Representing Requirements in Natural Language as Concept Lattices

Debbie Richards and Kathrin Boettger
Department of Computing,
Division of Information and Communication Sciences,
Macquarie University, Sydney, Australia
{richards@ics.mq.edu.au}

August 5, 2002

Abstract

We have developed a *viewpoint development* approach to identify and reconcile differences between stakeholder requirements. The initial phase in our approach seeks to provide a formal solution to the problem of converting requirements descriptions in natural language into a computer processable representation. After the group brainstorms the functional requirements in the form of *use cases*, natural language descriptions are entered by individual stakeholders for each alternative viewpoint. LinkGrammar is used by ExtrAns to translate the use case sentences into flat logical forms (FLFs). FLFs are used to create crosstables. Formal Concept Analysis uses the crosstables to develop a graphical representation of the viewpoints and to assist comparison of terms and concepts. We call our approach RECOCASE as we offer a CASE (computer aided software engineering) tool to assist with viewpoint RECOnciliation. This paper focuses on the translation of natural language into crosstables to allow the generation of *concept lattices* and subsequent comparison of viewpoints.

1 Overview

Many software projects do not go as expected [2, 13]. One critical part of the software development process is the definition of requirements. *Requirements definition* aims to establish a shared understanding of all stakeholder requirements. *Viewpoint development* has been proposed (e.g. [3, 4, 9]) as one possibilty to gain a more representative, complete and consistent shared understanding. To capture viewpoints we capture use cases in natural language as this approach is becoming well accepted for requirements specification and humans find specification of requirements in natural language more natural and manageable than in a formal representation. However we want to do some reasoning and rerepresentation of the requirements using a computer to allow identification and reconciliation of differences between viewpoints.

We start with brainstorming the main chunks of functionality from the user's point of view in the form of *use cases* [6]. Next, viewpoints of functional requirements are captured from individual stakeholders as they enter natural

language descriptions of their requirements for each *use case* and/or *scenario*. LinkGrammar is used by ExtrAns to translate natural language sentences into flat logical forms (FLFs). FLFs are used to create *crosstables*. Formal Concept Analysis (*FCA*) [14, 15] is used to develop a graphical representation of the viewpoints. Apart from the potential benefits of a visual representation and its popularity for modelling object-oriented systems, we were attracted to *FCA* for this task as we wanted to allow users to enter requirements in their own words and then perform comparison of terms and concepts. We then apply the various resolution strategies and operators that are part of our framework to develop a shared conceptual model of the requirements.

We call our approach RECOCASE as we offer a CASE (computer aided software engineering) tool to assist with viewpoint RECOnciliation. Our framework is not discussed in this paper as it was introduced in [10, 11]. This paper focuses on the translation of natural language into crosstables to allow generation of *concept lattices* using *FCA*.

2 Introducing the Process and the Foundational Concepts

To translate use cases in natural language into concept lattices we use the following process.

1. Each viewpoint agent enters their use case description into the RECOCASE-tool
2. The sentences are passed to ExtrAns, converted into FLFs and returned
3. The FLFs are processed using RECOCASE-logic to form verb and noun phrases
4. The sentences and phrases become the objects and attributes of a formal concept and represented as a crosstable
5. The crosstable is used by FCA to generate a concept lattice

An introduction to the basic concepts and theories used in this process are given next. Use cases and scenarios are introduced in section 2.1. LinkGrammar and ExtrAns are described in sections 2.2 and 2.3 respectively. *FCA* is described in section 2.4.

2.1 The Use Case Concept and the Scenario Concept

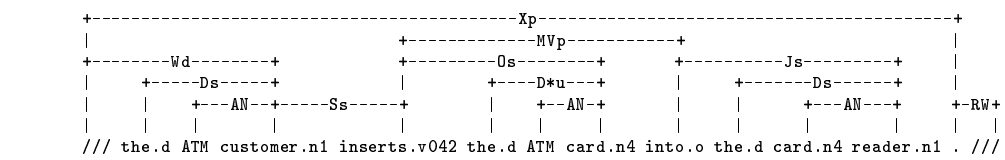
Ivar Jacobson was the first who applied the concept of *use cases* to software development as part of his object-oriented software engineering (OOSE) method. Following [6], a *use case* represents a complete course of events in a system from the user's perspective and describes the interaction between the system and an *actor*. An *actor* is a role played in relation to the system and can include an

individual, groups, another system or hardware device. Using the terminology of object-oriented software development *scenarios* are instances of *use cases*. A *scenario* is a concrete, focused and informal description of one possible behaviour of the system interacting with an *actor*. *Scenarios* are formalized into *use cases*.

Use case and *scenario* descriptions provide a textual description of such things as *use case* name, *actors*, preconditions, postconditions, trigger, main flow and alternative flows. Our work is primarily concerned with the main flow which is the step-by-step sequence of actions from trigger to achievement of postconditions. The textual description of the main flow is translated into FLFs using Link Grammar and ExtrAns.

2.2 LinkGrammar

LinkGrammar is a parser of English based on the link grammar theory. It returns all alternative syntactic dependencies between the words of a sentence. The syntactic structure consists of a set of labeled links connecting pairs of words. To find a syntactic structure LinkGrammar uses a dictionary which contains linking requirements of words. The words of a syntactic structure are connected in such a way, that the links satisfy the linking requirements for each word of the sentence (satisfaction), that the links do not cross (planarity) and that all words form a connected graph (connectivity) [12]. For example LinkGrammar provides the following syntactic structure for the sentence 'The ATM customer inserts the ATM card into the card reader.'



'S' connects the noun 'customer' to the finite verb 'insert', which is a transitive verb and again connected through 'O' with its object 'card'. 'AN' connects the noun-modifier 'ATM' with the noun 'customer'. The object 'card' is connected also through 'AN' with the noun modifier 'ATM'. 'D' connects determiners to nouns. 'customer', 'card' and 'reader' are in this way conntected with the determiner 'the'. 'MV' connects verbs and adjectives to modifying phrases which follow the verb like adverbs, preprositional phrases and subordinating conjunctions. In this example the verb 'insert' is connected through 'MV' with the preposition 'into', which introduces a prepositional phrase. To the right the preposition 'into' is connected to its object 'reader' through 'J'.

2.3 ExtrAns

ExtrAns is an answer extraction system, extracting answers from a collection of technical documents. The documents are first preprocessed by a tokeniser to

separate the sentences. In a second step ExtrAns uses LinkGrammar to get the syntactic structures of the document sentences. Then a pruner filters obviously wrong structures using a set of rules for the application words and domains. Since LinkGrammar does not carry out any morphological analysis a lemmatiser generates the lemmas of the inflected word forms. A disambiguator which is trained with data of the application domain is then used to resolve ambiguities. This disambiguator uses statistical knowledge in contrast to syntactic information which are used to resolve pronominal anaphors in a following step. These partially disambiguated dependency structures are used by ExtrAns to create FLFs as semantic representation for the core meaning of each sentence. For processing reasons these FLFs are translated into Prolog facts. The query is also translated into its logical form and converted into a Prolog query which is run against the facts of the sentence of the technical documents to retrieve the answer [8].

object	The predicate 'object(customer,o1,[x1])' is introduced by the noun 'customer' with the meaning that 'o1 is the concept that the object x1 is customer'. The concept 'o1' can then be used in construction with adjectives or in expressions of identity.
event	The predicate 'evt(insert,e1,[x1,x2])' is introduced by the verb 'insert' and means that 'e1 is the concept that x1 inserts x2'. The concept 'e1' can then be used to express modification by adverbs, prepositional phrases, etc.
property	The predicate 'prop(into,(p1,[e1,x1])' is introduced by the preposition 'into' to describe the concept 'p1' which means 'p1 is the concept that the event e1 is connected with x1 by into'. Properties can also be introduced by adjectives and adverbs.

These abstract concepts are used in other predicates of the FLFs.

compound noun	The predicate 'compound_noun(x1,x2)' connects 'x1' and 'x2' as compound nouns.
genitive	The predicate 'genitive(x1,x2)' stands for 'x1's x2'.
holds	The predicate 'holds(e1)' means that the event 'e1' actually exists. No information is given in those cases where ExtrAns has not enough information to assume the existence. 'holds' can also be connected with the concepts 'property' and 'object'.
if	The predicate 'if(op1,x1,x2)' describes the concept 'op1' as 'x2 if x1'.
and	The predicate 'and(op2,[x1,x2])' represents the logical operator 'and' and defines the concept 'op2' as 'x1 and x2'.
or	The predicate 'or(op3,[x1,x2])' represents the logical operator 'or' and defines the concept 'op2' as 'x1 or x2'.
not	The predicate 'not(op4,x1)' represents the logical operator 'not' and describes the concept 'op4' by 'not x1'.

FLFs consist of a conjunction of predicates where all variables are existentially closed. consist of a conjunction of predicates. To make this logical form expressive enough, the logical form generator of ExtrAns uses the abstract concepts *object*, *event*, and *property*.

2.4 Formal Concept Analysis

Formal Concept Analysis was founded by Rudolf Wille in 1980 at the TH Darmstadt, Germany. *FCA* is a mathematical approach to data analysis based on the based on the lattice theory of Birkoff [1]. *FCA* is usually used for data analysis tasks to find, structure and display relationships between concepts, which consist of attributes and objects. Thus this method assists development of domain models.

A *formal context* is a triple (G, M, I). G is a set of objects, M a set of attributes and I a binary relation $I \subseteq G \times M$ between objects and attributes. If $m \epsilon M$ is an attribute of $g \epsilon G$ then $(g, m) \epsilon I$ is valid. A *crosstable* is a typical representation of a *formal context* in tabular form. The rows represent the objects and the columns represent the attributes. A cell is marked if the particular object has the corresponding attribute.

A *formal concept* (A,B) of the *formal context* (G, M, I) is defined as a pair of objects $A \subseteq G$ and attributes $B \subseteq M$ with $B := \{m \epsilon M | \forall g \epsilon A : (g, m) \epsilon I\}$ and $A := \{g \epsilon G | \forall m \epsilon B : (g, m) \epsilon I\}$. A natural *subconcept / superconcept* relationship $\leq$ of *formal concepts* can be defined as $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ or, equivalently $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow B_2 \subseteq B_1$. Thus a *formal concept* $C_1 = (A_1, B_1)$ is a *subconcept* of the *formal concept* $C_2 = (A_2, B_2)$ if A_1 is a subset of A_2 or B_1 is a superset of B_2. C_2 is called *superconcept* of *formal concept* C_1.

A *concept lattice* is an algebraic structure with certain infimum and supremum operations. The set of concepts and their relations can be visualised as a line diagram. It allows the investigation and interpretation of relationships between *concepts*, objects and attributes. The nodes of the graph represent the *concepts*. Two *concepts* C_1 and C_2 are connected if $C_1 \leq C_2$ and if there is no *concept* C_3 with $C_1 \leq C_3 \leq C_2$. Although it is a directed acyclic graph the edges are not provided with arrowheads. Instead the convention holds that the *superconcept* always appears above its *subconcepts*. The top element of the graph is the *concept* (G, G') and the bottom element the *concept* (M', M). To find all attributes belonging to an object, one starts with the node which represents the object, and follows all paths to the top element. In this way one also finds all *superconcepts* of a *concept*. To find all *subconcepts* of a *concept* one follows all paths to the bottom element. Our application of *FCA* to natural language and use cases is novel. In order to apply *FCA* we need to determine the objects and attributes of the *crosstable*. We consider this next.

3 FLF Translation into Crosstables

Words and morphemes are the smallest meaningful units in language, but for the most part, humans communicate in phrases and sentences with each other [5]. Therefore this approach considers each sentence as an object. The question is now, what are the attributes of a sentence ?

If RECOCASE creates one attribute for each predicate representing a word of the sentence then the *concept lattice* would not be readable in most cases. As an example the *concept lattice* of the three sentences:
1. 'The customer inserts the card in the ATM.'
2. 'The ATM checks if the card is valid.'
3. 'The ATM gives a prompt for the code to the customer.'
where an attribute is created for each predicate of the FLFs, is given in figure 1.

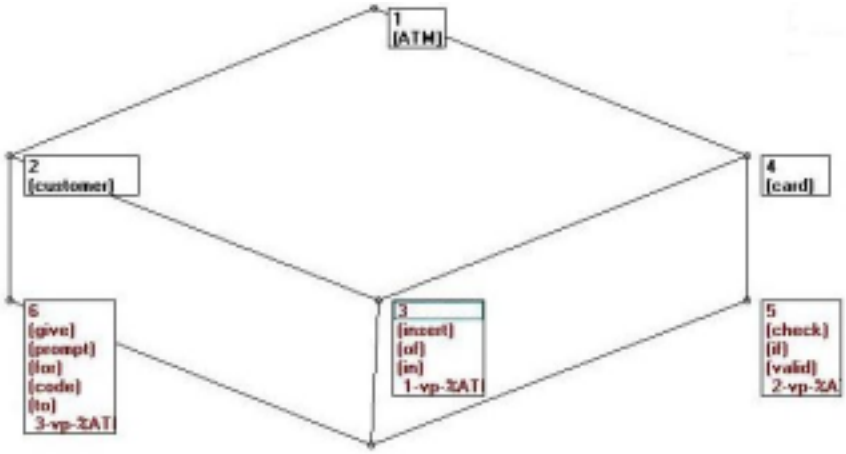


Figure 1: Detailed Representation of Three Simple Natural Language Sentences

Table 1: Crosstable for sentences 1-3

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
s1	x	x	x	x						
s2					x	x	x			
s3					x			x	x	x

Table 2: Columns for crosstable in Table 1

1 customer	**5** ATM	**9** prompt for code
2 insert	**6** check	**10** to customer
3 card of customer	**7** if valid card	
4 in ATM	**8** give	

To make the lattice more readable and assist reconstruction of the sentence we want to group words into phrases. To achieve this we transpose the FLFs into word phrases using what we call RECOCASE-logic. For example the algorithm used to create these word phrases and described in the following part of this section transposes the FLFs of the sentence 'The customer inserts the card in the ATM.' into the phrases 'customer', 'insert', 'card of customer' and 'in ATM'. Thus the sentences/steps of the use case description are the objects and the phrases are the attributes. Table 1 shows the crosstable for sentences 1-3 produced using RECOCASE-logic. Figure 2 shows the *line diagram* based on the crosstable in Table 1. We can see a much clearer structure in figure 2 than we could see for the same sentences in figure 1 which simply took each word individually.

The translation of FLFs into crosstable attributes follows an algorithm which uses graph theory. The algorithm contains two main parts. The first part includes building a graph representing the FLFs of a sentence. The nodes of the graph represent the

Table 3: Relations between Words or Word Phrases of the Predicates

#	nodes	relation between nodes	relation derived from	nodes combined to
1	A, B	compound_noun(A,B)	compound_noun(A,B)	$A $B
2	A, B	genitive(A,B)	genitive(A,B)	$A's $B
3	A, C	prop (A, ...,[C])	prop (A,B,[C])	$C $A
4	A, C	prop (A, ...,[C,...])	prop (A,B,[C,D])	$C $A ...
5	A, D	prop (A,...,[...,D])	prop (A,B,[C,D])	$C ...$A
6	A, C	evt(A,...,[C])	evt(A,B,[C])	$C $A
7	A, C	evt(A,...,[C,...])	evt(A,B,[C,D])	$C $A ...
8	A, D	evt(A,...,[...,D])	evt(A,B,[C,D])	...$A $D
9	if, A	if(...,B)	if(A,B)	if $B ...
10	if, B	if(A, ...)	if(A,B)	if ...$A
11	while, B	while(..., B)	while(A,B)	while $B ...
12	while, A	while(A, ...)	while(A,B)	while ...$B
13	until, B	until(..., B)	until(A,B)	until $B
14	until, A	until(A, ...)	until(A,B)	$A until
15	A, B	and(A,B)	and(A,B)	$A and $B
16	A, B	or(A,B)	or(A,B)	$A or $B
17	not, A	not(A)	not(A)	not $A

words of the FLF predicates and the edges represent the relations between them. The relation between words of the predicates is derived from the FLFs. At a coarse level the algorithm, called RECOCASE-logic, is looking for the main event, object or property which is defined by the predicate 'holds'. If the FLF of a sentence does not contain the predicate 'holds', the root 'A' is defined through 'if(A,B)', 'while(A,B)' or 'not(A)' in this sequence. For the main event, object or property a node is created as root. RECOCASE-logic is looking for predicates of the FLF which are connected with the main event, object or property and creates nodes representing the words of the predicates. Table 3 lists possible relations and the predicates which they are derived from. If all predicates, which are connected with the main event, object or property, are investigated each node of the graph is considered in the same way. By this RECOCASE-logic extends the graph until all nodes are considered. The second part includes the reduction of the graph to nodes representing the crosstable attributes. In a certain sequence all edges are considered and if possible the nodes which are connected by these edges combined. Thus RECOCASE-logic reduces the graph. The final nodes represent the crosstable attributes.

The algorithm uses the fact that the FLFs can be converted into a graph where all nodes are connected. This is based on Melcuk's [7] proof of the fact that all words for sentences of most natural languages form a connected graph, which is used by LinkGrammar to find the syntactic structure of a sentence.

In the following section the algorithm is described in pseudocode. A, B, C and D are variables which stand for the words of the FLF predicates or for the created phrases as a result of the reduction of the graph, respectively. $A stands for the content of the variable A. A := $B means that the content of variable B is assigned

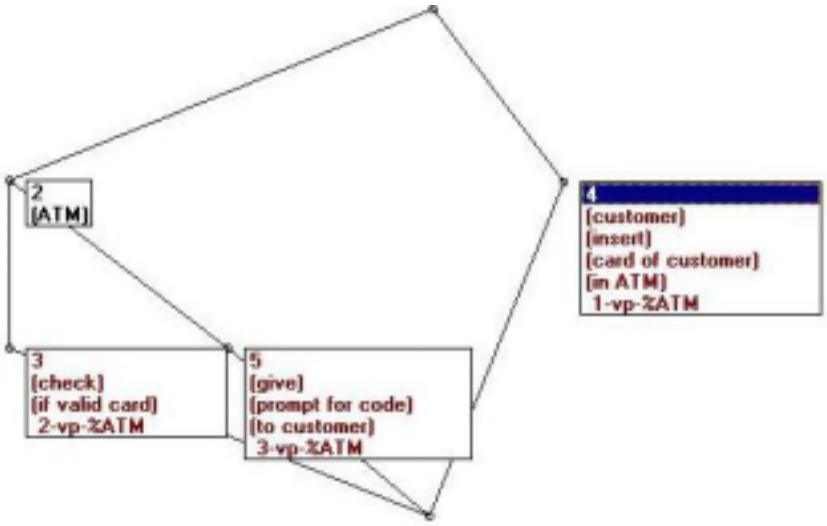


Figure 2: Representation of sentences in RECOCASE

to the variable A.

The translation from FLF into crosstable representation involves the following steps:

```
for each sentence
  % === build graph

    define empty graph G

    % look for root
    if ('holds(A)' exists in FLF)
      root := $A
    else if ('if(A,B)' exists in FLF)
      root := $A
    else if ('while(A,B)' exists in FLF)
      root := $A
    else if ('not(A)' exists in FLF)
      root := $A

    add root to G
    while (not (all knots of graph G are considered))
      K =  knot of G, which is not considered yet
      while (component C exists, which is connected with K in FLF)
        add C as knot to G
```

```
      connect C with K
      mark kind of relation at edge
          % kind of relations: if, while, not, property (prop),
          %                    object (object),
          %                    event (event), compound noun, genitiv
      mark C as considered

% === create attributes for crosstable

  for all edges
    connect knots of edge in the following way
      % this means to reduce the graph
      for all edges which are of type compound_noun(A,B)
        B := $A + $B

      for all edges which are of type genitive(A,B)
        A := $B $A

      for all edges which are of type not(A)
        A := 'not' + $A

      for all edges which are of type prop (A,B,[C])
        if (not (A is the root and A is an event))
          A := $A + $C
      for all edges which are of type prop (A,B,[C,D]) connecting A and D
        D := $A + $D
      for all edges which are of type prop (A,B,[C,D]) connecting A and C
        if (not (C is the root and C is an event))
          C := $C + $A

      for all edges which are of type evt(A,B,[C])
        if (A is not the root)
          A := $C + $A
      for all edges which are of type evt(A,B,[C,D]) connecting A and D
        if (A is not the root)
          A := $A + $D
      for all edges which are of type evt(A,B,[C,D]) connecting A and C
        if (A is not the root)
          C := $C + $A

      for all edges which are of type if(A,B) connecting if and B
        if := 'if' + $B
      for all edges which are of type if(A,B) connecting A and if
        if (not (A is the root))
          A := $A + 'if'

      for all edges which are of type while(A,B) connecting while and B
        while := 'while' + $B
      for all edges which are of type while(A,B) connecting A and while
        if (not (A is the root))
```

```
        A := $A + 'while'

    for all edges which are of type until(A,B) connecting until and B
      until := 'until' + $B
    for all edges which are of type until(A,B) connecting A and until
      if (not (A is the root))
        A := $A + 'until'

    for all edges which are of type and(A,[B1,B2,...,Bn])
      if (not (A is the root))
        A := $B1 + 'and'
         ...
        A := $Bn
    for all edges which are of type or(A,[B1,B2,...,Bn])
      if (not (A is the root))
        A := $B1 + 'or'
         ...
        A := $Bn

  for all knots of G
    create attribute
```

ExtrAns provides the following FLF for the sample sentence 'The ATM customer inserts the ATM card into the card reader.' :

```
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])
```

Figure 3 shows the steps involved in creation of the graph for this sample sentence.

Ia) RECOCASE-logic finds 'insert' as main event through the predicates 'holds(e4)' and 'evt(insert,e4,[x3,x7])' and creates the first node e4:'insert', called root or main node of the graph, which is the starting point for all further activities.

Ib) RECOCASE-logic looks for nodes which are directly connected with the root. The predicates 'evt(insert,e4,[**x3**,x7])' and 'object(customer,o1,[**x3**])' give the information that e4:'insert' has to be connected to x3:'customer' by 'evt(e4,...,[x3,...]'. A node x7:'card' has to be connected to the root because of the predicates 'evt(insert,e4,[x3,**x7**])' and 'object(card,o3,[**x7**])' which refers to the relation 'evt(e4,...,[...,x7])'. The predicate 'prop(into,(p8,[e4,x11])' connects a node into:'into' with e4:'insert'. The relation between them is marked as 'prop (into, ...,[e4,...])'.

Ic) RECOCASE-logic considers the node x3:'customer', which has to be connected with a node x2:'ATM' because of 'object(customer,o1,[**x3**])', 'compound_noun(**x2**,**x3**)'

and 'object(ATM,o2,[**x2**])'. The relation to the node x3:'customer' is marked as 'compound_noun(x2,x3)'.

Id) RECOCASE-logic considers the node x7:'card', which has to be connected with a node x6:'ATM' because of the predicates 'object(card,o3,[**x7**])', 'compound_noun (**x6**,**x7**)' and 'object(ATM,o2,[**x6**])'. The relation to the node x7:'card' is marked as 'compound_noun(x6,x7)'.

Ie) RECOCASE-logic considers the node into:'into' which has to be connected to a node x11:'reader' because of the predicates 'prop(into,(p8,[e4,**x11**])' and 'object(reader,o5,[**x11**])'. The relation is marked as 'prop (into,...,[...,x11])'.

If) RECOCASE-logic considers the node x11:'reader'. It has to be connected with a node x10:'card' because of the predicates 'object(reader,o5,[**x11**])', 'compound_noun(**x10**,**x11**)' and 'object(card,o6,[**x10**])'. The relation is marked as 'compound_noun(x10,x11)'.

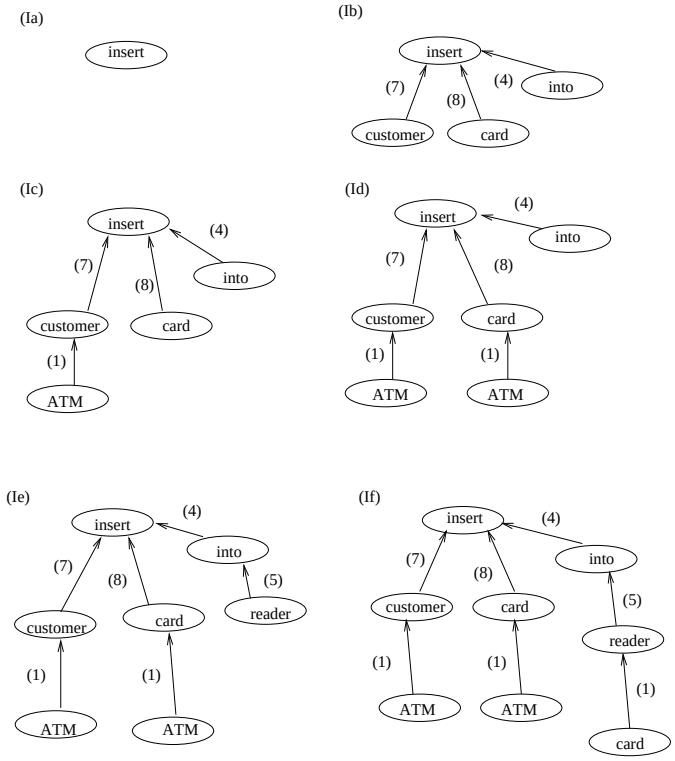


Figure 3: Step of Creation the Graph of the Sentence 'The ATM customer inserts the ATM card into the card reader.'

In a second step RECOCASE-logic reduces the graph to the crosstable attributes. Figure 4 shows the steps involved in the reduction of the graph to crosstable attributes for the sample sentence.

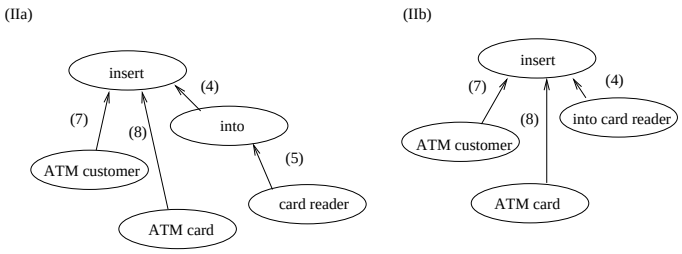


Figure 4: Step of Reducing the Graph of the Sentence 'The ATM customer inserts the ATM card into the card reader.'

IIa) RECOCASE-logic considers all edges which are of type 'compound_noun(A,B)' and combines the nodes which are connected through these edges to the node B:'A B'. Thus the nodes x3:'customer' and x2:'ATM' get combined to x3:'ATM customer', the nodes x7:'card' and x6:'ATM' to x7:'ATM card' and the nodes x10:'card' and x11:'reader' to x11:'card reader'. Since x2:'ATM', x7:'ATM' and x11:'card' are not connected with other nodes these nodes are deleted.

IIb) Following the algorithm the nodes into:'into' and x11:'card reader' get connected to into:'into card reader'. The node x11:'card reader' get deleted because this node is not connected to other nodes.

If all sentences are translated using RECOCASE-logic a *crosstable* can be created. The words or word phrases form the set of attributes where no word or phrase should exist twice. All translated sentences form the set of objects. Viewpoints can be compared by taking the set of objects for each use case description for each viewpoint and generating a combined crosstable and *concept lattice*.

4 Conclusion and Future Work

Many approaches to the formalisation of requirements begin with the assumption that requirements already exist in computer processable format such as a tabular or logical representation. We see this assumption as a major impediment to formal approaches becoming accessible to most organisations. While we offer our own viewpoints based approach, we believe the work reported which transforms natural language into a crosstable is beneficial for many other formal requirements engineering techniques.

Only a small amount of the work conducted in the RECOCASE project has been presented in this paper. Other parts of the project include:

- the RECOCASE process model
- the RECOCASE group decision support system approach
- development of the RECOCASE-tool to capture, edit and compare viewpoints
- the development of guidelines for specifying use case descriptions

- the specification of a controlled language to improve translation into FLFs and implementation of verification rules to assist the user in keepng to the controlled language

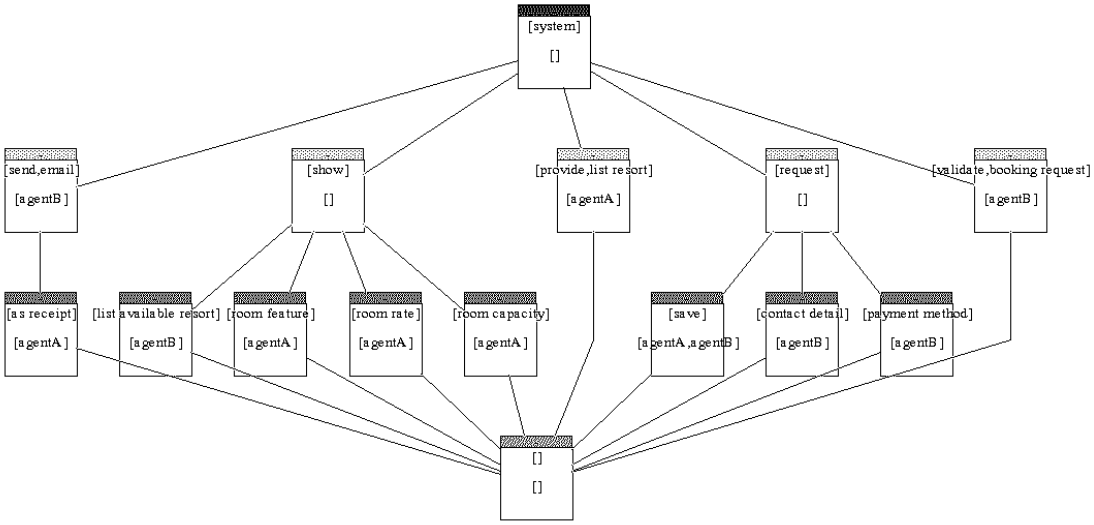


Figure 5: *Concept Lattice* for the Booking Room Use Case based on the sentences using the word 'system' from Agent A and Agent B viewpoints
To read the diagram start at a bottom node containing the owner of the sentence (agent A and/or agent B). Follow ALL ascending paths to pick up the phrases in the sentence. Nodes with more than one owner represent shared sentences. Nodes which share superconcepts (higher nodes) show shared phrases and may indicate partially shared sentences. A higher node with an owner shown in the node (see far left) shows a sentence subsumed by another sentence, in this case a sentence in another viewpoint.

We have conducted initial evaluations on the comprehensibility of the line diagram as a way of analysing and comparing use case descriptions. Our study involved 201 second year analysis and design students who answered questions using line diagrams OR the original textual descriptions. Five different line diagrams were considered, four of which included multiple viewpoints in the one diagram. One is shown in Figure 5. Our findings show that the line diagram could be understood accurately by 58% of our subjects after a 5 minute introduction, questions were up to 80% more likely to be correct when using the line diagram as opposed to textual sentences and that 61% of students preferred using the line diagram over sentences to answer the questions. Answering the questions using the diagrams was up to 9.9 times faster than using sentences. We are about to conduct evaluations of the group process. In particular we want to see how quickly a group facilitator can be trained to select features of interest to explore in the diagrams and to lead a project team in application of our resolution strategies. With the results of these evaluations we will revise each aspect of our project. Our ultimate goal is to offer an approach and a tool that provides a rigorous way of capturing a more complete and representative set of user requirements.

References

[1] Birkhoff, G., (1967) *Lattice Theory*, American Mathematical Society, Providence, Rhode Island

[2] Constantine, L. L. and Lockwood, L. A. D., (1999) *Software for Use : A Practical Guide to the Models and Methods of Usage-Centered Design*, ACM Press

[3] Darke, P. and Shanks, G., (1997) *Managing User Viewpoints in Requirements Definition*, 8th Australasien Conference on Information Systems

[4] Easterbrook, S. and Nuseibeh, B. (1996) *Using Viewpoints for Inconsistency Management* BCSEEE Software Engineering Journal January 1996, 31-43

[5] Fromkin, V., Rodman, R., Collins, P. and Blair, D., (1996) *An introduction to language*, 3th edition, Harcourt Brace & Company, Australia

[6] Jacobson, I., (1992) *Object-Oriented Software Engineering*, Addison-Wesley

[7] Melcuk, I., (1988) *Dependency Syntax: Theory and Practice*, State University of NY Press

[8] Molla, D., Schwitter, R., Hess, M. and Fournier, R., (2000) *Extrans, an answer extraction system* T.A.L special issue on Information Retrieval oriented Natural Language Processing

[9] Mullery, G. P. (1979) *CORE - a method for controlled requirements expression* In Proceedings of the 4th International Conference on Software Engineering (ICSE-4), IEEE Computer Society Press, 126-135.

[10] Richards, D. and Menzies, T., (1998) *Extending the SISYPHUS III Experiment from a Knowledge Engineering to a Requirement Engineering Task*, 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98), Banff, Canada, SRDG Pub.,Dept. of Comp. Sci., Uni. of Calgary, Calgary, Canada, Vol 1:SIS-6.

[11] Richards, D. and Zowghi, D., (1999) *Maintaining and Comparing Requirements*, Proc. of the Fourth Australian Conference on Requirement Engineering ACRE'99, Macquarie University, Sydney, 29-30 September,

[12] Sleator, D. D. and Temperley, D., (1991) *Parsing English with a Link Grammar*, Technical Report CMU-CS-91-196, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA

[13] van Vliet, H., (2000) *Software Engineering, Principles and Practice*, John Wiley & Sons Ltd, England,

[14] Wille, R., (1982) *Restructing Lattice Theory: An Approach Based on Hierarchies of Concepts*, Ordered Sets, D. Reichel, Dordrecht, pp. 445-470

[15] Wille, R., (1992) *Concept Lattices and Conceptual Knowledge*, Computers and Mathematics with Applications, 23, pp. 493-522