Uncertainty Handling in Tabular-Based Requirements Using Rough Sets

Zhizhong Li and Günther Ruhe

Software Engineering Decision Support Laboratory,
University of Calgary,
2500 University Drive N.W.,
Calgary, Alberta, Canada T2N 1N4
{zhizli, ruhe}@ucalgary.ca

Abstract. Software requirements management is an essential process to better understand, identify, derive, control and improve system requirements. Typically, requirements are unclear at the beginning and evolve over time. Uncertainties usually produce conflicts among requirements. Rough set analysis (RSA) is a promising technique of granular computing. The emphasis of this paper is on formally defining three software requirements uncertainty problems and on applying RSA to solve these problems. A systematic approach called MATARS was developed for that purpose. We use a modification of a real world software requirements specification (SRS) benchmark example to illustrate main concepts and ideas of the approach.

1 Introduction and Motivation

Software requirements engineering (RE) is the process of determining what is to be produced in a software system. Requirements engineering (RE) has evolved into a key issue as one of the most difficult activities in the field of software engineering. The goal of requirements management is the development of a good requirements specification document. The IEEE guide to software requirements specifications [1] defines a good software requirements specification as being unambiguous, complete, verifiable, consistent, modifiable, traceable, and maintainable.

There are at least three challenges that are currently inherent in requirements management: firstly, it needs to transfer informal requirements, which are often vague and imprecise, to formal specification methods [2]; secondly, requirements are often conflicting with each other, and many conflicts are implicit and difficult to identify; thirdly, requirements are changing dynamically.

Rough set analysis (RSA) [3] has attracted the attention of many researchers and practitioners [4]. In applications, rough sets focus on approximate representation of knowledge derivable from data. This leads to significant results in many areas including medicine, finance, industry, multimedia systems or control theory. For an overview we refer to [4]. RSA was applied in software engineering initially in [5] to make sense out of measurement data. Since then, RSA has been successfully applied for data analysis in various areas of software engineering: software maintenance [6], software safety analysis [7], software reverse engineering [8], application run time estimation [9], and knowledge discovery for software quality.

D. Ślęzak et al. (Eds.): RSFDGrC 2005, LNAI 3642, pp. 678 – 687, 2005.

Although RSA is used extensively as an approach to software engineering, it has rarely been applied as a systematic approach for requirements management. An objective of software requirements engineering is to improve systems modeling and analysis capabilities so that critical aspects of systems can be understood prior to system construction. A process is needed to guide the requirements engineer through requirements elicitation. The idea using RSA for requirements analysis was mentioned in [10]. In this paper, we are focusing on uncertainty problems solution and representation of tabular-based requirements.

The paper is organized into five sections. In section 2, we give a formal problem statement that will be later used to apply RSA. Section 3 provides the solution approach. This approach is illustrated in Section 4 for the modified example of A-7E requirements specification. Finally, Section 5 gives a final discussion and outlook to future research.

2 Problem Statement

2.1 Tabular-Based Requirements Management

Requirements can be represented in different ways, ranging from an informal to a more formal description. Tabular-based requirements representation is a special case of formal representation assuming that requirements specification can be done using tables. The goal of requirement management is to develop a requirement specification that contains all the needs of customers [11]. However, as time goes on, requirements change frequently or new requirements arise. Inconsistency or conflicts might result from this process. To check and handle ambiguity, incompleteness and uncertainty is of core importance for later quality of software products.

Definition 1: Tabular-based requirements
A set $R = \{A, B, \ldots\}$ of requirements is said to be in tabular form if each requirement $A \in R$ is described by a set of descriptive and a set of prescriptive attributes:

- Descriptive attributes denoted by $D_1(A)$, … , $D_m(A)$ specify the conditions under which a system behaves.
- Prescriptive attributes, denoted by $P_1(A)$, … , $P_n(A)$ describe how the system should behave under the descriptive conditions $D_1(A)$, … , $D_m(A)$.

Definition 2: Inconsistent requirements
Two tabular-based requirements A and B are called inconsistent to each other if they have the same value for all descriptive attributes, but are different in value for at least one prescriptive attribute. More formally,

If $D_1(A) = D_1(B)$, $D_2(A) = D_2(B)$, …, $D_m(A) = D_m(B)$
There is an attribute $j \in \{1, \ldots, n\}$ so that $P_j(A) \neq P_j(B)$.

Definition 3: Redundancy between requirements
Two tabular-based requirements A and B are said to be redundant if all the descriptive and prescriptive attribute values of requirements A and B are the same. More formally,

$D_1(A) = D_1(B)$, $D_2(A) = D_2(B)$, …, $D_m(A) = D_m(B)$ and
$P_1(A) = P_1(B)$, $P_2(A) = P_2(B)$, …, $P_n(A) = P_n(B)$.

Definition 4: Attribute redundancy
In tabular-based requirements, certain combinations of descriptive attribute values result in certain prescriptive properties. A descriptive attribute D_p with $p \in \{1, \ldots, m\}$ is called redundant if the specification of the whole system remains the same after elimination of descriptive attribute D_p.

2.2 Requirements Uncertainty Problems

Uncertainty is inherent in requirements management. In this paper we will address three types of uncertainty problems as described in the following.

2.2.1 Problem 1: Inconsistency between Requirements

Inconsistency between requirements results in conflicts in the specification of the system behavior. These conflicts are the origin of software failures as typically detected in later stages. There are numerous known efforts to detect and resolve inconsistencies [12].

2.2.2 Problem 2: Missing Data in Requirements

For a tabular-based requirement, lack of information can be related to either prescriptive or descriptive attributes. Under-specification of system behavior is critical as this would force different interpretations on how the system should perform. This could result in unintended actions, causing critical system failures. The question is to suggest those values that would not create inconsistencies with existing requirements.

2.2.3 Problem 3: Redundancy

Requirements redundancy should be avoided as it is useless information. The same is true for attribute redundancy. That means the question is to detect redundancies between requirements as well as redundancies between attributes.

3 Handling Uncertainties by Using Rough Set Analysis

Tabular-based software requirements are described by descriptive and prescriptive attributes. This is very similar to the notion of condition attributes and decision attributes as used in rough set theory. Tabular-based requirements representation is mapped into an information system with the rows corresponding to the requirements and the columns corresponding to the different attributes. Descriptive and prescriptive attributes in tabular-based requirements correspond to condition respectively decision attributes as used in rough set theory.

3.1 Overview of Approach MATARS

Management of tabular-based requirements using Rough Sets (MATARS) is a method for uncertainty handling in the special case of requirements given in tabular

form. MATARS has an underlying process model that is described in detail in [13]. The process here addresses the evolution of requirements as well as the overall process to handle uncertainty related to requirements. Rough set analysis plays a crucial role in MATARS. The main purpose of MATARS is to handle uncertainty in a systematic manner based on formal notation. However, we do not expect that this is completely possible without inclusion of more informal existing conflict resolution approaches as described by Robinson [12].

MATARS is focused on requirements uncertainty handling by RSA combined with existing conflict resolution approaches in requirements engineering. This process is built in order to execute requirements elicitation and resolve inconsistency with the help of RSA during evolution of requirements. The key components of MATARS are shown in Fig. 1. The stakeholders are generating requirements over time. Simultaneously, they are integral part of the resolution method.

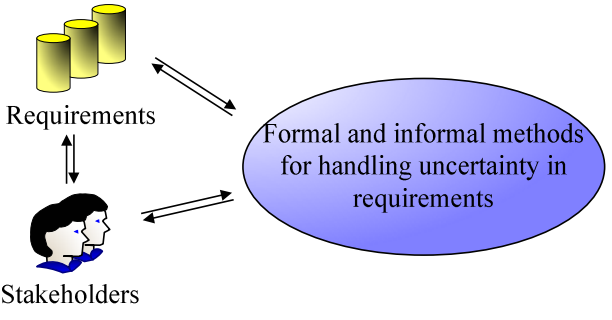

Fig. 1. Main components of MATARS

Uncertainty handling based on formal methods includes the following basic techniques described below.

3.2 Technique 1: Change of Granularity to Resolve Requirements Inconsistency

Change of granularity is one of the fundamental ideas of rough set theory. What we propose is to use this concept for resolution of conflicts in case of tabular-based requirements. Conflicts occur where different classification unions intersect with each other. Change of granularity can help to overcome these conflicts. Higher granularity with respect to one condition attribute can result in conflict resolution. We will illustrate this idea by an example in Section 4.

3.3 Technique 2: Inconsistency Check to Fill Missing Values

Incomplete data is a major problem in data analysis. Pawlak [3] and Gediga [14] present general algorithm to the treatment of missing data with classical RSA. In this paper our focus is specifically on featured tabular-based software requirements, where there are two kinds of missing value problems: missing descriptive values and missing prescriptive values. Suggestions for missing prescriptive values can be generated by learning from already specified requirements. Pawlak [3] gives four possible situations of learning from former examples.

(1) IF the new object matches exactly one of the deterministic rules
(2) THEN the classification suggestion is direct;
(3) IF the new object matches exactly one of the nondeterministic rules
(4) THEN the number (strength) of examples which support each possible category is computed. The considered object most likely belongs to the strongest category;
(5) IF the new object doesn't match any of the sorting rules
(6) THEN a set of rules 'nearest' to the description of the new object is presented;
(7) IF the new object matches more than one rule
THEN the suggestion can be based either on the strength of possible categories or on an analysis of the objects which support each possible category.

Similarly, RSA can also help to complete missing condition values. The process is as follows:

(1) Extend incomplete cases by replacing lost value "?" with all possible values.
(2) Classify all these cases by learning from previously generated rules.
(3) Analyze the classification results with the actual decision values of extended data;
(4) All the values not creating inconsistency are the possible suggestions for missing values.

3.4 Technique 3: Classifying Requirements and Attributes to Remove Redundancies

RSA deals with the classification of requirements and induces minimal decision rules to simplify requirements representation by means of explaining prescriptive attribute values by combination of descriptive attribute values.

Discovering dependencies between attributes enables the reduction of the set of attributes. In RSA, the significance of condition attributes is of three levels.

(1) Core represents the most essential set of condition attributes.
(2) Attributes that belong to reduct are significant to represent whole system.
(3) Redundant attributes are those which do not belong to any reduct. These attributes have no contribution to classification and usually can be removed from information system.

By applying RSA, it is convenient to find condition attributes of these three levels. Redundant attributes have no contribution to classification of whole system thus they should be removed to simplify SRS. In this way we deduce attributes redundancy.

4 Illustrative Example

We consider a modification of a real world requirements benchmarking example [15] to illustrate the uncertainties management in tabular-based requirements. In this paper we use the rough set analysis tool ROSE version 2.2 developed at Technical University Poznan to analyze required tabular data [16]. This RSA tool generates decision rules using a modified version of the LEM2 Algorithm [17].

Table 1 below shows a decision table. There are four possible outside reference points for decision attribute *D*: OAP, 'fly–to point', 'target' and 'none'. Condition attributes include four Boolean attributes denoted by *C*1 to *C*4. In addition, attribute *C*5 which has six possible values (denoted by M1-M6).

Table 1. Decision table for a real world SRS [15]

Record	*C*1	*C*2	*C*3	*C*4	*C*5	*D* (*Decision*)
1	1	1	1	1	M1	None
2	1	1	1	0	M1	None
…	…	…	…	…	…	…
16	0	0	0	0	M1	Fly–to point
17	1	1	1	1	M2	Target
…	…	…	…	…	…	…
48	0	0	0	0	M3	OAP
49	1	1	1	1	M4	Fly-to-point
…	…	…	…	…	…	…
64	0	0	0	0	M4	Target
65	1	1	1	1	M5	Target
…	…	…	…	…	…	…
80	0	0	0	0	M5	None
81	1	1	1	1	M6	Target
82	1	1	1	0	M6	Target
…	…	…	…	…	…	…
95	0	0	0	1	M6	Fly-to-point
96	0	0	0	0	M6	OAP

To better demonstrate the proposed capabilities of MATARS, we slightly modify the original example and assume that the condition attribute *C*5 has only five groups of modes denoted by M1-M5. For that purpose, we change *C*5 value of all the last 16 requirements (from 81st till 96th) from M6 to M5. These values are shaded in Table 1. The IF-THEN rules using RSA are generated and two inconsistent rules are detected out of 14 IF-THEN rules in Table 2:

Table 2. RSA IF-THEN rules for modified SRS

Consistent rules	
(*C*1=0)(*C*5=M1) --->(*D*=Fly-to-point)	(*C*2=1)(*C*5=M2) ---> (*D*=Target)
(*C*1=0)(*C*2=0)(*C*5=M2)-->(*D*=Fly-to-point)	(*C*2=1)(*C*5=M3) ---> (*D*=Target)
(*C*2=0)(*C*3=1)(*C*5=M3)-->(*D*=Fly-to-point)	(*C*3=0)(*C*4=0)(*C*5=M4)---> (*D*=Target)
(*C*3=1)(*C*5=M4) ---> (*D*=Fly-to-point)	(*C*2=1)(*C*5=M5) ---> (*D*=Target)
(*C*4=1)(*C*5=M4) ---> (*D*=Fly-to-point)	(*C*1=1)(*C*5=M1) ---> (*D*=None)
(*C*2=0)(*C*3=0)(*C*5=M3) ---> (*D*=OAP)	(*C*1=1)(*C*2=0)(*C*5=M2) ---> (*D*=None)
Approximate rules	
(*C*2 = 0)(*C*4 = 1)(*C*5 = *M*5) ---> (*D* = Fly-to-point) OR (*D* = None)	
(*C*2 = 0)(*C*4 = 0)(*C*5 = M5) ---> (*D* = OAP) OR (*D* = None)	

And the responding validation results in Table 3:

Table 3. Validation results for modified SRS

CLASS	Number	Lower	Upper	Accuracy
Fly-to-point	32	28	36	0.7778
OAP	8	4	12	0.3333
Target	36	36	36	1.0000
None	20	12	28	0.4286

The shaded area in both tables exposes the inconsistencies, and the accuracy is low. Obviously, if we change the granularity of *C*5 from 5 intervals back to 6, inconsistencies are removed successfully.

Another capability of RSA is the ability to handle missing values of incomplete requirements. Here from the original example in Table 1, 20 requirements were randomly selected, and they are shown in Table 4. The question is: how to complete these new cases based on the existing 76 requirements? In addition to the already introduced changes, we have added records 21 and 22 with missing condition values. All the missing values are highlighted by "?" in Table 4.

Table 4. Incomplete requirements (missing condition and decision values)

Record	*C*1	*C*2	*C*3	*C*4	*C*5	*D*
1	0	1	0	0	M1	?
…	…	…	…	…	…	?
20	1	1	0	1	M6	?
21	0	1	0	0	?	Fly-to-point
22	1	1	1	?	M2	Target

In order to accomplish decision values, we classify the twenty new objects by applying the explanation rules generated from the remaining 76 objects. The classification results are shown in Table 5. The shaded area exposes the 4 requirements (7, 9, 11 and 12) whose predicted values do not match the actual values, in other words, validation of these four cases has failed.

The two incomplete requirements can be extended by replacing lost value "?" with all possible values. Attribute *C*5 has the six levels M1-M6. Thus the first incomplete requirement can be extended to six cases from record 1 to record 6; attribute *C*4 is Boolean variable, thus the second incomplete requirement can be extended by two cases from record 7 to record 8. Table 6 shows the new complete decision table with additional eight cases.

Three shaded records (1, 7, and 8) are successfully validated, thus they are the most reasonable requirements to replace the incomplete ones. The conclusion is satisfying: record 1 and record 8 are the first two requirements in Table 5; record 7 is already inside the 76 cases of modified SRS.

Table 5. Results of ten-fold cross validation

Record	*C*1	*C*2	*C*3	*C*4	*C*5	*D*(Actual)	*D*(Predicted)	Matched Rule
1	0	1	0	0	M1	Fly-to-point	Fly-to-point	4
2	1	1	1	1	M2	Target	Target	9
3	0	0	0	0	M2	Fly-to-point	Fly-to-point	1
4	0	1	0	1	M2	Target	Target	9
5	0	1	0	0	M2	Target	Target	9
6	0	0	0	1	M2	Fly-to-point	Fly-to-point	1
7	0	0	1	1	M3	Fly-to-point	OAP	6
8	0	0	0	1	M3	OAP	OAP	6
9	1	0	1	0	M3	Fly-to-point	OAP	6
10	0	1	0	0	M3	Target	Target	8
11	1	0	1	1	M3	Fly-to-point	OAP	6
12	0	0	1	0	M3	Fly-to-point	OAP	6
13	1	1	1	1	M4	Fly-to-point	Fly-to-point	3
14	0	0	1	0	M4	Fly-to-point	Fly-to-point	5
15	0	1	1	1	M5	Target	Target	11
16	1	1	0	1	M5	Target	Target	11
17	1	1	1	1	M6	Target	Target	10
18	1	1	0	0	M6	Target	Target	1
19	0	0	1	0	M6	OAP	OAP	7
20	1	1	0	1	M6	Target	Target	10

Table 6. Extended decision table and validation results

Record	*C*1	*C*2	*C*3	*C*4	*C*5	*D*(Actual)	*D*(Predicted)	Matched Rule
1	0	1	0	0	M1	Fly-to-point	Fly-to-point	4
2	0	1	0	0	M2	Fly-to-point	Target	9
3	0	1	0	0	M3	Fly-to-point	Target	8
4	0	1	0	0	M4	Fly-to-point	Target	12
5	0	1	0	0	M5	Fly-to-point	Target	11
6	0	1	0	0	M6	Fly-to-point	Target	10
7	1	1	1	0	M2	Target	Target	9
8	1	1	1	1	M2	Target	Target	9

Finally, as part of approach we also apply RSA to simplify a decision table by reducing redundant attributes and requirements. On the basis of original example in Table 1, we added three additional attribute *C*6, *C*7 and *C*8, plus a requirement 97th that is exactly the same as the 96th, as shaded area shown in Table 7.

Firstly, requirement 97th is redundant and it will not affect the IF-THEN rules by RSA. Secondly, from the above table, there exists a single Core = {C1, C2, C3, C4}, together with two reducts: Reduct1 = {C1, C2, C3, C4, C5} and Reduct2 = {C1, C2,

C3, C4, C6, C7}. Attribute C8 does not belong to either reduct, thus it is redundant and can be removed from the SRS. Core = {C1, C2, C3, C4} is the most important set of attributes and each attribute inside the core is necessary to specify the requirements. However, the other three attributes *C*5, *C*6 and *C*7 are only needed in conjunction with specific reducts. For example, if we select Reduct1 then attributes {*C*6, *C*7} are superfluous; if we choose Reduct2 then attribute *C*5 becomes superfluous.

Table 7. Modified SRS with redundancies

Record	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*D*
1	1	1	1	1	M1	0	0	1	None
2	1	1	1	0	M1	0	1	1	None
…	…	…	…	…	…	…	…	…	…
32	0	0	0	0	M2	1	2	1	Fly-to-point
…	…	…	…	…	…	…	…	…	…
96	0	0	0	0	M6	3	5	1	OAP
97	0	0	0	0	M6	3	5	1	OAP

5 Conclusions and Future Work

We have investigated the usage of rough sets for uncertainty handling for tabular-based requirements specification. This approach is part of a more comprehensive method including informal techniques as well. RSA has some fundamental conceptual advantages that can be used for conflict resolution in tabular-based requirements management. The formal approach becomes the more useful, the more complex the underlying table is. In the process of requirements elicitation and specification, RSA plays the role of an intelligent oracle answering the question for the existence of inconsistency and guiding the process to resolve it. This principle was demonstrated by an example using a modified version of the A-7E benchmark SRS.

Future research will be devoted to fully integrate RSA into the process of managing evolving requirements. MATARS is intended to be applied to further examples to validate its applicability.

Acknowledgements

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for its financial support of this research.

References

1. IEEE Guide to Software Requirements Specifications. IEEE Std 830-1984: Software Engineering Technical Committee of the IEEE Computer Society (1984)
2. Liu XF, Yen J: An Analytic Framework for Specifying and Analyzing Imprecise Requirements. In: International Conference on Software Engineering. (1996) 60-69

3. Pawlak Z (ed.): Rough Sets - Theoretical Aspects of Reasoning about Data: Kluwer Academic Publishers. (1991)
4. Komorowski J, Polkowski L, Skowron A: Rough sets: a tutorial. In: Rough-Fuzzy Hybridization: A New Method for Decision Making. Edited by Skowron SKPaA: Springer-Verlag. (1998)
5. Ruhe G: Qualitative Analysis of Software Engineering Data Using Rough Sets. In: 4th International Workshop on Rough Sets, Fuzzy Sets and Machine Discovery (RSFD'96). Tokyo, Japan. (1996) 292-299
6. Morasca S, Ruhe G: A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance. Journal of Systems and Software (2000) 53(3):225-237
7. Chen-Jimenez I, Kornecki A, Zalewski J: Software Safety Analysis Using Rough Sets. In: Proceedings of IEEE SOUTHEASTCON98. IEEE Press. (1998) 15-19
8. Jahnke JH, Bychkov Y: Reverse Engineering Software Architecture Using Rough Clusters. In: Proceeding of the 6th International Conference on Software Engineering & Knowledge Engineering (SEKE'04). (2004) 270-275
9. Krishnaswamy S, Loke SW, Zaslavsky A: Application run time estimation: a quality of service metric for web-based data mining services. In: Proceedings of the 2002 ACM symposium on Applied computing (SAC '02). Madrid, Spain: ACM Press. (2002) 1153-1159
10. Li Z, Ruhe G: Management of Tabular-based Requirements Using Rough Sets. In: Proceedings of the 4th ASERC Workshop on Quantitative and Soft Computing based Software Engineering (QSSE'04). Banff, Alberta, Canada. (2004) 29-34
11. Davis AM (ed.): Software Requirements: Analysis and Specification: Prentice Hall Press. (1990)
12. Robinson WN, Pawlowski SD, Volkov V: Requirements Interaction Management. ACM Comput Surv (2003) 35, no. 2:132-190
13. Li Z: Management of Tabular-based Requirements Using Rough Sets. University of Calgary (2005)
14. Gediga G, Duentsch I: Maximum Consistency of Incomplete Data via Non-Invasive Imputation: Artificial Intelligence Review (2003) 19, no. 1: 93-107
15. Heninger K, Kallander J, Parnas D, Shore J: Software Requirements for the A-7E Aircraft. In. Washington, D.C.: Naval Research Laboratory. (1978)
16. ROSE: Rough Sets Data Explorer [http://www-idss.cs.put.poznan.pl/rose/index.html]
17. Grzymala-Busse JW: LERS - A system for learning from examples based on rough sets. In: Intelligent Decision Support Handbook of Applications and Advances of the Rough Sets Theory. Edited by Slowinski R: Kluwer Academic Publishers. (1992) 3-18