_2), (a_1, a_2); \\ & (\neg a_1, a_2), (a_1, \neg a_2), (\neg a_1, \neg a_2) \\ \geq_B\ & (a_1, a_2). \end{aligned}$$

The plurality rule is used as the vote rule. Then the scores of candidates are:

$$\begin{aligned} s_A((\neg a_1, a_2)) &= 0; \\ s_A((a_1, \neg a_2)) &= 1; \\ s_A((\neg a_1, \neg a_2)) &= 0; \\ s_A((a_1, a_2)) &= 0; \\ s_B((a_1, \neg a_2)) &= 1; \\ s_B((\neg a_1, a_2)) &= 1; \\ s_B((\neg a_1, \neg a_2)) &= 1; \\ s_B((a_1, a_2)) &= 0; \end{aligned}$$

Clearly, $Select_{plurality}(P) = \{(a_1, \neg a_2)\}$. That is, $(a_1, \neg a_2)$ is the winner in $\mathcal{X}$. Since $t^{-1}((a_1, \neg a_2)) = \pi_2$, π_2 is the acceptable common proposal for handling the inconsistency in $\Delta_A \cup \Delta_B$. Therefore, by voting, $(\forall x)(Free(x) \rightarrow \neg Maintain(x))$ should be deleted from Δ_B for maintaining consistency.

The combinatorial vote-based approach to identifying the acceptable common proposals presented above may be illustrated as follows:
$(\Pi, \geq_i) \xrightarrow{t} (\mathcal{X}, \geq_i) \xrightarrow{plurality\ rule} Selet_{plurality}(\geq_i,\ 1 \leq i \leq n) \xrightarrow{t^{-1}} \pi_i(winner)$.

4 Discussion and Comparison

For the combinatorial vote, the computational complexity of the different problems obtained from the choice of a given representation language (propositional logic) and a give vote rule (plurality rule) has been studied by Lang in [11]. However, there are other vote rules such as the veto rule also appropriate for combinatorial vote mentioned in [11]. The veto rule is obtained by letting $s_i(\mathbf{x}) = 1$ if and only if there is at least a candidate $\mathbf{y}$ such that $\mathbf{x} >_i \mathbf{y}$. If the veto rule is used as the vote rule in *Example 4.*, then

$$\begin{aligned} s_A((\neg a_1, a_2)) &= 0; \\ s_A((a_1, \neg a_2)) &= 1; \\ s_A((\neg a_1, \neg a_2)) &= 0; \\ s_A((a_1, a_2)) &= 0; \\ s_B((a_1, \neg a_2)) &= 1; \\ s_B((\neg a_1, a_2)) &= 1; \\ s_B((\neg a_1, \neg a_2)) &= 1; \\ s_B((a_1, a_2)) &= 0; \end{aligned}$$

And $Selet_{veto}(P) = \{(a_1, \neg a_2)\}$. The winner is also $(a_1, \neg a_2)$ under the veto rule. Of course, it is possible to get the different winners under different vote rules.

On the other hand, inconsistency handling in requirements engineering is a rather complex issue. Most works focus on the inconsistencies that result from misunderstand customer's demands or incorrect statement of requirements [17,18,5]. In contrast, the combinatorial vote-based approach is more appropriate to handling inconsistencies that result from conflictive goals or intentions of stakeholders. This kind of inconsistency handling is always associated with many social activities such as vote and negotiation. It is not just a technical issue. The vote-based approach may be viewed as a first attempt to provide appropriate mechanism for handling inconsistencies result from conflict goals or intentions.

The preferences on the set of possible proposals of each individual viewpoints play an important role in electing the acceptable common proposals in the vote-based approach. In this paper, we just use $\lfloor G_i^l \rfloor$ to evaluate the relative importance of proposal π_l from the perspective of viewpoint v_i. However, different goals (formulas) in G_i may have different relative importance. So the relative importance of each formula of G_i that can be derived from $\pi_l(\Delta_i)$ should be also taken into consideration in representing preferences on Π of v_i. This would be a direction of future work.

5 Conclusions

We have presented a combinatorial vote-based approach for identifying the acceptable common proposals for handling inconsistency in Viewpoints framework.

Identifying appropriate inconsistency handling actions is still a difficult, but important challenge. The vote-based approach presented in this paper focuses on the inconsistency that results from conflicting intentions of different stakeholders. The main contribution of this paper is to transform identifying appropriate proposals for handling inconsistency into a problem of combinatorial vote. It consists of four key steps:

(1) we define a 1-1 mapping from the set of proposals to a set of candidates that has combinatorial structure;
(2) we transform the evaluation of proposals from the perspective of an individual viewpoint into a voter's preference representation on the set of candidates;
(3) Given a social vote rule, the set of elected candidates is identified automatically.
(4) The set of elected candidates is transformed into the set of acceptable common proposals.

However, inconsistency handling in requirements engineering is a rather complex issue. For the vote-based approach presented in this paper, the choice of social vote rules used in the combinatorial vote and the approaches to evaluating each proposal should be considered further in the future work.

Acknowledgements

This work was partly supported by the National Natural Science Fund for Distinguished Young Scholars of China under Grant No.60625204, the Key Project of

National Natural Science Foundation of China under Grant No.60496324, the National Key Research and Development Program of China under Grant No. 2002CB312004, the National 863 High-tech Project of China under Grant No. 2006AA01Z155, the Knowledge Innovation Program of the Chinese Academy of Sciences, and the NSFC and the British Royal Society China-UK Joint Project.

References

1. Finkelsetin, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. International Journal of Software Engineering and Knowledge Engineering 2(1), 31–58 (1992)
2. Kotonya, G.I.: Sommerville: Viewpoints for requirements definition. IEE Software Eng.Journal 7, 375–387 (1992)
3. Andrade, J., Ares, J., Garcia, R., Pazos, J., Rodriguez, S., Silva, A.: A methodolog ical framework for viewpoint-oriented conceptual modeling. IEEE Trans. Softw. Eng. 30, 282–294 (2004)
4. Gervasi, V.D.: Zowghi: Reasoning about inconsistencies in natural language re quirements. ACM Transaction on Software Engineering and Methodologies 14, 277–330 (2005)
5. Hunter, A.B.: Nuseibeh: Managing inconsistent specification. ACM Transactions on Software Engineering and Methodology 7, 335–367 (1998)
6. Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software Technology 45, 993–1009 (2003)
7. Nuseibeh, B., Easterbrook, S., Russo, A.: Leveraging inconsistency in software development. IEEE Computer 33, 24–29 (2000)
8. Nuseibeh, B.S., Easterbrook, A.: Russo: Making inconsistency respectable in software development. Journal of Systems and Software 58, 171–180 (2001)
9. Gabbay, D., Hunter, A.: Making inconsistency respectable 2:meta-level handling of inconsistent data. In: Moral, S., Kruse, R., Clarke, E. (eds.) ECSQARU 1993. LNCS, vol. 747, pp. 129–136. Springer, Heidelberg (1993)
10. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency handling in multiperspective speci?cations. IEEE Trans. on Software Engineering 20, 569–578 (1994)
11. Lang, J.: From logical preference representation to combinatorial vote. In: Proceedings of 8th International Conference on Principles of Knowledge Representation and Reasoning, pp. 277–288. Morgan Kaufmann, San Francisco (2002)
12. Leite, J.P.A.: Freeman: Requirements validation through viewpoint resolution. IEEE Trans. on Soft. Eng. 17, 1253–1269 (1991)
13. Robinson, W.N.: Integrating multiple specifications using domain goals. In: IWSSD '89: Proceedings of the 5th international workshop on Software specification and design, pp. 219–226. ACM Press, New York, NY, USA (1989)
14. Nuseibeh, B., Kramer, J., Finkelstein, A.: Viewpoints: meaningful relationships are difficult? In: Proceedings of the 25th International Conference on Software Engineering, pp. 676–681. IEEE Computer Society Press, Los Alamitos (2003)

15. Kifer, M., Lozinskii, E.L.: A logic for reasoning with inconsistency. Journal of Automated Reasoning 9, 179–215 (1992)
16. Mu, K., Jin, Z., Lu, R.: Inconsistency-based strategy for clarifying vague software requirements. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 39–48. Springer, Heidelberg (2005)
17. Easterbrook, S., Nuseibeh, B.: Managing inconsistencies in an evolving specification. In: Proceedings of the Second International Symposium on Requirements Engineering (RE95), pp. 48–55 (1995)
18. Easterbrook, S.M., Chechik, A.: framework for multi-valued reasoning over inconsistent viewpoints. In: Proceedings of International Conference on Software Engineering (ICSE'01), Toronto, Canada, pp. 411–420 (2001)