Reconciling Use Cases via Controlled Language and Graphical Models

Kathrin Böttger, Rolf Schwitter, Debbie Richards, Oscar Aguilera, and Diego Mollá

{kathrinb,schwitt,richards,aguilera,molla}@ics.mq.edu.au
Department of Computing
Macquarie University, Sydney, Australia

Abstract. In requirements engineering use cases are employed to describe the flow of events and the occurrence of states in a future information system. Use cases consist of a set of scenarios each of them describing an exemplary behaviour of the system to be developed. Different stakeholders describe the steps in varying ways since they perceive the state of affairs in the application domain from different viewpoints. This results in ambiguous use cases written in natural language that use different terminology and are therefore difficult to reconcile. To solve this problem, use cases and scenarios are rewritten in a controlled language following a simple set of guidelines. The sentences are processed and translated into flat logical forms by the Prolog-based RECOCASE system. These resulting flat logical forms can be used to generate graphical models for the elaboration and refinement of functional requirements between project stakeholders. As an experiment we have chosen Formal Concept Analysis to present the viewpoints of different stakeholders graphically in a concept lattice.

1 Introduction

It is well known that many software projects do not go as well as they are supposed to - and some completely fail. One way to improve software development is to pay more attention to the outcomes of the requirements definition phase in the software development process. Requirements definition aims to establish a shared understanding of all stakeholder requirements.

Conventional requirements capture techniques use a series of interviews to acquire requirements. In interviews users play a relatively passive role. Usually system analysts document the results in specifications described in plain natural language using varying graphical models. These specifications are presented to the users for confirmation but are typically incomplete and inconsistent and do not reflect the real needs of all project stakeholders.

To overcome these problems, viewpoint development has been proposed to improve requirement definitions. Viewpoint development is defined as a process of identifying, understanding and representing different stakeholder viewpoints

(Darke and Shanks 1995). In our viewpoint development approach, several viewpoint agents are identified who play the role of actors for each use case (Jacobson 1992). These agents describe their viewpoints of use cases and scenarios in plain natural language.

To reduce ambiguity and vagueness in use cases written in plain natural language, we propose the use of a controlled natural language that has a well-defined grammar and that comes with a set of simple writing guidelines. The controlled natural language is computer-processable and can be unambiguously translated into flat logical forms. Due to the formal properties of the controlled language the use cases can be checked whether they are consistent with the writing guidelines during the intra-viewpoint analysis phase and during the inter-viewpoint analysis phase the use cases can be compared to identify misunderstandings, inconsistencies and conflicts.

Apart from these formal properties, flat logical forms can be translated automatically into crosstables. Once in crosstable format we use Formal Concept Analysis (FCA) (Wille 1982, 1992) to develop a concept lattice. FCA is a mathematical approach to data analysis based on the lattice theory of Birkhoff (1967). In our approach, the use cases of multiple stakeholders are combined to allow further discussions, identification of similar terminology, integration of viewpoints into one viewpoint, elaboration and refinement of functional requirements.

In Section 2 of this paper we will introduce our controlled language and show how a use case written in plain natural language can be translated into the controlled language version by following a set of simple writing guidelines. In Section 3 we will discuss how the resulting use case can be unambiguously translated into flat logical forms. In Section 4 we show how crosstables can be generated automatically out of these flat logical forms. Crosstables build the starting point to produce concept lattices.

2 Example Use Case

Use cases are usually written in plain natural language. But as we will see even simple sentences with no apparent ambiguities for humans are interpreted as ambiguous by computers that cannot access the relevant knowledge sources. To solve this problem, one could either let the stakeholders disambiguate the sentences or teach them a subset of English that is unambiguously translatable into a formal representation. The first approach is complex and arduous since longer sentences may have hundreds of analyses and interpretations through which the stakeholder would have to go. The second approach also takes some effort since the stakeholders have to learn a set of guidelines about how to specify circumstances in a use case with words. However, we can ease this task by keeping the set of guidelines minimal and by providing a sophisticated interface for writing use cases.

The use case (Gomaa 2000) below is written in plain natural language and contains a number of linguistic problems that need to be solved at some stage if the use case is to be processed by a computer. For each problem we are detecting,

we will formulate a writing guideline that will circumvent the problem in an unambiguous way.

Use Case Name: Withdraw Funds (*in plain natural language*)

Summary: Customer withdraws a specific amount of funds from a valid bank account.
Actor: ATM Customer
Precondition: ATM is idle, displaying a Welcome message.
Description:

1. Customer inserts the ATM Card into the Card Reader.
2. If the system recognizes the card, it reads the card number.
3. System prompts customer for PIN number.
4. Customer enters PIN.
5. System checks the expiration date and whether the card is lost or stolen.
6. If card is valid, the system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, the system checks what accounts are accessible with the ATM Card.
8. System displays customer accounts and prompts customer for transaction type: Withdrawal, Query, or Transfer.
9. Customer selects Withdrawal, enters the amount, and selects the account number.
10. System checks whether customer has enough funds in the account and whether daily limit has been exceeded.
11. If all checks are successful, system authorizes dispensing of cash.
12. System dispenses the cash amount.
13. System prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.
14. System ejects card.
15. System displays Welcome message.

In **sentence (1)** the noun *customer* is used without an article and denotes the same concept as *ATM Customer*. Another potential problem for an automatic processor is the structural ambiguity of the prepositional phrase *into the Card Reader* that modifies here the underlying verbal event and not the object *ATM Card*. The minimal rule set to resolve these problems are:

P1 Use a noun together with a determiner (*customer* $\rightarrow$ *the customer*).
P2 Use words in a consistent way (*the customer* $\rightarrow$ *the ATM customer*).
P3 Use a prepositional phrase to modify a verb (*inserts the ATM Card into the Card Reader*).
P4 Use a relative clause to modify a noun (e.g.: *inserts the ATM Card that has a PIN number*).

In **sentence (2)** the personal pronoun *it* refers back to *system* and not to *card*. Personal pronouns are notoriously difficult to resolve since the search space for the correct noun might be very deep and not enough linguistic information might be available to find the correct antecedent. Therefore, we do not allow personal pronouns in the controlled language:

P5 Use the appropriate noun instead of a personal pronoun (*it* → *the system*).

Sentence (5) expresses that the system checks three conditions but uses only one explicit operator (*whether*). In the controlled language we make the logical dependence between the clauses explicit by logical operators and parallel syntactic structures (... *if A and if B and if C*).

P6 Use logical operators to make the dependence between clauses and phrasal structures explicit and eliminate all the embeddings (*The system checks if the date is expired and if the card is lost and if the card is stolen*).[1]

Sentence (6) uses a passive construction and a compound noun (*card PIN*). In passive constructions the actor is often omitted, therefore we do not allow passive constructions in the controlled language. Another problem is that the compound noun *card PIN* is a combination of two terms that have been introduced before (*PIN number* and *ATM card*).

P7 Use active sentences instead of passive sentences (*PIN maintained by the system* → *The system maintains the PIN number*).[2]
P2 applies again (*card PIN* → *PIN number of the ATM card*).

Sentence (7) uses a plural form. The set of objects described by this plural form (*what accounts*) is underspecified and can be made more explicit by using a determiner (universal quantifier and a singular form).

P8 Use singular instead of plural forms (*the system checks what accounts are accessible with the ATM Card* → *the system checks every account that is accessible with the ATM card*).

Sentence (8) enumerates three transaction types: *Withdrawal, Query, or Transfer.*

P6 applies again (*Withdrawal or Query or Transfer*).

Sentences (9-12) are very problematic since the logical dependences between the clauses are not made explicit. Apart from the missing operators two vague expressions (*enough* and *has been exceeded*) are used that are not precise enough for a specification.

P6 applies again (*If ... then ... if ... then*).

---

[1] Note that the original sentence (5) is not accurate in the sense that it does not tell what to do with the results of the tests. Sentence (6) says *if the card is valid ...*, but sentence (5) does not explicitly say how to determine whether the card is valid. This shows that the original specification is not complete, and there are no rules that can detect this automatically.

[2] There are expressions that denote states, such as *lost* and *stolen* in sentence (5) — the verbs are used as predicative adjectives. In these cases P7 does not apply.

P9 Use a comparative clause to compare specific values (*bigger than the amount X, smaller than the amount Y*).

In **sentence (13)** a noun is modified by a present participle and three noun phrases are enumerated.

P4 applies again (*a receipt that shows the transaction number ...*).

P10 Use commas followed by a comma plus an *and* operator to enumerate more than two noun phrases *(the transaction number, the transaction type, the withdrawn amount, and the account balance*).

If we apply these writing guidelines to the original use case we can rewrite it as shown below. It is important that the viewpoint agent needs only to know these guidelines and no grammar rules as in (Fuchs et al. 1999). The RECOCASE system will automatically flag all inadmissible grammatical structures.

Use Case Name: Withdraw Funds (*in controlled natural language*)

Summary: The ATM customer withdraws a specific amount of funds from a valid bank account.
Actor: ATM customer
Precondition: If the ATM is idle then the system displays a Welcome message.
Description:

1. The ATM customer inserts the ATM card into the card reader.
2. If the system recognizes the ATM card then the system reads the card number.
3. The system prompts the ATM customer for the PIN number.
4. The ATM customer enters the PIN number.
5. The system checks if the date is expired and if the ATM card is lost and if the ATM card is stolen.
6. If the ATM card is valid then the system checks if the PIN number matches the PIN number of the ATM card.
7. If the PIN number matches the PIN number of the ATM card then the system checks every account that is accessible with the ATM card.
8. The system displays every customer account and prompts the ATM customer for the transaction type: Withdrawal or Query or Transfer.
9. If the ATM customer selects the transaction type Withdrawal and enters the amount and selects the account number then the system checks if the funds of the ATM customer is bigger than the amount X and if the daily limit of the ATM customer is smaller than the amount Y and then the system dispenses the cash amount.
10. The system prints a receipt that shows the transaction number, the transaction type, the withdrawn amount, and the account balance.
11. The system ejects the card.
12. The system displays a Welcome message.

The RECOCASE system takes this use case as input and produces for each sentence a flat logical form.

3 From Use Cases to Flat Logical Forms

The RECOCASE system is a Prolog implementation that uses Link Grammar parser (LG) (Sleator & Temperlay 1993) to parse the use case and an extension of ExtrAns' logical form generator (Mollá et al. 2000) is used to produce flat logical forms. LG consists of a fast parser and a grammar of English written in the spirit of dependency grammar showing the words that are linked and the types of links. Since the original LG parser outputs all the alternative dependency structures for a sentence, we use a filter that only accepts dependency structures that are defined in our controlled language. If RECOCASE discovers a sentence that is not in the subset of the controlled language it displays a message and informs the user about its coverage. From the dependency structures RECOCASE derives a flat logical form as a semantic representation for each sentence. Flat logical form consists of a conjunction of predicates where all variables are existentially closed. To make this notation expressive enough, the logical form generator uses reification for objects, events, properties, and operators.

For example, the sentence

The ATM customer inserts the ATM card into the card reader.

results in the following flat logical form:

```
holds(e4)
object(customer,o1,[x3])
compound_noun(x2,x3)
object('ATM',o2,[x2])
evt(insert,e4,[x3,x7])
object(card,o3,[x7])
compound_noun(x6,x7)
object('ATM',o2,[x6])
prop(into,p8,[e4,x11])
object(reader,o5,[x11])
compound_noun(x10,x11)
object(card,o6,[x10])
```

The compound noun *ATM customer* introduces three predicates:

```
object(customer,o1,[x3])
compound_noun(x2,x3)
object('ATM',o2,[x2])
```

The meaning of the first predicate `object(customer,o1,[x3])` is "`o1` is the concept that the object `x3` is a customer" and the meaning of the third predicate `object('ATM',o2,[x2])` is "`o2` is the concept that the object `x2` is an ATM". The second predicate `compound_noun(x2,x3)` says that the objects `x2` and `x3` stand in a compound noun relation that is not further specified.

The verb *inserts* introduces the predicate

```
evt(insert,e4,[x3,x7])
```

with the meaning "e4 is the event that x3 inserts x7". x3 and x7 represent the objects introduced by the arguments of the verb. The reification of the event e4 provides a handle that can be used to modify this event.

The prepositional phrase *into the card reader* introduces four predicates: the predicate

```
prop(into,p8,[e4,x11])
```

deduced from the complete prepositional phrase and three predicates deduced from the compound noun *card reader*. Prepositions introduce properties: the meaning of the above predicate is "p8 is the property that x11 modifies e4".

Reification can also be used to encode the existence of concepts and logical operators. To express that an event actually exists the predicate `holds(e4)` is used. All logical operators that occur in the controlled language are reified and represented in the following way: `if(op1,e1,e2)`, `and(op2,[e1,e2])`, `or(op3,[e1,e2])`, `not(op4,e1)`. Nested logical expressions can be flattened-out by using the reification of the logical operators as handles. Thus, the expression "`and(x,or(y,z))`" is converted into `and(op1,[x,op2])`, `or(op2,[y,z])`.

By using flat logical forms we can avoid embedded structures, this has the nice effect that the logical forms of two use cases are easy to compare and to work with.

4 From Flat Logical Forms to Concept Lattices via Crosstables

Graphical models have been recognized as useful communication mediums between project stakeholders. We have chosen to use FCA to present the viewpoints of different stakeholders as a concept lattice. We were attracted to FCA for the problem of reconciling differences in viewpoints since a concept in FCA is based on the philosophical understanding of a concept as a set of objects and the set of attributes shared by that object, known as the intent and extent of the concept, respectively. This means that similar concepts and differences in terminology should be identifiable either through their extensional or intensional definition. As a graph, the lattice also allows us to compute the closeness between viewpoints and to test when we are moving towards a shared viewpoint. To generate a concept lattice using FCA we begin with a crosstable that can automatically be generated from the flat logical forms.

4.1 Example

A crosstable is made up of rows of objects (sentences) and columns of attributes (terms) used by those objects (see Table 2 below). As an example we translate the logical forms for the sentence

The ATM customer inserts the ATM card into the card reader.

into a row in the crosstable. The predicate `holds(e4)` of the logical form of this sentence refers to the event `(insert)` as the main event. We create an attribute `(insert)` for this main event. The components, which are directly connected with the main event, are the objects `(customer)` and `(card)` and the preposition `(into)`. In a recursive way we are looking for other connected components. `(customer)` and `(card)` are only connected with `(ATM)` as compound nouns. Since `(customer)` and `(card)` are directly connected with the main event, we connect each of them with the components with which they are connected recursively and create thus the attributes of the crosstable `(ATM customer)` and `(ATM card)`. The preposition `(into)` is connected with `(reader)` which is connected to `(card)` to build a compound noun. This way we get the prepositional phrase `(into card reader)` as the fourth attribute. Thus the final attributes of the object 'sentence 1' are:

s1: (ATM customer), (insert), (ATM card), (into card reader)

Using the algorithm in (Böttger forthcoming) we get the following attributes for sentences (2-5). Each of these sentences defines an object and thus a row in the crosstable.

s2: (system), (read), (card number), (if system recognizes ATM card), (then)
s3: (system), (prompt), (ATM customer), (for PIN number)
s4: (ATM customer), (enter), (PIN number)
s5: (system), (check), (if expired date), (if anonym_object lose ATM card), (if anonym_object steal ATM card)

Using sentences (1-5) we get the following columns (Table1) for the crosstable:[3]

Table 1. Columns for crosstable in Table 2

1 ATM customer	**7** card number	**13** PIN number
2 insert	**8** if system recognizes ATM card	**14** check
3 ATM card	**9** then	**15** if expired date
4 into card reader	**10** prompt	**16** if anonym_object lose ATM card
5 system	**11** for PIN number	**17** if anonym_object steal ATM card
6 read	**12** enter	

[3] RECOCASE does not have enough lexical and world knowledge to tell that *lost* and *stolen* denote states, and therefore it produces the active form of the expressions.

Table 2. Crosstable for sentences (1-5)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
s1	x	x	x	x													
s2					x	x	x	x	x								
s3	x				x					x	x						
s4	x											x	x				
s5					x									x	x	x	x

By finding intersections of shared attributes we are able to develop higher-level concepts. By ordering the concepts we are able to create an abstraction hierarchy in the form of a complete lattice (see Figure 1). For further discussion regarding FCA and the generation of concepts and concept lattices please refer to Wille (1992).

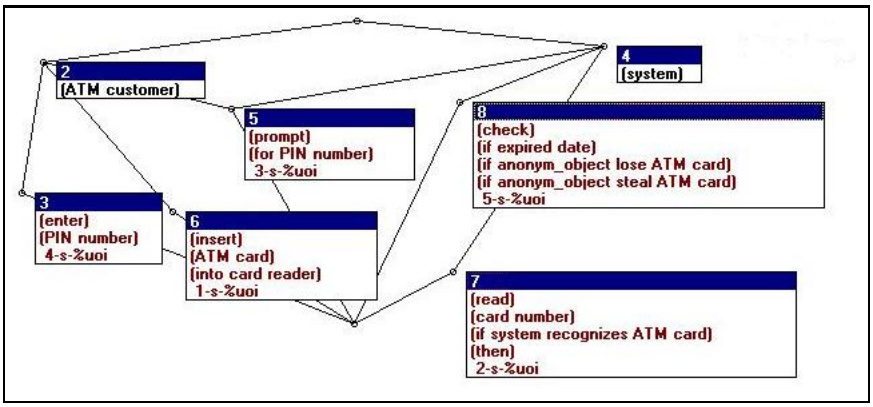

Fig. 1. Concept lattice of sentences (1-5)

5 Further Work

We are currently investigating how to represent the relations between the attributes. A possibility is to tag the attributes in the objects and add expressions about the nature of the relations. By adding such additional information the attributes can be simplified and thus it is more likely to find shared attributes. For example `(for PIN number)` would be converted into `(PIN number)` and a new concept (say, Concept 9) would be generated. Figure 2 shows the necessary changes in the affected concepts to represent who prompts whom for what and who enters what (the relations between the concepts are not shown).

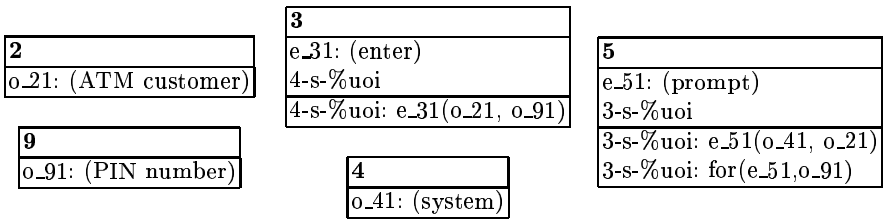

Fig. 2. Possible changes in some nodes of the concept lattice in Figure 1

6 Conclusion

The RECOCASE system is a Prolog implementation that translates use cases written in controlled language into flat logical forms. If the RECOCASE system discovers a sentence that is not in the subset of the controlled language it informs the user about its coverage. It is then the task of the user to rewrite the sentence according to the guidelines of the controlled language. The flat logical forms of the use cases can automatically be translated into crosstables. Once in crosstable format we use Formal Concept Analysis to develop a concept lattice to reconcile differences in viewpoints. Lattices allow to compute the closeness between viewpoints and to test when we are moving towards a shared viewpoint.

References

Birkhoff, G.: Lattice Theory. American Mathematical Society. Providence, Rhode Island. 1967.

Böttger, K: Modelling and Reconciling Functional Requirements from Different Viewpoints Using Use Case / Scenarios and Formal Concept Analysis. University of Mannheim, Germany. 2001.

Darke, P., Shanks, G.: Managing user viewpoints in requirement definition. 8^{th} Australasian Conference on Information Systems. 1995.

Fuchs, N. E., Schwertel, U., Schwitter, R: Attempto Controlled English — Not Just Another Logic Specification Language. Lecture Notes in Computer Science 1559. Springer Verlag. 1999.

Gomaa, H.:Withdraw funds (example use case), in Matthews, M.G. Object-Oriented Analysis and Modeling. http://mason.gmu.edu/~mmatthe1/ObjectOrientedAnalysis.pdf. 2001.

Jacobson, I.: Object-Oriented Software Engineering. Addison-Wesley. 1992.

Mollá, D., Schwitter, R., Hess, M., Fournier, R.: ExtrAns, an answer extraction system. T.A.L **41**:2 (2000) 495–519.

Sleator, D. D., Temperley, D.: Parsing English with a link grammar. Proceedings of the Third International Workshop on Parsing Technologies, pp. 277–292. 1993.

Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In Reidel, D. Ordered Sets, Dordrecht, pp. 445–470. 1982.

Wille, R.: Concept lattices and conceptual knowledge. Computers and Mathematics with Applications **23** (1992) 493–522.