Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements

Didar Zowghi
University of Technology - Sydney
e-mail: `didar@it.uts.edu.au`

Vincenzo Gervasi
University of Pisa, Italy
e-mail: `gervasi@di.unipi.it`

Andrew McRae
CSIRO-Macquarie University
e-mail: `amcrae@mpce.mq.edu.au`

Abstract

The use of logic in identifying and analysing inconsistency in requirements from multiple stakeholders has been found to be effective in a number of studies. Default reasoning is a theoretically well founded formalism that is especially suited for supporting the evolution of requirements. However, direct use of logic in eliciting requirements and in discussing them with stakeholders poses serious useability problems. In this paper we explore the integration of natural language parsing techniques with default reasoning to overcome these difficulties. We also propose a method for automatically discovering scenarios that expose inconsistencies in requirements, and show how to deal with them in a formal manner. These techniques were implemented and tested in a prototype tool called CARL.

Keywords. *Requirements, default logic, natural language, inconsistency, scenarios.*

1 Introduction

Requirements often tend to be informal and ambiguous (at least in their initial stages), being the subject of much thought and experimentation, and they are usually communicated in natural language [17]. They are frequently inconsistent and contradictory since they are expressed by different stakeholders, who typically change their minds many times during software development. Requirements are also often incomplete and frequently in a state of flux as they are elicited in an incremental and evolutionary manner. The goal of the participants in requirements engineering (RE), however, is to develop a software requirements specifications (SRS) that is concise, consistent, clear to readers, complete with respect to user needs, and ideally can be delivered on time. Since inconsistency in software is problematic, management of consistency must be given a high priority in software development in general and in RE in particular. In recent years, tolerating inconsistency has been advocated by several researchers (e.g. [4, 8, 9]). Some approaches have also identified the need to enable reasoning and performing actions to continue in the presence of inconsistency [14]. Whatever approach one may take in dealing with inconsistency, it is clear that first and foremost inconsistencies have to be discovered and analyzed.

Maintaining consistency of the entire SRS and even of requirements models from which the SRS is derived is a difficult, expensive and time consuming activity for a requirements engineer, especially if requirements are expressed entirely in an informal representation scheme such as natural language. Clearly, a requirements engineer would greatly benefit from any methods, techniques or tools that could assist in this important task.

While it may not always be possible to get a requirements specifications absolutely correct and consistent, the ability to change requirements models whenever necessary with relative ease and rigour is crucial. The use of logic to formalize requirements models provides a proof theoretic framework which facilitates the modification of specifications to more adequately meet user requirements as well as to meet changing requirements.

In our earlier work [27, 28], we have developed a formal framework for reasoning about requirements change which builds on results from the areas of default reasoning [21, 22] and belief revision [10]. We have shown that viewing requirements models at the meta-level as default theories in a nonmonotonic knowledge representation language provides a useful basis for handling inconsistency. Given a commitment to such a formal language, we have shown that operators which map between theories of this language can be used to obtain a formal account of requirements change with features that are specially useful in the context of requirements engineering.

The Computer-Assisted Requirements Evolution Toolset (CARET) was proposed as a supporting environment for this framework [26]. CARET accepts requirements expressed in logic and modeled as default theories. It performs consistency checking on these models by utilizing the nonmonotonic operators for revision and contractions of requirements. CARET's reasoning and analysis capability is based on the theoretical foundations of classical logic, nonmonotonic reasoning and belief revision. To make this tool usable by a typical requirements engineer, it is necessary to build an additional interface layer on top of the basic formal machinery, that would not assume any strong familiarity with formal logic. In order to meet usability objectives and have at the same time a system whose functioning is based on semantically well-founded theory, in this paper we explore the use of Cico, a simple engine for natural language parsing [2, 3] as a front-end for entering requirements in natural language. In particular, we introduce here a novel scheme for translating NL requirements to default logic formulas, thus for the first time applying the CARET framework to NL requirements, and a novel technique for generating and checking scenarios corresponding to the requirements, that we use to expose potential conflicts between stakeholders.

We present a simple example that illustrates the different facets of our contribution. The example is taken from a case study of the Computer Aided Despatching system for the London Ambulance Service [7, 14]. We use this example to illustrate that :

- Natural language is a useful and reasonable medium for expressing requirements;
- Controlled natural language can be translated effectively into nonmonotonic logic;
- Nonmonotonic reasoning can be used to identify and analyze inconsistencies.

The paper is organized as follows. Section 2 introduces some motivation and theoretical basis for the use of default reasoning in support of the evolution of a requirements document. Section 3 describes the tools Cico and CARET, and discusses their integration, aimed at providing a theoretically well-founded treatment for natural language requirements. The practical applications of the integrated tools are shown through an example in Section 4. Some conclusions and ideas for future work complete the paper.

2 Using default reasoning in requirements evolution

The formal model for the management of inconsistency in changing requirements which was introduced in [26, 27, 28] draws on results from two distinct strands of research in AI, *default reasoning* and *belief revision*. Firstly, from research in the area of default reasoning, it obtains a knowledge representation scheme which permits the explicit representation of three distinct classes of entities in a requirements model. These are the *firm (essential) requirements* that must necessarily be satisfied, the *defeasible requirements* (requirements that are tentative and which may not hold for all subsequent models), and the *discarded requirements* (requirements that must not be included in the requirements model). Secondly, from research in the area of belief revision, it draws on a *competence theory*, that is, a set of semantically-motivated correctness criteria and on a *performance theory*, which offers an effective starting point for the construction of operators for requirements evolution.

We made a specific commitment to using the language of THEORIST for default reasoning [21, 22]. This choice is driven by the fact that the THEORIST system represents a simple, yet elegant, approach to default reasoning. More importantly, it meets the requirements and properties of a formal model for management of changing requirements identified in [26].

The THEORIST system decomposes a knowledge base into three distinct sets of assertions (*buckets*). These are, the assertions that must necessarily hold, called *facts* (denoted by the set F), the defeasible, or tentatively true assertions, called *hypotheses* (denoted by the set H), and the assertions that must necessarily not hold, called *constraints* (denoted by the set C). A default theory (or specification) in the THEORIST framework is denoted by the triple $(\mathsf{F}, \mathsf{H}, \mathsf{C})$. In our formal model, a requirements model is viewed as a THEORIST specification. Within each requirements model, requirements that are known to be true of the domain, together with the domain knowledge (i.e., essential requirements), are treated as a set of facts or the elements of the theory F. The elements of set H (hypotheses), are used to represent tentative or defeasible requirements, which may not always hold during the evolution process. Hypotheses are additionally useful for representing domain-specific default requirements (such as the requirement that "all vehicles on public roads must have a driver"). Elements of set C convey the explicit notion of disbelief (i.e., discarded requirements). These are conditions that must not hold in a requirements model. Constraints are specially useful for recording, in a semantically meaningful manner, the withdrawal (or *retraction*) of requirements.

Default reasoning in this framework involves identifying *extensions*, where each extension consists of the set of facts F together with a maximal (with respect to set inclusion) subset of the set of hypotheses ($h \subseteq \mathsf{H}$) that is consistent with the facts. An additional proviso is that each extension must be consistent with the set of constraints. Constraints can thus be used to specify what assertions may not appear in an extension (this is achieved by placing their negations in the set C). Following [21], we present the following definition of a maximal scenario (or extension):

Definition 1 (extension) *For a THEORIST specification* $(\mathsf{F},\mathsf{H},\mathsf{C})$*, where* F *is the set of facts,* H *is the set of hypotheses and* C *is the set of constraints, such that* $\mathsf{F}\cup\mathsf{C}$ *is satisfiable, a* **maximal scenario (extension)** *is a set* $\mathsf{F}\cup h$ *such that* $h \subseteq \mathsf{H}$ *and* $\mathsf{F}\cup h\cup\mathsf{C}$ *is satisfiable and* $\nexists h'$ *such that* $h \subset h'$ *and* $\mathsf{F}\cup h'\cup\mathsf{C}$ *is satisfiable.*

Using the relative degree of importance associated with each requirement (referred to *epistemic entrenchment ordering* in belief revision jargon), the issue of which scenario of the current default theory should be preferred over the others at each *step of evolution* can be addressed. Parallel exploration of alternatives has the advantage that the consequences of abandoning a belief can be explored before a decision is made. Such parallel exploration of several internally consistent, but mutually inconsistent, belief systems may of course, give an external observer the illusion of a single inconsistent system [16]. This is precisely what the advocates of "tolerating inconsistency" [4, 8, 9] have been interested to achieve. In a way, what is being achieved here is well beyond just tolerating inconsistency in that it also involves delaying the resolution of inconsistencies, which is what our formal framework supports intrinsically.

By holding on to the requirements that are responsible for causing an inconsistency (and were previously believed to be less important than those that remained in the requirements model), they have both been tolerated and have given the stakeholders another chance to revisit them at future stages of the evolution of the requirements. The framework presented in this paper provides the opportunity for the requirements analyst to present to the problem owners the consequences of changing the rank of a requirement with respect to the inconsistency it can create.

The revision operation on requirements models is defined as follows:

Definition 2 (revision)
$\mathsf{Rev}((\mathsf{F},\mathsf{H},\mathsf{C}),\alpha) = (\mathsf{F}^*_\alpha, \mathsf{H}\cup(\mathsf{F}-\mathsf{F}^*_\alpha), \mathsf{C}^-_{\neg\mathsf{F}^*_\alpha})$

Here, $\mathsf{Rev}((\mathsf{F},\mathsf{H},\mathsf{C}),\alpha)$ denotes the outcome of revising a requirements model, $(\mathsf{F},\mathsf{H},\mathsf{C})$ with an input α (i.e. a new requirement), which is itself another default theory. Here, F^*_α means that the set of essential requirements in F is revised with the new requirement α. Then $\mathsf{H}\cup(\mathsf{F}-\mathsf{F}^*_\alpha)$ means that the set of defeasible requirements H is augmented to include those requirements that belonged to the previous set of essential requirements F but do not belong to the revised set F^*_α. This action is essentially demoting the status of this special class of requirements from "essential" to "tentative". Finally, the revised set of essential requirements is contracted from the prior set of constraints C so that the consistency of the new set of constraints $\mathsf{C}^-_{\neg\mathsf{F}^*_\alpha}$ with new F is ensured.

Contraction involves the removal of an existing requirement from the requirements model. Once again, this removal may cause an inconsistency in the resulting requirements model. The inconsistency may be resolved in multiple possible ways. The operation of contracting a sentence α from a requirements model represented as a THEORIST default theory $(\mathsf{F},\mathsf{H},\mathsf{C})$ is defined as follows:

Definition 3 (contraction)
$\mathsf{Con}((\mathsf{F},\mathsf{H},\mathsf{C}),\alpha) = (\mathsf{F}^-_{\neg\mathsf{C}^*_{\neg\alpha}}, \mathsf{H}\cup(\mathsf{F}-\mathsf{F}^-_{\neg\mathsf{C}^*_{\neg\alpha}}), \mathsf{C}^*_{\neg\alpha})$

Here $\mathsf{Con}((\mathsf{F},\mathsf{H},\mathsf{C}),\alpha)$ denotes the outcome of contracting a requirement α from the current requirements model denoted by $(\mathsf{F},\mathsf{H},\mathsf{C})$, which is itself another default theory. As can be observed from the obvious similarities of the two operations, the contraction operator is symmetrically opposite to the revision operator. Firstly, through revising C by $\neg\alpha$, (i.e. $\mathsf{C}^*_{\neg\alpha}$), it is guaranteed that new C contains $\neg\alpha$. This means that no extension of the resulting theory shall contain α which is the essence of contraction operation. The second set $\mathsf{H}\cup(\mathsf{F}-\mathsf{F}^-_{\neg\mathsf{C}^*_{\neg\alpha}})$, is identical to that of revision operator where discredited requirements from F are demoted to the status of tentative requirements. The set of tentative requirements H is augmented to include those requirements that belonged to the previous set of essential requirements F but do not belong to the revised set F^*_α. Finally the negation of $\mathsf{C}^*_{\neg\alpha}$ is contracted from F to guarantee that new F is consistent with new C.

3 Tool support

Stakeholders can often have difficulties in expressing their intended requirements directly in some sort of formal language (including logic-based languages). As the elicitation process involves a substantial amount of *communication*, both between a stakeholder and the requirements engineer, and among the stakeholders, some kind of common language must be used to express the requirements in the initial phases of the RE process.

In this paper we advocate that natural language (NL) can serve this role, provided that some restrictions are placed on its use in order to encourage the expression of precise requirements. Constrained natural language of this sort has long been suggested as an effective means for recording and discussing requirements [5, 6, 12, 13, 15, 18, 20, 23]. Several studies

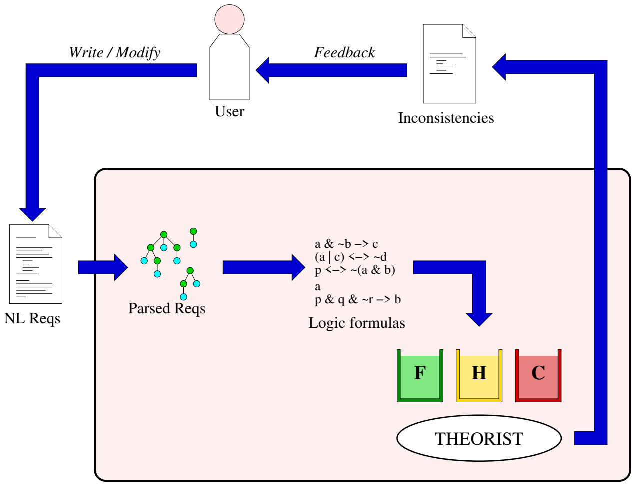


Figure 1. Processing of requirements in the CARL system. NL Requirements are first parsed by Cico and translated into logic formulas, then analyzed by CARET to discover and report any inconsistency.

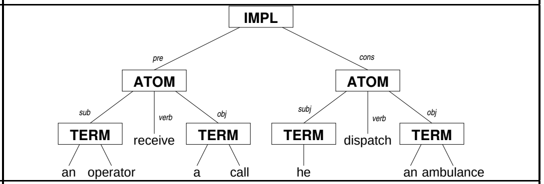
Original requirement	`When an operator receives a call, he should dispatch an ambulance.`
After morpho-syntactic analysis	`When/CC/WRB an/DT operator/NN receive/VB/Z a/DT call/NN he/PP should/MD dispatch/VB an/DT ambulance/NN`
Parse tree, as produced by Cico	
Equivalent logic formula	`receive(operator,call)` $\rightarrow$ `dispatch(operator,ambulance)`

Figure 2. An example of how NL requirements are translated into predicate logic.

exist on translation of natural language into logic of different kinds, including [1, 25]; we have opted to integrate the domain-based NL parser Cico [3, 11] with the computer-assisted requirement evolution tool CARET [26].

Domain-based parsing is a shallow parsing technique that exploits knowledge about the domain-specific properties of terms in order to determine an "optimal" parse tree for a natural language sentence. It is particularly suited to the analysis of documents that are of technical nature, since in such cases several simplifying assumptions hold. However, Cico can also be used as a generic syntax-based parser, by simply taking (a constrained subset of) the English grammar as its domain. The parser can also be described as an engine for the application of a fuzzy rewriting system to a text, using backtracking and heuristic optimization strategies to determine an optimal parse tree for a statement. "Optimal" in this context refers to optimality relative to a scoring system, which encourages the synthesis of parsing trees as complete as possible. Examples of factors that are taken into account by the scoring system are inversion in the order of words, dropped terms, depth of the parse tree, and specificity of the various parsing rules (see [11] for details).

In this work, we extend Cico with a preprocessing stage devoted to morphosyntactic analysis, and with a postprocessing stage devoted to the translation of parse trees into logic formulas. Morphosyntactic analysis on the text is performed with the help of TreeTagger [24], a tagger based on probabilistic decision trees obtained from annotated corpora. Given a sentence representing a requirement, this step annotates each word with a set of tags describing its part-of-speech role, and with the base form (lemma) of the word. The annotated text is then parsed by Cico, according to a small set of fuzzy parsing rules, to produce a parse tree corresponding to the original statement. The parse tree is finally translated into predicate logic form, and submitted to CARET for consistency checking and adding into its database. The whole process is exemplified in Figure 2, in which the tags used in morphosyntactic annotation are derived from those in the Penn Treebank set [19]. The intermediate nodes in the parse tree have the following meaning: a TERM node represents an entity in the domain of discourse, an ATOM node represents the application of a predicate (verb) to its arguments (subject, object, and complements), and an IMPL node represents an implication relation between an antecedent (premise) and a consequent (consequence).

While we certainly do not make any claim of completeness, the parser handles a number of common linguistic phenomena, i.e. reference, characterization through adjectives and relative clauses, negation, syndetic and asyndetic collation, etc.

Translation of Cico's parse trees into logic is performed according to a small number of patterns:

- simple actions expressed by *verbs* are translated into plain predicates. Every action has at least a *subject* performing the action, and may have an *object* and an arbitrary number of *complements*. Subjects, objects and the nominal part of complements are treated as arguments to the predicate. Thus, the statement

  The operator shall[1] dispatch an ambulance.

[1] Modal verbs are not directly translated into logic, but are

is translated into logic as the predicate

`dispatch(operator,ambulance)`

- asserting or checking properties of terms by using adjectives results in a predicate named after the property, and having the term as argument. Thus,

  Ambulance1 is available.

  is translated as

  `available(ambulance1)`

- common linguistic structures are mapped to the equivalent logic operators. This includes the basic *and*, *or*, and *not* operators, but also *implication* and *equivalence*. For example, the statement

  If the operator receives a phone call, he should dispatch an ambulance.

  is translated into logic as

  `receive(operator,phone_call)`
  `→ dispatch(operator,ambulance)`

  In the same vein, a statement like

  An ambulance is either working or broken.

  results in its logic equivalent

  `working(ambulance) ↔ ¬broken(ambulance)`

- subordinate clauses introduced by verbs in the present participle or by relative pronouns are expanded into their explicit form and conjuncted with the rest of the clause. Thus, for example,

  If the operator receives a phone call concerning a medical emergency...

  is treated as if it were

  If the operator receives a phone call **and** the phone call concerns a medical emergency...

- adjectives serve the role of *qualifiers*, constraining the applicability of a logic formula to only those instances that satisfies the qualification. All the qualifiers from a formula are collected, and used as premises to the formula itself. Thus, the statement

  If an operator receives a phone call concerning a medical emergency, he should dispatch the nearest ambulance.

  is translated as

```
medical(emergency) ∧ near(ambulance)
→ ( (receive(operator,phone_call) ∧
      concern(phone_call,emergency))
      → dispatch(operator,ambulance) )
```

Of course, these schemata are far from covering the whole range of linguistic phenomena that could be encountered in real, unrefined requirements; however, they prove sufficient for many applications.

The CARET component represents the implementation of the two operators for revision and contraction which were defined above. Every time a requirement is added to the model using the revision function, if an inconsistency occurs, the THEORIST algorithm in CARET systematically generates maximal consistent subsets (extensions), of the requirements model. The consistency of these subsets is determined by a call to a resolution-based linear theorem prover to check the satisfiability of the generated default theory. The next step involves making a choice from among the generated extensions. The tool offers both a manual and an automated facility for this selection. In the manual process these extensions are presented to the user to make a choice among them for the next step of the requirements evolution. In the automated version a choice is made based on the preference ordering of the individual requirements in each extension. This choice is based fundamentally on the "principle of minimal change" [10], which in this context means that when requirements models are being revised, the loss of information in the resulting requirements model should be kept to a minimum. This implies that if as a result of revision, an inconsistency occurs and one has a choice of removing a requirement from the model, more important requirements should be retained in favour of those that are less important or critical.

Representing the degree of epistemic entrenchment of individual requirements in a requirements model is a non-trivial task. There are many factors that may contribute to such ordering. Perhaps the most important ones are the cost and project schedule constraints, but the relative importance of the stakeholders can also be taken into account. We do not investigate the issue in this paper, assuming that the relative importance of the various requirements is assigned by the user (possibly assisted by some tool or formal technique).

Our reasoning tool ensures that requirements are never discarded. A requirement, once added via a revision operation, is contained in either F or H at all future times. Thus, if a new requirement r_2 contradicts an existing requirement r_1 contained in F, then r_1 is demoted to the status of a default (i.e., it becomes an element of H); thus, no maximal scenario of the THEORIST specification will contain r_1. If, however, r_2 is later contracted, r_1 can reappear in a maximal scenario of the resulting specification. One may view every element of F and C as representing a prior requirements evolution step. Every element of F represents a prior revision, while every element of C represents a prior contraction. Thus, priority relations on F and C, which are the only two prerequisites necessary for generating a consistent outcome of a requirements evolution step, can be obtained by merely requiring an ordering on the belief change steps. CARET uses the preference ordering entered by the user to select the maximal consistent subset that minimizes information loss.

used to assign a priority, or importance to the speaker, to the resulting formula. We will see in the following how priorities are used when inconsistencies arise among a set of statements.

4 An example

To show how the techniques we presented above would be used in a production environment, we resort to a classic example concerning a computer aided dispatch system employed by the London Ambulance Service (LAS). The LAS system has been used as a common case study in a number of works; an introduction and related references can be found in [7]. In particular, we follow here the steps of [14], in which a subset of the requirements for the LAS system is considered.

In our example, three stakeholders are identified: the Incident Room Controller (IRC), the Operations Manager (OM), and the Logistics Manager (LM). The IRC is in charge of processing phone calls to the emergency service, and to discriminate between calls that should prompt further action from the LAS, and calls that should be forwarded to other specialized emergency services (e.g., the fire brigade or the police station). The IRC stakeholder could then express the following requirements:[2]

Incident Room Controller	
IRC_1	A medical emergency is either an illness or an accident.
IRC_2	When an operator receives a phone call concerning a medical emergency, he should dispatch the nearest available ambulance.
IRC_3	When an operator receives a phone call concerning a non-medical emergency, • the operator should not dispatch an ambulance, and • he should transfer the phone call to another service.

The Operations Manager, on the other hand, is more concerned with the response times and with the general efficiency of the service. He could concentrate on which ambulances are dispatched, rather than on when they are dispatched. Let us assume thus that the following requirements are expressed by the OM:

Operations Manager	
OM_1	When an operator receives a phone call, he should dispatch the nearest available ambulance.
OM_2	When an operator receives a phone call, if an ambulance is not the nearest available, then the operator should not dispatch that ambulance.

[2]The examples are somewhat oversimplified in order to better illustrate the analysis techniques that are applied to the requirements of the various stakeholders. In particular, the concept of the *location* of the incident should be introduced, in order to determine where to dispatch an ambulance, and to define the concept of the "nearest" ambulance.

Finally, we have the Logistics Manager, who is mainly interested in managing the vehicles and their crews. The LM is not concerned with the actual usage of the ambulances to serve emergency call. However, the LM must ensure that only vehicles that are "in order" are used to carry the service on. We consider thus the following requirements for the LM:

Logistics Manager	
LM_1	If an ambulance does not have a crew, it is not available.
LM_2	If an ambulance was not serviced during the last year, then • the ambulance is not available, and • a technician must check it for maintainance.
LM_3	The operator does not dispatch an ambulance if it is not available.

Automatic translation of the requirements above, according to the parsing and translation techniques outlined in the previous section, yields the set of logic formulas in Table 1. None of these requirements cause any inconsistency *per se*, thus all of them are entered into the F bucket. Naturally, this is a consequence of the fact that all of the requirements are expressed in conditional terms, and we have no facts in our model.

We can use the analysis capabilities of CARET to instantiate the specification and simulate the behavior of the system under certain conditions. An analyst could then use the results of the simulation to verify his or her understanding of the problem, or to check with the stakeholders whether the behavior resulting from the combination of the requirements from all the stakeholder is acceptable to each of them.

A set of facts describing a certain state of the world and sequence of events is called a *scenario*.[3] For example, the following scenario expresses the "normal" case for processing a phone call received by the Incident Room Controller:

1. there is a medical emergency;
2. the operator receives a phone call concerning the emergency;
3. an ambulance is near and available.

When these facts — as natural language sentences — are entered into the system, their logic translation:

```
medical(emergency)
receive(operator,phone_call) ∧
concern(phone_call,emergency)
near(ambulance) ∧ available(ambulance)
```

[3]Several different definitions for the word "scenario" are known in the literature. Here we refer to a set of NL statements, or to their logic translation, providing information on particular cases that the software system described by the requirements may have to handle.

IRC_1	medical(emergency)	[a]
	→ (illness(emergency) ↔ ¬accident(emergency))	
IRC_2	(medical(emergency) ∧ near(ambulance) ∧ available(ambulance))	[o]
	→ ((receive(operator,phone_call) ∧ concern(phone_call,emergency))	
	→ dispatch(operator,ambulance))	
IRC_3	(¬medical(emergency))	[o]
	→ ((receive(operator,phone_call) ∧ concern(phone_call,emergency))	
	→ (¬dispatch(operator,ambulance) ∧ transfer(operator,phone_call,service)))	
OM_1	(near(ambulance) ∧ available(ambulance))	[o]
	→ (receive(operator,phone_call)	
	→ dispatch(operator,phone_call))	
OM_2	receive(operator,phone_call)	[o]
	→ (¬(near(ambulance) ∧ available(ambulance))	
	→ ¬dispatch(operator,ambulance))	
LM_1	¬have(ambulance,crew) → ¬available(ambulance)	[a]
LM_2	(¬(near(ambulance) ∧ available(ambulance)))	[a]
	→ ¬dispatch(operator,ambulance)	
LM_3	last(year)	[i]
	→ (¬revise(ambulance,year)	
	→ (¬available(ambulance) ∧ check(technician,ambulance,maintainance)))	

Table 1. Logic formulas for the requirements of the LAS. The codes on the right represent priorities: [a] = absolute (indicative mood), [o] = optative (should), [i] = imperative (must).

is inserted into the F bucket, again without causing any inconsistency. This assures that all the stakeholders agree on what the processing of the normal case should be. Moreover, we can inspect through CARL what other facts can be deduced for this scenario. Indeed, the system identifies these other facts as true (see Figure 3):

- `dispatch(operator,ambulance)` from IRC_2 and also from OM_1, i.e. the operator actually dispatches that nearest available ambulance;
- `have(ambulance,crew)` from LM_1, i.e. the ambulance will be dispatched with a crew on board.

Analogously, the Logistics Manager might want to know what happens when an ambulance has not been serviced lately. The corresponding scenario, in logic terms, is `last(y2k)` ∧ `¬revised(ambulance,y2k)`. This scenario does not produce any inconsistency, and yields as consequences the following facts:

- `¬available(ambulance)` from LM_3;
- `check(technician,ambulance,maintainance)` also from LM_3;
- `¬dispatch(operator,ambulance)` from LM_2.

The LM can then rest assured that all the stakeholders agree on the fact that in no case an ambulance that was not serviced during the last year will be dispatched by an operator.

This approach, analogous to the one used in [14], is effective for confirming the analyst's or the stakeholders' expectations for certain cases. However, it does

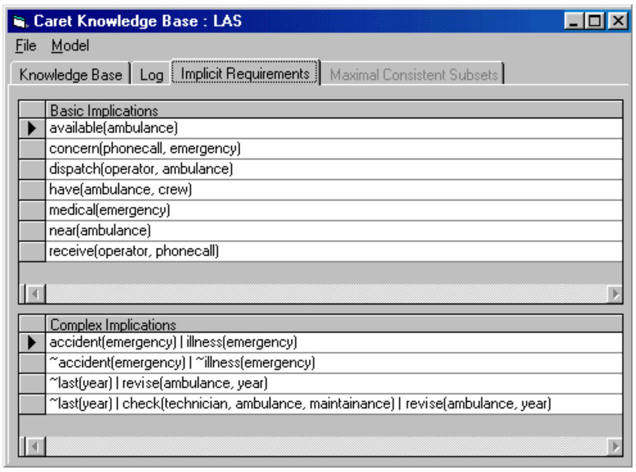


Figure 3. Checking the consequences of a scenario with CARL.

not help in checking the behavior of the system for *unexpected* scenarios. To this end, a more exhaustive analysis is needed.

CARL can perform an exhaustive search for plausible scenarios causing latent inconsistencies to emerge. To this end, the tool computes first the set U of *unbound* atoms, i.e., all atoms a that appear in the requirements and whose truth value is not known (either by explicitly having a or $\neg a$ in the requirements, or by deducing its value as a consequence of other facts). Then, all possible scenarios are generated, by considering all the possible subsets u of U, and assigning all the combinations of truth values to members of u.[4] Scenarios which lead to an inconsistency are then presented to the user for further analysis and investigation.

It is also possible to have a scenario partially entered as a set of facts in F, and let CARL generate all possible *completions* for that scenario, checking their consistency. In this way, we could for example enter the partial scenario "The operator receives a phone call" (automatically translated as `receive(operator, phone_call)`) in F, and have the tool check for all possible interactions with other facts that expose inconsistencies in the requirements.

Indeed, if we apply this technique to our example, CARL identifies the following scenario as triggering an inconsistency in the LAS requirements:

```
¬medical(emergency)
concern(phone_call,emergency)
near(ambulance)
available(ambulance)
```

In this scenario, IRC_3 and OM_1 are the causes of inconsistency. In fact, OM_1 tacitly assumes that a phone call will always result in the dispatching of the nearest available ambulance, regardless of the nature of the emergency, whereas the more complete IRC_3 specifies that in the scenario presented above no ambulance should be dispatched, and the call should be forwarded to some other service instead. Tacit assumptions of this kind are indeed often found in requirements documents. Three maximal consistent subsets are identified:

$\mathsf{F} - \{$`receive(operator,phone_call)`$\}$,
$\mathsf{F} - \{\mathrm{IRC}_3\}$, and
$\mathsf{F} - \{\mathrm{OM}_1\}$.

In other words, to remove the inconsistency we have either to assume that the operator will never receive a phone call, or we have to drop one of IRC_3 or OM_1. CARL also provides suggestions regarding the proper course of action, based on the priorities assigned to each requirement. In our example, "The operator receives a phone call" was stated in the indicative mood, and thus has a greater priority than the other two candidates (which were stated in the optative mood). CARL thus suggests retracting IRC_3 or OM_1 from our requirements.

In our example, our understanding of the problem would probably be sufficient to make a decision. But in a real case, the analyst might want to consult the stakeholders interested in the inconsistency (i.e., the Incident Room Controller and the Operations Manager) to decide how the inconsistency should be handled. Pending a final decision on the matter, one of the conflicting requirements will be demoted and moved from F to H. As a consequence, the demoted requirement will be treated as a default, to be considered valid unless some other requirements in F specifies otherwise. If we decide to demote OM_1, ambulances will be dispatched every time a phone call is received by the operator, unless some other requirement (i.e., IRC_3) explicitly prohibits it.

5 Conclusions and future work

This paper has addressed an important and challenging issue in RE, one that concerns identifying and analysing inconsistency in natural language requirements. Inconsistencies of this kind have been often found to be one of the greatest risk factors for the success of a software project. We have shown how requirements that are expressed in constrained natural language can be effectively parsed and translated into symbolic logic using Cico, a simple engine for NL parsing. We have also demonstrated how these translated logical statements of requirements can be analysed for inconsistency using the reasoning engine of CARET. Our approach to modelling and analysing requirements is novel because it combines the expressiveness of natural language with preciseness, rigour and formality of logic for handling inconsistency in requirements models. We have also developed a number of tools and techniques in support of this approach. We believe that this work has achieved significant steps towards providing automated support for identifying, analysing and handling inconsistency in NL requirements. In developing the work reported here, diverse threads from different research disciplines were brought together. The novel application of these theories and models to analysing NL requirements has opened up a new and fruitful area of research for which this work is a starting point.

Further work is needed to move to a more expressive logic that would allow us to specify more complex requirements. Hence the NL translation would have to be refined to accommodate the extended logic. If a reasoning system is based on first-order logic, there is no sound and complete decision procedure for logical consequences which is guaranteed to terminate. One possible solution to this problem is to restrict the language to a subset with known computational advantages such as Datalog. A less critical issue is to enhance CARL's functionality so that it can report inconsistencies and related scenarios to the user in NL. This information is already available in logical form within CARL, but is not reported to the user explicitly through NL sentences. We believe that by providing this feature, the user will be assisted in performing validation of the

[4] To obtain reasonable performances, particular care is taken to avoid generating scenarios whose outcome is already known, i.e. any scenario v with $v \subseteq u$ such that $u \cup \mathsf{F} \cup \mathsf{C}$ is already know to be consistent, or any scenario w with $u \subseteq w$ such that $u \cup \mathsf{F} \cup \mathsf{C}$ is already known to be inconsistent.

requirements, and the useability of the tool will be improved. Although we have not fully validated CARL on real projects this work has already captured the interest of industry. So much so that a project has already started with an industry partner in Australia to use and validate this tool in a production environment. Work is underway to assess the scalability issues associated with large industrial projects.

References

[1] S. Ali. A logical language for natural language processing. In *Proceedings of the 10th Biennial Canadian Artifical Intelligence Conference*, pages 187–196, Banff, AL, Canada, 1994.

[2] V. Ambriola and V. Gervasi. Processing natural language requirements. In *Proceedings of the Twelfth IEEE Conference on Automated Software Engineering*, pages 36–45. IEEE Press, Nov. 1997.

[3] V. Ambriola and V. Gervasi. Experiences with domain-based parsing of natural language requirements. In G. Fliedl and H. C. Mayr, editors, *Proceedings of the Fourth International Conference on Applications of Natural Language to Information Systems*, number 129 in OCG Schriftenreihe (Lecture Notes), pages 145–148, 1999.

[4] R. Balzer. Tolerating inconsistency. In *Proceedings of the 13th International Conference on Software Engineering (ICSE-13)*, pages 158–165, Austin, Texas, 1991. IEEE Computer Society Press.

[5] H. Dalianis. A method for validating a conceptual model by natural language discourse generation. In *Advanced Information Systems Engineering*, number 593 in LNCS. Springer-Verlag, 1992.

[6] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, 4(3), 1994.

[7] A. Finkelstein and J. Dowell. A comedy of errors: the london ambulance service case study. In *Proc. of the 8th International Workshop on Software Specification & Design*, pages 2–4. IEEE CS Press, 1996.

[8] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–577, 1994.

[9] D. Gabbay and A. Hunter. Making inconsistency respectable: A logical framework for inconsistency in reasoning, Part 1 - a position paper. In *Proceedings of Fundamentals AI Research*, pages 19–32. Springer-Verlag, LNCS Volume: 535, New York, 1991.

[10] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* MIT Press, Cambridge, MA, 1988.

[11] V. Gervasi. *Environment Support for Requirements Writing and Analysis.* PhD thesis, University of Pisa, February 2000.

[12] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. In *Proc. of the 4th International Conference on Requirements Engineering*, pages 140–148, June 2000.

[13] A. Hars. Advancing CASE productivity by using natural language processing and computerized ontologies: The ACAPULCO system. In *Proceedings of the Eleventh Automated Software Engineering Conference*, 1996.

[14] A. Hunter and B. Nuseibeh. Managing inconsistent specifications: Reasoning, analysis and action. *Transactions on Software Engineering and Methodology*, 7(4):335–367, Oct. 1998.

[15] N. Juristo, A. M. Moreno, and M. López. How to use linguistic instruments for object-oriented analysis. *IEEE Software*, 17(3):80–89, May/June 2000.

[16] R. Kowalski. *Logic for Problem Solving.* North Holland Elsevier, New York, 1979.

[17] B. Le Charlier and P. Flener. Specifications are necessarily informal or, some more myths of formal methods. *Journal of Systems and Software*, 3(40):275–296, Mar. 1998.

[18] B. Macias and S. G. Pulman. *Natural Language Processing for Requirements Specification*, pages 57–89. Chapman and Hall, 1993.

[19] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[20] L. Mich. NL-OOPS: From natural language requirements to object oriented requirements using the natural language processing system LOLITA. *Journal of Natural Language Engineering*, 2(2):161–187, 1996.

[21] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[22] D. Poole, R. Goebel, and R. Aleliunas. Theorist: a logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer Verlag, 1987.

[23] C. Rolland and C. Proix. A natural language approach for requirements engineering. In P. Loucopoulos, editor, *Advanced Information Systems Engineering*, number 593 in LNCS. Springer-Verlag, 1992.

[24] H. Schmid. Probabilistic part–of–speech tagging using decision trees. In *Proceedings of the Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK, Sept. 1994.

[25] B. L. Webber. *So What can we talk about now*, pages 331–370. MIT Press, Cambridge, MA, 1983.

[26] D. Zowghi. *A Logic-Based Framework for the Management of Changing Software Requirements.* PhD thesis, Sydney, June 1999.

[27] D. Zowghi, A. Ghose, and P. Peppas. A framework for reasoning about requirements evolution. In *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI'96)*, pages 157–168, 1996.

[28] D. Zowghi and R. Offen. A logical framework for modelling and reasoning about the evolution of requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE97)*, pages 247–259, 1997.