BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **13**, Special Issue

Sofia • 2013

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2013-0039

A Negotiation Framework for Managing the Requirements Changes

Yirui Zhang, Ying Jin, Jianxiu Bai, Jing Zhang

College of Computer Science and Technology, Jilin University, Changchun, 130012, China
Emails: SoNbility@163.com jinying@jlu.edu.cn baijianxiu@126.com Corresponding author: zhangjing99@jlu.edu.cn

Abstract: *The consistent system requirements set is the basis of successful software projects. The requirements change is very usual in a software project, and it may cause inconsistency of the requirements set, and become the key factor that affects the quality of the requirements and the software. Aiming at the problem of requirements inconsistencies caused by the requirements change, this paper proposes a compromise-based negotiation framework to manage the requirements changes, illustrates the efficiency of the proposed method by a software engineering case, gives a contrast experiment with the current mainstream method, and finally gives a comparison with the related work and a conclusion. The experimental results show that the framework proposed in this paper is more flexible and accurate than the results of the current popular framework, so it is more suitable for the requirement changes management.*

Keywords: *Software engineering, requirements inconsistencies, compromise, negotiation framework.*

1. Introduction

Software requirements engineering is the most critical part of the entire software engineering. Compared with traditional industrial engineering, a software requirement has the following features: ambiguity, uncertainty, subjectivity and

variability. In the software development process, the requirements change throughout the entire life cycle of the software project [1]. Generally speaking, appropriate changes of the requirements will not only improve a more perfect system, but will enhance the quality of the software requirements specifications and software products. However, many uncontrolled changes will cause many fatal problems in the software development process [2]. Therefore, it is urgent and necessary to provide a flexible and efficient management strategy to eliminate the inconsistencies caused by the requirements changes.

At present the logic-based technology is widely recognized to eliminate the requirements inconsistencies [3]. Alchourron, Gäardenfors, and Makinson [4] proposed that the priority-based idea could eliminate the requirements inconsistencies [4]: the new requirements always have higher priority than the old ones, using new requirements to replace the old ones could eliminate inconsistencies. Garcez et al. [5, 6] proposed using the combination of the cycle reductive inference and inductive learning to eliminate the inconsistencies of the requirements specification. They believe that the development of requirements specifications must include revision and deduction: and using the cycle two-stage model represented by two phases that are composed of analysis and revision can eliminate the inconsistencies of the requirements and remain the main requirements goal and nature. Booth [7] proposed the use of a negotiation-based framework to eliminate the requirements inconsistencies.The introduction of the negotiation can help adjust the process flexibly, and ensures the elimination of the inconsistencies. Ke-Dian [8], Mu et al. [12] proposed a series of activities to manage the changes of the software requirements. In [8] they used the belief revision-based negotiation framework to eliminate the inconsistencies. In the negotiation framework, there are three schemes to deal with the requirements changes request: fully accept the request, give up the requirements request and partly accept the request. Their framework as a newer method for managing the requirements changes, has been widely recognized.

However, the current scheme and framework cannot manage the requirements changes flexibly and eliminate the requirements inconsistencies accurately. The project proposed in [4] by Alchourron, Gäardenfors, Makinson can always eliminate the inconsistencies, but the method cannot ensure that the new requirements are more reliable than the old ones, it has to replace the old ones with the new requirements, it cannot retain the old requirements and abandon the new ones, this method does not have flexible managing tools. The project proposed in [5, 6] by Garcez can eliminate the inconsistencies. But the process that combines the cycle reductive inference and the inductive learning is too long, and some correct requirements may be abandoned because of the different reasoning and learning methods, so this project cannot eliminate the inconsistencies accurately. The negotiation-based framework [7] proposed by Booth can manage the requirements changes flexibly, but the framework is a processing idea, it does not have a specific implementation scheme. The belief revision-based negotiation framework proposed in [8] by Ke-Dian et al. can adjust the negotiation process flexibly, but the framework can only handle the inconsistency caused by the single

atom change, it cannot handle the complicated situation caused by multi-atom changes when changing the requirements and ensure the eliminating of the inconsistencies accurately.

Section 2 introduces the logical representation of the requirements specification and the definition of the requirements inconsistencies. Section 3 explains the idea of the compromise and builds a compromise-based negotiation framework to eliminate the requirements inconsistencies. Section 4 gives a case study of the software engineering to prove the efficiency of the method. Section 5 gives a contrast experiment to prove the flexibility and validity of the proposed method. Section 6 compares the relevant work done at home and abroad, summarizes this paper and comes up with the future work.

2. Preliminaries

The requirement priority is the order to be achieved of the requirement in the software development life cycle. Generally, the division of the priorities is based on the property, quality, degree of importance, degree of urgency of the requirement and the relationship between the requirements. The common division method divides the requirements into several groups that have priority. For example: the theory of Analytical Hierarchies (AHP) [13] and the Quality Function Deployment (QFD) [14]. Generally, most division methods divide the requirements into three-level priority [1, 15] and five-level priority [16]. This paper gives the concept of the requirement priority as follows:

The priority order L^m [8]: Let m be a natural number, $L^m = \{l_1, l_2, ..., l_m\}$ is a priority order, where $l_i (i \in m)$ is a priority order. Generally, $i < j (i, j \in m)$, iff $l_i < l_j$, $l_i < l_j$ means that the requirements with priority l_i are more preferable than the requirements with priority l_j.

For a priority order $L^3 = \{l_1, l_2, l_3\}$ the meaning of each priority [1] in L^3 can be explained as follows: $l_1 : \text{High}, l_2 : \text{Medium}, l_3 : \text{Low}$, under the interpretation, for example, that the requirements with priority l_2 are more preferable than the requirements with priority l_3. That is, the requirements with priority l_2 have higher priorities than the requirements with priority l_3.

This paper will use the priority order L^3 to handle most issues, though it is not the most flexible one, the priority order has a wide range of representation.

The definitions of requirements inconsistencies vary in software engineering [1]. The logic-based work considers the requirements inconsistencies as logical contradictions. This paper defines the logical contradictions as requirements inconsistency [6]: Consider a requirements set *S*, if $\exists a$, and $(S \vdash \neg a) \cap (S \vdash \neg a)$, then there is inconsistency in *S*. Namely, $a \wedge \neg a$ is inconsistency, denoted as $\perp$.

3. A compromise-based negotiation framework

In requirements engineering, the final complete set of the system is composed of the results that classify, sort, merge and revise the requirements set proposed by the customers and eventually form the final requirements set. Inconsistency must not exist in the system set. The compromise is a negotiating strategy and its purpose is to avoid a deadlock in the negotiating process, thus contributing to the success of the negotiations.

In order to get satisfied and reasonable results, the framework proposed in this paper completed the negotiation process, in which at least one party makes concessions, and eliminated the inconsistencies existing in the system. Through the following sections, this paper creates the negotiation framework.

3.1. Logical representation of users' requirements

The classical logic-based language in the representation of the requirements is very popular at present [3, 17]. Though different symbols and tools can be used to express the requirements in each phase of the software development process, the first order logic without a free variable can always indicate the inconsistencies of the requirements set [3, 8]. This paper uses the first order logic language without function symbols and existential quantifiers to represent the consistency of the requirements set.

Let L_{Φ_0} be the set composed of a logical language, such as the classical atom Φ_0 and logical connectives $\{\wedge, \vee, \neg, \rightarrow\}$, then it can be used to express a natural language. Thus, the negotiator $C_1's$ requirement and $C_2's$ requirement can be expressed as logical symbols, and get the requirements set S and T.

Example 1. The Access Control System is widely used and known. Literature [8] gave the requirements text of a small Access Control System, the requirements text is very normative and has already been used many times as a requirements instance. The specific description is as follows:

a. The requirements specification of the Access Control System of an area which manages parking is as follows: The car is not allowed to enter the residential area without a specific permission. The car is allowed to enter the residential area with a specific permission. The situation when a car tries to enter the residential area without a specific permission will trigger the alarm. If the alarm is triggered, the owner of the car will not be able to press the button for entering the residential area again.

b. The requirements specification of the Access Control System of an area which manages the fire engines is as follows: The fire engines are regarded as emergency vehicles. The emergency engine is allowed to enter the residential area without a specific permission. In addition to the emergency engines, the others are not allowed to enter the residential area without a specific permission.

The following symbols used to indicate the natural language and the logical representation of the requirements set are given in Table 1.

Table 1. Symbols used to indicate the natural language

(1) Use a predicate symbol $\mathrm{Aut}(x)$ to denote x is awarded a special license. (2) Use a predicate symbol $\mathrm{Ent}(x)$ to denote x can enter the residential area. (3) Use a predicate symbol $\mathrm{Ala}(x)$ to denote that if x tries to enter the residential area, the alarm will be triggered. (4) Use a predicate symbol $\mathrm{Push}(x,y)$ to denote x pushes the button y. (5) Use a predicate symbol $\mathrm{Eme}(x)$ to denote x is the emergency engine. (6) Use constant entr to denote the button for entering the residential area. (7) Use constant fire_e to denote the fire engines	a. The requirements specification of the Access Control System of an area which manages the parking: $S = \{\neg\mathrm{Aut}(\mathrm{fire}\_e) \to \neg\mathrm{Ent}(\mathrm{fire}\_e),$ $\mathrm{Aut}(\mathrm{fire}\_e) \to \mathrm{Ent}(\mathrm{fire}\_e),$ $\neg\mathrm{Aut}(\mathrm{fire}\_e) \to \neg\mathrm{Ala}(\mathrm{fire}\_e),$ $\mathrm{Ala}(\mathrm{fire}\_e) \to \neg\mathrm{Push}(\mathrm{fire}\_e,\mathrm{entr})\}$ b. The requirements specification of the Access Control System of an area which manages the fire engines: $T = \{\mathrm{Eme}(\mathrm{fire}\_e),$ $\mathrm{Eme}(\mathrm{fire}\_e) \to$ $\mathrm{Ent}(\mathrm{fire}\_e) \wedge \neg\mathrm{Aut}(\mathrm{fire}\_e),$ $\neg\mathrm{Aut}(\mathrm{fire}\_e) \wedge \neg\mathrm{Eme}(\mathrm{fire}\_e) \to$ $\neg\mathrm{Ent}(\mathrm{fire}\_e)\}$

3.2. Use the priority order to divide the requirements set

The priority equivalence relation is necessary when dividing the set. In this paper the priority equivalence relation R is defined as follows:

Definition 1. There is given the priority order $L^m = \{l_1, l_2, ..., l_m\}$ and the set $S = \{a_1, ..., a_n\}(n \in N)$, to any element a_i and a_j, $(a_i, a_j) \in R$ iff they have the same priority l_k, and R is called the priority equivalence relation.

According to this equivalence relation, the elements of the set S form a different set of priorities $\Delta^k (k \in m)$. Then we define the set $S_\Delta = \{\Delta^k \mid (k \in m)\}$ as the equivalence class set. The requirements users C_1 and C_2 can divide the equivalence classes set according to the priority equivalence relation R.

According to the above definition, divide S and T by the use of the priority order L^m, and get the equivalence classes set S_Δ and T_∇. This paper will use the priority order L^3 to divide the requirements set, and get the equivalence classes set: $S_\Delta = \{\Delta^1, \Delta^2, \Delta^3\}, T_\nabla = \{\nabla^1, \nabla^2, \nabla^3\}$.

Example 2. Set the requirements set $S = \{a \vee \neg b, \neg a \vee b, a \vee \neg c, \neg c \vee \neg d\}$ and the requirements set $T = \{e, \neg e \vee (\neg a \wedge b), a \vee \neg b \vee e\}$, then use the priority order L^3 to divide the requirements set:

The equivalence classes set for S is: $S_\Delta = \{\Delta^1, \Delta^2, \Delta^3\}$, and a possible division is: $\Delta^1 = \{a \vee \neg b, \neg a \vee b\}, \Delta^2 = \{a \vee \neg c\}, \Delta^3 = \{\neg c \vee \neg d\}$.

The equivalence classes set for T is: $T_\nabla = \{\nabla^1, \nabla^2, \nabla^3\}$, and a possible division is: $\nabla^1 = \{e\}, \nabla^2 = \{\neg e \vee (\neg a \wedge b), a \vee \neg b \vee e\}, \nabla^3 = \{\varnothing\}$.

3.3. Set the system set and the problem domain set

The problem domain refers to the scope of the problem, the internal relation among the problems and the logical possible space [18]. In this paper the interpretation set

of the system set is the problem domain set. Here the interpretation set of the formulas set is given as follows:

Definition 2. R is a formulas set, then the atoms set $G(R)$ is the union of the atoms sets of all the atoms in R, and the interpretation set $A(R)$ is $A(R)=\{a \mid \exists U \in R, a \vDash U\}$, where a is an interpretation, U is a formula.

Example 3. A formulas set $R=\{\neg a, a \to b\}$, then the atoms set $G(R)$ is $\{a,b\}$, the interpretation set $A(R)$ is $\{00,01\}$.

Thus, this paper sets up the system set Sys, its interpretation set as the problem domain set ESys. Set the initial value of the system set Sys as all the atoms appeared in the negotiation, and the initial value of the problem domain set ESys as the interpretation set of the system set Sys.

3.4. Set the negotiation order and define the solution set

The compromise ideology refers to one or two making concessions to come to an agreement. According to the compromise ideology, both of the negotiators can extend their own solution set in the problem domain, and get an agreement after concessions, finally complete the negotiation.

1. Set the negotiation order: As mentioned in 2.1, the requirements with higher priority are more preferable than the requirements with a lower priority. If the negotiators use the dynamic division method: they need to divide their own requirements priorities once again when negotiating, use a more reliable negotiation method: choose their own highest priority requirement to negotiate. So the negotiation process will be more flexible, the negotiation result will be more reasonable.

Thus in this paper, both negotiators re-divide their own requirements priorities when facing each negotiation, and the equivalence classes Δ^1 and ∇^1 are selected to be negotiated.

2. Define the solution set: In this paper we define the formulas set which meets the interpretation set, the relation $(=_{\odot}, >_{\odot})$ on interpretation, division operation $(\bullet)$ and the result set as follows:

Definition 4. R is a formulas set, $A(R)$ is the interpretation set, then the formulas set $R \| A(R)$ which meets the interpretation set $A(R)$ is: $R \| A(R)=\{U \in R \mid \exists a \in A(R), a \vDash U\}$, where U is a formula, a is an interpretation.

Definition 5. R is a formulas set, then the elation $=_{\odot}$ and $>_{\odot}$ on interpretation are: To $\forall a,b, a=_{\odot} b$, iff $\left|R \| \{a\}\right| = \left|R \| \{b\}\right|$, $a >_{\odot} b$ iff $\left|R \| \{a\}\right| > \left|R \| \{b\}\right|$, where a,b is an interpretation respectively.

Definition 6. Given a formulas set R and an interpretation set E, then the division operation $\bullet$ of R to E is

$R \bullet E=\{k^1,...,k^n\}$, where $k^1,...,k^n$ is a division of E which meets

$\forall a,b \in k^i, a=_{\odot} b, (1 \le i \le n)$, $\forall a \in k^i, \forall b \in k^j, a >_{\odot} b, (1 \le i < j \le n)$.

Example 4. R is a formulas set, $R=\{\neg a, a\to b, a\wedge b\}$, $A(R)=\{00,01,11\}$ then:

(a) $R\,||\,\{00\}=\{\neg a, a\to b\}, R\,||\,\{01\}=\{\neg a\}, R\,||\,\{11\}=\{a\to b, a\wedge b\}$;

(b) $00=_{\odot}11, 00>_{\odot}01, 11>_{\odot}01$. (c) $R\bullet A(R)=\{\{00,11\},\{01\}\}$.

Definition 7. Given the equivalence classes set $\Delta^m(m\in 1,2,3)$, $\nabla^n(n\in 1,2,3)$, and the problem domain set ESys, then the result got by the use of the division operation $\Delta^m\bullet \mathrm{ESys}$ is the result set under Δ^m, denoted as $S^m[i](i\in N)$. And the result got by the use of the division operation $\nabla^n\bullet \mathrm{Esys}$ is the result set under ∇^n, denoted as $T^n[j](j\in N)$.

Example 5. Given the problem domain set:

$\mathrm{ESys}=\{111,110,101,100,011,010,001,000\}$, and the equivalence classes set

$$\Delta^1=\{a,b,a\vee\neg b\},\ \Delta^2=\{\neg a\vee b\},\ \Delta^3=\{b\wedge c\},\ \nabla^1=\{\neg a,b,\neg c\},\ \nabla^2=\{\neg a\vee r\},$$
$$\nabla^3=\{a\vee\neg r, b\vee r\}.$$

Then get the result set under Δ^1 by the use of the division operation $\Delta^1\bullet \mathrm{Esys}$ is $S^1[0]=\{111,110\}$, $S^1[1]=\{101,100\}$, $S^1[2]=\{011,010,001,000\}$.

And the result set under ∇^1 by the use of the division operation $\nabla^3\bullet \mathrm{ESys}$ is $T^3[0]=\{111,110,101,010\}$, $T^3[1]=\{100,011,001,000\}$.

3.5. Structure of the compromise algorithm

Based on the idea of compromise, under the premise that the system set Sys and the problem domain set ESys are known, the result set $S^m[i]$ and $T^n[j]$ under the equivalence classes set Δ^m and ∇^n are selected to be negotiated to eliminate the inconsistencies possibly existing in the result set of the negotiators. The algorithm and chart (Fig. 1) is as follows:

```
.....Compromise – Algorithm.....
a = 0, b = 0, tr = true // initialization
If ((S^m[i] ≠ ∅) ∩ (T^n[j] ≠ ∅))
{
 While(S^m[i] ∩ T^n[j] = ∅) // loop the alternant compromise process
 {
 If (tr = true)
 {a++, S^m[i] = S^m[i−1] ∪ S^m[i], tr = false} // S makes compromise
 else
 {b++, T^n[j] = T^n[j−1] ∪ T^n[j], tr = true} // T makes compromise
 }
 W = S^m[i] ∩ T^n[j], Sys = Sys ∪ (Δ^m || W) ∪ (∇^n || W)
}
Sys = Sys ∪ (Δ^m || ESys) ∪ (∇^n || ESys)
```

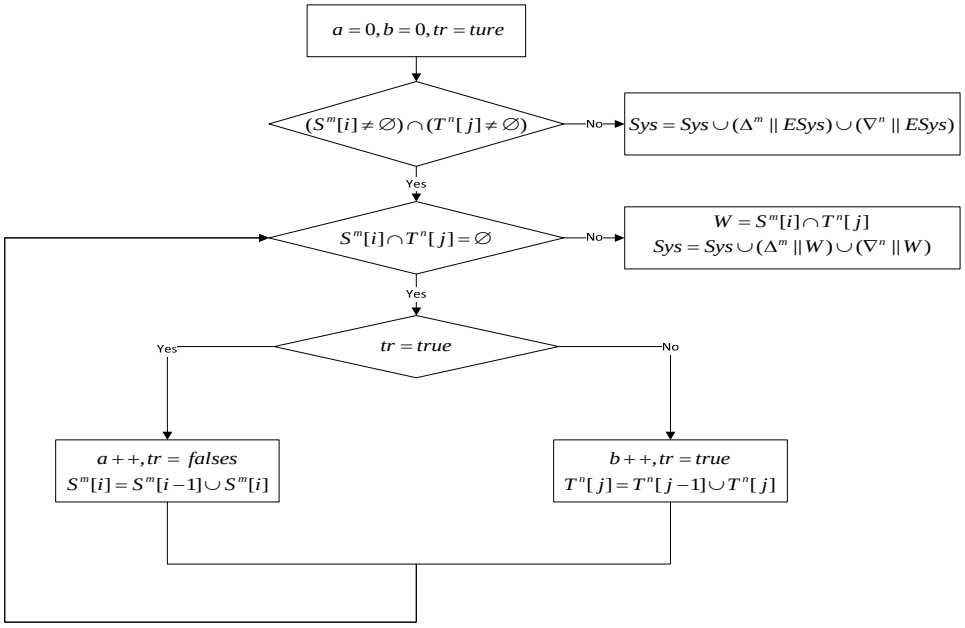

Fig. 1. The compromise algorithm

3.6. Structure of the compromise-based negotiation framework

Based on the above steps, this paper structures the compromise-based negotiation framework. This framework can help achieve a flexible negotiation process, and get accurate negotiation results. The frame chart is shown in Fig. 2.

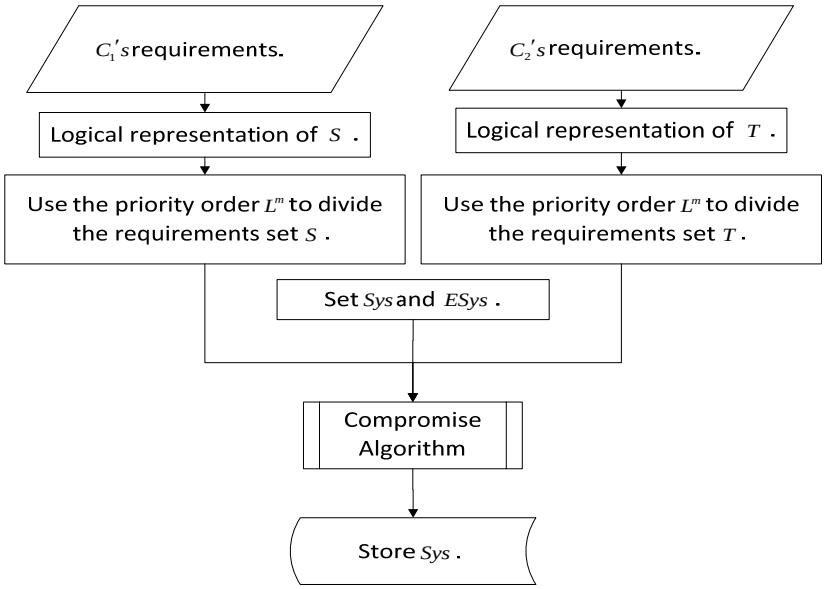

Fig. 2. The compromise-based negotiating framework

4. A case study

Automated Clearing System is a management system for clearing the goods in a mall with computer-aided control. At present, most of the exits in the shopping mall in our country are installed with an Automated Clearing System. Among them, the requirements of the mall managers and the requirements of the customer are shown in Table 2.

Table 2. The Requirements of the mall managers

(1) Customers are required to use cash and be line up (2) If customers do not line up, there will be no discount (3) If customers use a card to checkout, they could enjoy the discount	(1)Customers are required to use a card and enjoy the discount (2) If customers line up, then they get the chance of a discount (3) If customers use a cash checkout, then they enjoy the discount (4) If customers use cash, then they do not have to line up

Using logic symbols express the problem: 1. cash checkout expressed as a; credit card checkout expressed as $\neg a$; line up checkout expressed as b; a discount expressed as c. The functional logic symbols that the shopping mall managers and consumers need to implement are shown in Table 3.

Table 3. The logical requirements

Manager's requirement S	Consumer's requirement T
$a \wedge b$ $\neg b \to \neg c$ $\neg a \to c$	$a \to c$ $b \to c$ $a \to \neg b$ $\neg a \wedge c$

From the above, the requirements set $S \cup T$ contains $a \wedge \neg a$, so $S \cup T \vdash \perp$. So we use this negotiation framework to eliminate inconsistencies, get a new collection of the system.

(a) First, according to the sequence of L^3,we divide S and T for the first time, and they are divided as follows:

$$\Delta^1 = \{a \wedge b, b \vee \neg c\}, \Delta^2 = \{a \vee c\}, \Delta^3 = \{\varnothing\} \ ,$$

$$\nabla^1 = \{\neg a \wedge c, \neg b \vee c\}, \nabla^2 = \{\neg a \vee c\}, \nabla^3 = \{\neg a \vee \neg b\}.$$

Set $\mathrm{Sys} = \{\varnothing\}$, set $\mathrm{ESys} = \{111,110,101,100,011,010,001,000\}$. Then the negotiation is as follows:

With the use of $\Delta^1 \bullet \mathrm{ESys}$ we get:

$S^1[0] = \{111,110\}, S^1[1] = \{100,011,010,000\}, S^1[2] = \{101,001\}, S^1[3] = \{\varnothing\}$.

With the use of $\nabla^1 \bullet \mathrm{ESys}$ we get:

$T^1[0] = \{011,001\}, T^1[1] = \{111,101,100,000\}, T^1[2] = \{110,010\}, T^1[3] = \{\varnothing\}$.

$S^1[0] \cap T^1[0] = \varnothing$, so S first makes concessions, then $S^1[1] = S^1[0] \cup S^1[1]$. Rejudge, $S^1[1] \cap T^1[0] = \{011\} \neq \varnothing$, then the two reach a compromise.

$$W = \{011\}, \text{Sys} = \text{Sys} \cup (\Delta^1 \| W) \cup (\nabla^1 \| W) = \{b \vee \neg c, \neg a \wedge c, \neg b \vee c\}.$$

(b) According to the priority order L^3,divide $S - \Delta^1$ and $T - \nabla^1$ again, the division result is:

$$\Delta^{1'} = \{a \vee c\}, \Delta^{2'} = \{\varnothing\}, \Delta^{3'} = \{\varnothing\}, \nabla^{1'} = \{\neg a \vee c, \neg a \vee \neg b\}, \nabla^{2'} = \{\varnothing\}, \nabla^{3'} = \{\varnothing\}.$$

Then $\text{Sys} = \{b \vee \neg c, \neg a \wedge c, \neg b \vee c\}, \text{ESys} = \{011\}$, negotiate between $\Delta^{1'}$ and $\nabla^{1'}$:

With the use of $\Delta^{1'} \bullet \text{ESys}$ and $\nabla^{1'} \bullet \text{ESys}$ we get:

$S^{1'}[0] = \{011\}, S^{1'}[1] = \{\varnothing\}$, $T^{1'}[0] = \{011\}, T^{1'}[1] = \{\varnothing\}$.

$S^{1'}[0] \cap T^{1'}[0] = \{011\} \neq \varnothing$, then the two can reach an agreement without concessions:

$$W = \{011\}, \text{Sys} = \text{Sys} \cup (\Delta^{1'} \| W) \cup (\nabla^{1'} \| W) =$$
$$= \{b \vee \neg c, \neg a \wedge c, \neg b \vee c, a \vee c, \neg a \vee c, \neg a \vee \neg b\}.$$

At this point the requirements specification is changed as:

$\{b \vee \neg c, \neg a \wedge c, \neg b \vee c, a \vee c, \neg a \vee c, \neg a \vee \neg b\}$, there is no inconsistency.

The entire requirements specification can be translated from a logical language into natural language descriptions as shown in Table 4.

Table 4. The requirements of Mall Automated Clearing System

The Automated Clearing System	(1) customers are required to use a card and enjoy a discount (2) if customers line up, then they get the chance of a discount (3) if customers do not line up, then they do not get the chance of a discount (4) If customers use a cash checkout, then they enjoy the discount (5) if customers use a card checkout, then they enjoy the discount (6) If customers use a cash, then they do not have to line up

As we can see from Table 4, after negotiations, in Mall Automated Clearing System requirements there does not exist any inconsistency, which proves the efficiency of the method proposed.

5. Contrast experiment

In order to verify the proposed processing architecture flexibility and accuracy, we designed experiments and compared them with a framework of eliminating inconsistencies processing proposed by [4] and [8].

The experimental environment is restricted so that the two parts of negotiating are the same, our method selects a different inconsistencies problems, using the framework proposed in this paper and the architecture proposed in [4] and [8] to test.

According to [8], Table 5 gives two groups of problems to be processed, Table 6 shows the comparison of results of three different processing architectures. The experimental results show that the proposed architecture can flexibly give all solutions for all types of inconsistencies (questions 1, 2). The architecture proposed by [4] can only give a solution for all types of inconsistencies (questions 1, 2). Although the architecture proposed by [8] can give some solutions for inconsistencies (question 1), it cannot accurately resolve all types of inconsistencies (question 2).

Considering the results of the process, the proposed processing architecture based on a compromise and negotiation framework is flexible in eliminating an inconsistency and can lead to a complete result, and keep accuracy.

Therefore, the experiment proved that the compromise ideas based framework is more flexible compared to priority-based processing architecture proposed in [4], and more accurate compared to the negotiating framework based on belief revision proposed in [8].

Table 5. Two pending issues

Set	Problem 1	Problem 2
A set of two parts of negotiation	$S=\{a,b,c\}$ $T=\{\neg b,\neg c\}$	$S=\{a,\neg a\vee b,c,b,d\}$ $T=\{\{\neg d\}\,\|\|\,\{b\}\}$

Table 6. The comparison of three treatment options' processing results

Problem	Result using the architecture in document [4]	Result using the architecture in document [8]	Result using the architecture in this paper
Problem 1	$\{a,\neg b,\neg c\}$	$\{a,b,c\}$ or $\{a,\neg b,c\}$ or $\{a,b,\neg c\}$ or $\{a,\neg b,\neg c\}$	$\{a,b,c\}$ or $\{a,\neg b,c\}$ or $\{a,b,\neg c\}$ or $\{a,\neg b,\neg c\}$
Problem 2	$\{a,\neg a\vee\neg d,c,\neg d\}$	$\{a,\neg a\vee b,c,b,d\}$ or $\{a,\neg a\vee b,c,\neg d\}$	$\{a,\neg a\vee\neg d,c,\neg d\}$ or $\{\neg a\vee\neg d,c,d\}$ or $\{a,c,d\}$

6. Conclusion and comparison

The requirements inconsistencies issue is one of the hot issues, many researchers have done extensive and in-depth work connected with this problem [1, 17]. Applying the use of the logical language, this paper uses the idea of a compromise negotiation to solve such problems, and describes the related work.

The process towards requirements inconsistencies is usually based on logical language. The basic built in ideas of the processing framework can be divided into priority-based and non-priority based. Alchourron, Gäardenfors and

Makinson [4] proposed a priority-based idea to eliminate inconsistencies. This method did not take into consideration the situation that the new requirements may be wrong and the old be more reasonable. Booth proposed that a non-priority based negotiating framework can be used to eliminate the requirements inconsistencies in [7]. But this scheme proposed a processing thought and a processing framework, it did not realize a specific flow. Ke-Dian et al. [8] proposed a complete framework based on the negotiation revision to manage the requirements changes. However, the framework may not be used to deal with the inconsistencies arising from complex requirements, it has great limitations when facing the actual requirements inconsistency problems in software engineering.

Aiming at the requirements inconsistencies problems, this paper proposed a compromise-based negotiation framework to eliminate the requirements inconsistencies: First the software requirements are expressed by a logical language, then the scope of the problem domain is confirmed by the use of a system set according to the priority order, finally the requirements inconsistencies are eliminated. It presented the complete process of handling a case and proved the flexibility and accuracy of the proposed framework through a contrast experiment. In future studies we will consider joining a reasonable method for dividing the priority orders and a better scheme for managing the requirements changes.

Acknowledgements: This work is supported by the Program for New Century Excellent Talents in University (No NECT-10-0436).

References

1. Sikora, E., B. Tenbergen, K. Pohl. Industry Needs and Research Directions in Equirements Engineering for Embedded Systems [J]. – Requirements Engineering, Vol. **17**, 2012, No 1, 57-78.
2. Jin, Z., L. Liu, Y. Jin. Software Requirements Engineering: Principles and Methods [M]. Bei Jing, Science Press, 2008 (in Chinese).
3. Hunter, A., B. Nuseibeh. Managing Inconsistent Specification: Reasoning, Analysis, and Action. – ACM Transactions on Software Engineering and Methodology, Vol. **7**, 1998, No 4, 335-367.
4. Alchourron, C., P. Gäardenfors, D. Makinson. On the Logic of Theory Change: Partial Meeting Contraction and Revision Functions [J]. – Journal of Symbolic Logic, Vol. **50**, 1985, No 2, 510-530.
5. Garcez, A. S., A. Russo, B. Nuseibeh, J. Kramer. An Analysis-Revision Cycle to Evolve Requirements Specifications [C]. – In: Proceedings of the 16th IEEE Conference on Automated Software Engineering (ASE'2001), Coronado, USA, 26-29 November 2001, 354-358.
6. Garcez, A. S., A. Russo, B. Nuseibeh, J. Kramer. Combining Abductive Reasoning and Inductive Learning to Evolve Requirements Specifications [J]. – IEEE Proceedings of Software, Vol. **150**, 2003, No 1, 25-38.
7. Booth, R. A. Negotiation-Style Framework for Non-Prioritised Revision [C]. – In: Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'2001), Siena, Italy, 8-10 July 2001, 137-150.
8. Ke-Dian, M., L. Weiru, J. Zhi, H. Jun, D. Bell. Managing Software Requirements Changes Based on Negotiation-Style Revision [J]. – Journal of Computer Science and Technology, Vol. **26**, 2011, No 5, 890-907.

9. M u, K., Z. J i n, R. L u, Y. P e n g. Handling Non-Canonical Software Requirements Based on Annotated Predicate Calculus [J]. – Knowledge and Information System, Vol. **11**, 2007, No 1, 85-104.
10. M u, K., Z. J i n, R. L u. Inconsistency-Based Strategy for Clarifying Vague Software Requirements [C]. – In: Proceedings of AI2005, LNCS3809, 2005, 39-48.
11. M u, K., W. L i u, Z. J i n, R. L u, A. Y u e, D. A. B e l l. Handling Inconsistency in Distributed Software Requirements Specifications Based on Prioritized Merging [J]. – Fundam. Inform., Vol. **91**, 2009, No 3-4, 631-670.
12. M u, K., Z. J i n, R. L u, W. L i u. Measuring Inconsistency in Requirements Specifications [C]. – In: Proceedings of ECSQARU'2005, LNAI3571, 2005, 440-451.
13. S a a t y, T. L. Modeling Unstructured Decision Problems – the Theory of Analytical Hierarchies [J]. – Mathematics and Computers in Simulation, Vol. **20**, 1978, No 3, 147-158.
14. P a r d e e, W. J. To Satisfy and Delight Your Customer: How to Manage for Customer Value [M]. New York, Dorset House Publishing, 1996.
15. W i e g e r s, K. E. Software Requirements [M]. 2nd Ed. Portland, Microsoft Press, USA, 2003.
16. D a v i s, A. Just Enough Requirements Management: Where Software Development Meets Marking [M]. New York, Dorset House Publishing, 2005.
17. A l-K h a l d i, R. Inconsistency Management in Software Functional Requirements: A Machine Learning System [J]. LAP Lambert Academic Publishing, Germany, 2012.
18. B o n o m a, T. V. Case Research in Marketing: Opportunities, Problems, and a Process. – Journal of Marketing Research, Vol. **12**, 1985, No 1, 199-208.