Reasoning About Inconsistencies in Natural Language Requirements

VINCENZO GERVASI
University of Pisa
and
DIDAR ZOWGHI
University of Technology, Sydney

The use of logic in identifying and analyzing inconsistency in requirements from multiple stakeholders has been found to be effective in a number of studies. Nonmonotonic logic is a theoretically well-founded formalism that is especially suited for supporting the evolution of requirements. However, direct use of logic for expressing requirements and discussing them with stakeholders poses serious usability problems, since in most cases stakeholders cannot be expected to be fluent with formal logic. In this article, we explore the integration of natural language parsing techniques with default reasoning to overcome these difficulties. We also propose a method for automatically discovering inconsistencies in the requirements from multiple stakeholders, using both theorem-proving and model-checking techniques, and show how to deal with them in a formal manner. These techniques were implemented and tested in a prototype tool called *CARL*. The effectiveness of the techniques and of the tool are illustrated by a classic example involving conflicting requirements from multiple stakeholders.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*Elicitation methods (e.g., rapid prototyping, interviews, JAD); methodologies (e.g., object-oriented, structured); tools*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Natural language*

General Terms: Theory, Verification, Human Factors

Additional Key Words and Phrases: Requirements, default logic, natural language, inconsistency

1. INTRODUCTION

Requirements engineering (RE) has been defined in Zave and Jackson [1997] as the branch of software engineering concerned with real-world goals for,

Authors' addresses: V. Gervasi, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, I-56127 Pisa, Italy; email: gervasi@di.unipi.it; D. Zowghi, Department of Software Engineering, Faculty of Information Technology, University of Technology, Sydney (UTS), P.O. Box 123, Broadway, NSW 2007, Australia; email: didar@uts.edu.au.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2005 ACM 1049-331X/05/0700-0277 $5.00

functions of, and constraints on a software system. The definition also refers to the relationships of these factors to precise specifications of system behavior, and to their evolution over time and across software families.

Many studies have recognized that requirements engineering is an area of paramount importance in software engineering research and practice. When asked about the causes of system failures, most practitioners identify poor requirements, incorrect specifications, and ineffective requirements management as the major sources of problems in the development process [CHAOS 1995; Ibanez 1996]. Early [Boehm 1976; Daly 1977] as well as recent [Davis et al. 1997] analyses have provided evidence that, the later in the software development life cycle an error is detected and corrected, the more costly it is to amend the final product. Clearly, early validation and correction of requirements can alleviate many of the problems associated with software development. Consequently, RE has the greatest leverage in the economics of software development.

One of the main classes of defects in requirements specifications is *inconsistency*. Inconsistency occurs when a specification contains conflicting, contradictory descriptions of the expected behavior of the system to be built or of its domain [Ghezzi and Nuseibeh 1998]. Such conflicting descriptions may come (i) as a result of conflicting goals between the various parties that contribute to the specification (usually called the *stakeholders*), or (ii) as a consequence of uncoordinated changes introduced in the specification during the usual evolution of the requirements.

Inconsistency is a major problem that permeates all aspects of software development. It makes it impossible to design and implement a system that respects its specification. In some cases, inconsistencies may be arbitrarily resolved by the programmer during the implementation of the system. This means that one of the conflicting requirements will be preferred over the others, usually without performing an in-depth analysis of the consequences, and without notifying the relevant stakeholders. Even worse, *undetected inconsistency* may lead to incorrect and unreliable systems, whose faults are only discovered when it is too late—during operation.

Two main schools have emerged with respect to the treatment of inconsistency in software specifications. The first proposes techniques and tools to ensure that inconsistency is not present in the specification at all times—inconsistency is treated as an *error*, to be corrected before further activities can take place [Sadri and Kowalski 1986; Tsai et al. 1992]. As this may be difficult to obtain in practice, the second school proposes instead that inconsistency may be *tolerated*, and resolved at a later stage [Balzer 1991; Gabbay and Hunter 1991]. In this second case, useful reasoning can be performed on the requirements even while inconsistencies are present in the specification [Hunter and Nuseibeh 1998; Nuseibeh 1996]. Moreover, the requirements can continue evolving: we say that inconsistency is *nonblocking* in this approach.

To effectively manage inconsistency, we need a certain degree of formality. It is relatively easy to identify *explicit* inconsistencies in the requirements (e.g., when two stakeholders have provided conflicting descriptions for the same phenomena). However, only a formal specification allows the identification of *implicit* (or *hidden*) inconsistencies—those cases in which the inconsistency

arises between the consequences of some requirements, rather than between the requirements themselves.

In this article, we concentrate on a particular kind of inconsistency, *logical contradiction*: any situation in which some fact α and its negation $\neg\alpha$ can be simultaneously derived from the same specification. Despite being so radically simple, contradiction is at the heart of many different phenomena, like infeasible requirements (i.e., requirements that contradict domain properties), disagreement among stakeholders (contradiction between requirements from different sources), or more recent requirements that conflict with previous ones (contradiction between new and old requirements). Indeed, given an arbitrary property π that we want to hold, it is sufficient to state it among the requirements to have any violation of π cause a contradiction. We will consider contradiction in the framework of a nonmonotonic variant of propositional logic. Propositional logic is the simplest form of logic, and its expressive power is limited. However, it is well suited to model a large set of problems, its decision procedure is sound and complete, and it is guaranteed to terminate. These advantages make it the most convenient to illustrate our approach, as will be shown in the rest of the article.

Writing and analyzing a specification in formal logic, or in most other formal specifications languages, for that matter, requires high expertise [van Lamsweerde 2000]. Often, such expertise is not readily available, and this contributes to the limited use of formal methods in industrial context. Moreover, even when an expert in formal methods is available, the stakeholders cannot be expected to be or become experts themselves. Some translation between formal and informal languages is thus needed. This translation itself introduces in the process a new source of potential errors and delays that may actually make matters worse than they were in the first place.

As a solution to this problem, we propose using natural language (NL) as a representation language for the requirements. This choice is motivated by the observation that NL is the language normally used by the stakeholders when proposing, discussing, and assessing new requirements: according to a recent survey [Mich et al. 2004], 95% of the requirements documents found in industrial practice are written in common (79%) or structured (16%) natural language. Comparable findings have been reported in an independent survey [Neill and Laplante 2003], showing that only 7% of the respondents used some kind of formal language to express requirements. NL is also the only language that can be assumed to be common to all the stakeholders. Its use encourages expression and experimentation, which are of paramount importance in the early stages of the evolution of a specification. However, NL lacks a formal semantics, and is thus not well suited, by itself, to the kind of formal analysis we need to perform to discover inconsistencies. Also, NL is inherently ambiguous. Using a controlled form of NL (e.g., forbidding the use of "some") can reduce the degree of ambiguity, but cannot remove it completely: a deeper analysis is needed for that purpose (see Berry et al. [2003] for a review on the topic).

Given that logic and NL have complementary advantages and disadvantages, we can envision an environment in which requirements are expressed in NL, and automatically translated into formal logic. Analysis and reasoning is

performed on the formal representation of the specification, and results are presented back to the stakeholders as NL sentences, thus effectively hiding the formal reasoning machinery from them. The existence of a completely automatic[1] proof procedure for propositional logic is of great value in this context.

This article presents three major contributions:

(1) We define a formal framework for identifying, analyzing, and managing inconsistency in requirements specifications. In our framework, we provide a formal definition for what others have called *tolerating inconsistency*, and provide the means to discover such inconsistencies—both explicit and hidden ones.
(2) We define a parsing technique and a translation schema that allow requirements expressed as simple (controlled) natural language sentences to be automatically transformed into propositional logic formulae, as well as their reverse translation (from logic formulae into NL sentences).
(3) We present a prototype tool, called *CARL*, that we have developed as a proof of concept. CARL implements all the techniques defined in the article, and provides a graphical user interface to simplify the access to the various operations that comprise our framework.

The article is organized as follows. Section 2 presents a general overview of the various techniques that we propose, and of how they fit together to address the problems that we outlined above. This is followed by two sections covering the theoretical foundations and formal details of those techniques. These sections also provide a description of how the techniques have been implemented in our tool CARL. In particular, Section 3 deals with our formal model of specifications, and with the operations that can be performed on such specifications, while Section 4 presents the techniques for translating between natural language and logic. A number of small examples illustrate the various definitions. Section 5 presents a more substantial example, and shows how CARL can be used to discover and manage inconsistencies that are introduced in the specification by multiple stakeholders having different goals. Section 6 discusses the limitations of the approach, and compares our work with other proposals in the literature. Some conclusions and directions for future works complete the article. Two appendices are included: Appendix A may serve as a quick reference for those readers not familiar with the terminology and basic concepts of belief revision; Appendix B reports on the results of some tests on the effectiveness of the translation between NL and logic performed by CARL.

2. OVERVIEW

In this section, we present a broad overview of our approach to handling inconsistency in NL requirements, while we present the details in Sections 3 and 4. The reader may refer to Figure 1 for a graphical depiction of the entire process.

[1]Theorem provers for more sophisticated forms of logic usually require human intervention to guide the derivation of a proof, thus defeating the whole purpose of using NL for requirements.

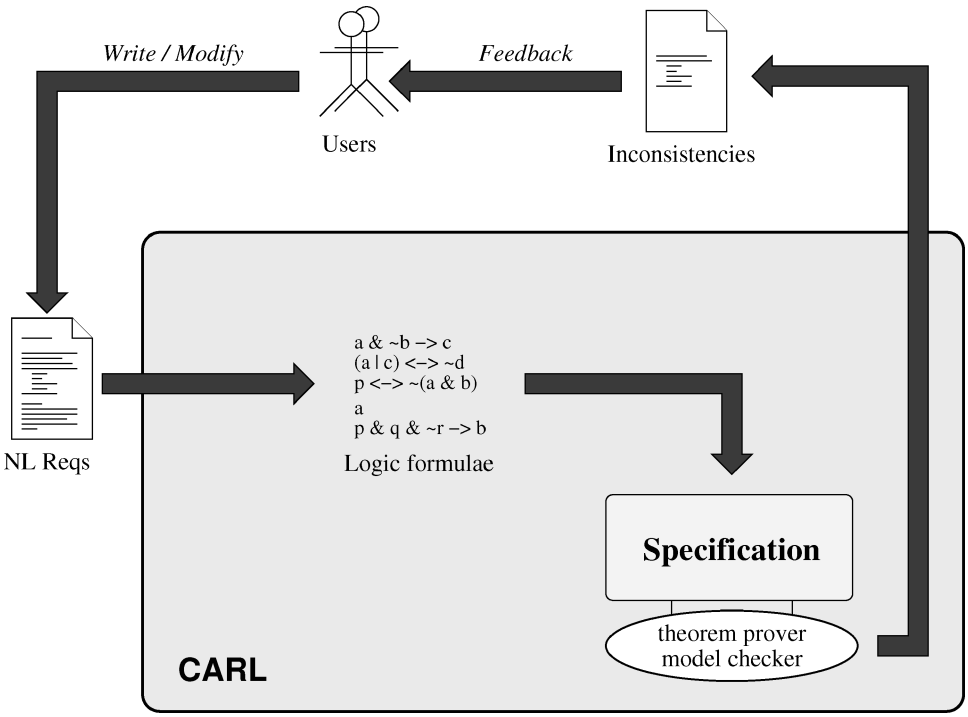
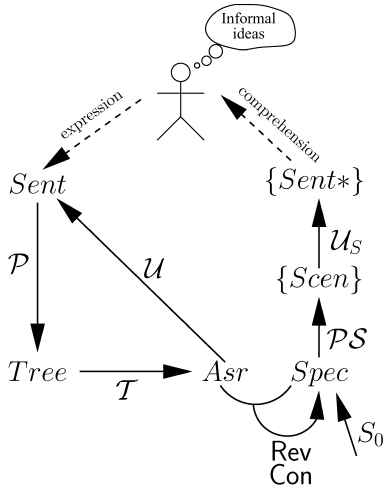


Fig. 1. A general overview of our approach to inconsistency handling in NL requirements, as embodied by CARL.

In the course of the requirements evolution process, requirements are introduced by a number of stakeholders. At this stage, requirements are expressed as natural language sentences, and each stakeholder can state the requirements that are significant from her particular viewpoint. These NL requirement are then submitted to a tool (CARL), and subjected to a series of transformations: (i) typographical adjustments, tokenization, and synonym substitution; (ii) parts-of-speech tagging; (iii) parsing, and transformation into a set of parse trees, and (iv) translation into a set of logic formulae.

These logic formulae are then added to the specification (which is represented as a theory in default logic), either as requirements (a property that is desired to hold), or as constraints (a property that is desired not to hold—notice that this is different from a property that is not desired to hold), depending on the type of operation initially requested by the stakeholder. The various requirements operations are modeled as belief revision operations on the theory that represents the specification. As a result of these changes, inconsistencies may arise in the specification; the existence of such inconsistencies is checked by using a theorem prover as part of the processing for the change. If any inconsistency is found, a detailed report and the various alternative interpretations are presented to the stakeholders, providing an opportunity to examine the situation. In any case, the specification is revised to accommodate the new requirements, possibly demoting other requirements in the process.



Domains

—$Asr \equiv \mathbb{N} \times \mathcal{L}$ (where $\mathcal{L}$ is the language of propositional logic, defined as usual)
—$Spec \equiv \mathbb{P}(Asr) \times \mathbb{P}(Asr) \times \mathbb{P}(Asr)$
—$Scen \equiv \mathbb{P}(Asr)$
—$Tree \equiv$ standard term language, where the terms are those listed under the Action heading in Table I
—$Sent \equiv$ NL statements in the fuzzy language defined in Table I

Functions

—$\mathcal{P} \equiv$ parsing function
—$\mathcal{T} \equiv$ logic translation function
—$\mathcal{U} \equiv$ unparsing function
—Rev $\equiv$ revision operator
—Con $\equiv$ contraction operator
—$S_0 \equiv (\emptyset, \emptyset, \emptyset)$, the null specification
—$\mathcal{PS} \equiv$ problem scenarios function
—$\mathcal{U}_s \equiv$ set-wise unparsing function

—$A*$ denotes a finite sequence of A
—$\{A\}$ denotes a set of A
—$\mathbb{P}(A)$ denotes the power set of A

Fig. 2. An algebraic map of the most important domains and functions that will be used in the article.

A different, more elusive type of inconsistency can be revealed by using a model-checking approach.[2] This allows CARL to provide actual examples of situations that would expose hidden inconsistencies (these are in fact counterexamples that confute the theorem "the specification is consistent"), thus helping the stakeholders in focusing their analysis to specific cases.

In keeping with our general goal of simplifying the interaction with the stakeholders, the use of logic is completely hidden from them. Both the input and the output from CARL are entirely in natural language (although the logic forms of requirements and specifications can also be inspected, if so desired). Moreover, a graphical user interface is provided to further reduce the complexity of the interaction: stakeholders should concentrate on the system they are trying to specify, rather than on the tools they are using to do so.

Formally, we will use a number of domains and functions to define all the transformations and operations in our approach. These domains and functions are shown in Figure 2, and will be introduced in due course in the rest of the

[2]In this article, we use the term *model checking* in its pristine, more general sense, that is, "To algorithmically check whether a program (the model) satisfies a specification" [Clarke et al. 1986]. Other authors have used the term *model finding* in this sense, and *model checking* exclusively in reference to checking temporal logic properties of state space models, which is different from what CARL does.

article. The reader may use Figure 2 as a map of the general context for each of the functions defined in Sections 3 and 4.

The figure also includes two informally defined functions, *expression* and *comprehension*. These model two basic assumptions: (i) that the stakeholders are able to express their informal ideas (about the system that is being specified and about its domain) as NL sentences, and (ii) that the stakeholders are able to comprehend descriptions of inconsistencies that are presented as sets of NL sentences. This article does not address the issue of how to ensure that the requirements expressed reflect the intuitive understanding of the stakeholders. It should be noted, however, that the assumptions we make on the abilities of the stakeholders are much weaker than those needed by most heavy-weight formal methods. We regard that as an advantage of our approach.

3. USING DEFAULT REASONING IN REQUIREMENTS EVOLUTION

3.1 Formal Model

Our formal model for the management of inconsistency in changing requirements (which is based on the one originally introduced in Zowghi [1999], Zowghi et al. [1996], and Zowghi and Offen [1997] and implemented in a prototype called CARET [Zowghi et al. 1997]) draws on results from two distinct strands of research in AI, *default reasoning* and *belief revision*. First, 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 specification. 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 specification). Second, 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 adopt a version of nonmonotonic reasoning inspired by the THEORIST framework for default reasoning [Poole 1988; Poole et al. 1987]. This choice is motivated by the fact that the THEORIST framework 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 Zowghi [1999]. THEORIST provides a mechanism for explaining *observations* in terms of *facts* and *hypothesis*. We apply this mechanism to requirements, and extend it by providing operators for evolving these sets of facts and hypothesis, and by further adding the concept of a *scenario*, that is, a provisional set of observations describing a possible state of the world.

We decompose a requirements specification (or simply a *specification*) 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).

Definition 3.1 (*Specification*). A *default theory* (or *specification*) is a triple (F, H, C) where F, H, and C are sets of propositional formulae, and F ∪ C is satisfiable.

In our formal model, a requirements specification is viewed as a default theory. F contains assertions about the domain, which constitute a domain model, and essential requirements that the stakeholders are not prepared to give up. Thus, F describes the world as it would be after a system satisfying all essential requirements has been deployed in the domain. 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.

Example 3.2. Let us consider a specification detailing the domain of public roads. Parts of the specification could describe how bus lanes are to be used. In this case, the various buckets could include the following information:

F (facts)	H (hypotheses)	C (constraints)
emergency(ambulance) → drive(ambulance,bus_lane) emergency(car) → drive(car,bus_lane) drive(bus,bus_lane)	¬drive(car,bus_lane) ¬drive(ambulance,bus_lane)	¬park(car,bus_lane)

This information can be interpreted as follows: normally, ambulances and private cars cannot drive on bus lanes (from H). However, F states that, as a matter of fact, ambulances and private cars can indeed drive on bus lanes, if they are attending an emergency. Buses always drive on bus lanes. Finally, the contents of C constrain the future evolution of the specification so that private vehicles are never allowed to park on bus lanes (ambulances will be able to park on those lanes, if so permitted by F or H).

It should be noted that, while we hold both domain description statements and requirements in the same buckets for the purpose of our framework, their roles and the ways they evolve during the requirements engineering process are markedly distinct. See Zave and Jackson [1997] for the distinction between indicative domain descriptions and optative requirements, and Zowghi and Gervasi [2004] for a full treatment of their evolution, with a particular emphasis on consistency and completeness properties.

Default reasoning in our framework involves identifying *extensions*, where each extension consists of the set of facts F together with a subset of the set of hypotheses ($h \subseteq$ 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). Extending Poole [1988], we introduce the following definitions:

Definition 3.3 (*Extension*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$, and a set of assertions $h \subseteq \mathsf{H}$, an *extension* of $\mathcal{S}$ on h is a set of assertions e such that $e = \mathsf{F} \cup h \cup \mathsf{C}$ and e is consistent.

It follows from the definition that an extension is a consistent subset of $\mathsf{F} \cup \mathsf{H} \cup \mathsf{C}$. Among the many possible extensions of a specification, we are interested only in those that minimize the loss of information from H. This choice is in accordance with the *principle of minimal change* that is the founding principle in the AGM paradigm [Alchourrón et al. 1985]. We provide thus a characterization of those extensions that are maximal as follows:

Definition 3.4 (*Maximal Extension*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$, a *maximal extension* of $\mathcal{S}$ is an extension e for which $\nexists h'$ such that $h \subset h'$ and $\mathsf{F} \cup h' \cup \mathsf{C}$ is an extension.

Naturally, a specification may have multiple maximal extensions—each internally consistent, but mutually inconsistent with all the others. This phenomenon is shown in the following example.

Example 3.5. Consider the extended version of the specification (from Example 3.2) given below, in which we have added the hypothesis that an ambulance is currently servicing an emergency call:

F (facts)	H (hypotheses)	C (constraints)

`emergency(ambulance) →`
`  drive(ambulance,bus_lane)`
`emergency(car) →`
`  drive(car,bus_lane)`
`drive(bus,bus_lane)`
	`¬drive(car,bus_lane)`
	`¬drive(ambulance,bus_lane)`
	`emergency(ambulance)`
		`¬park(car,bus_lane)`

This specification allows three maximal extensions, namely:

e_1 = {`emergency(ambulance)` → `drive(ambulance,bus_lane)`,
`emergency(car)` → `drive(car,bus_lane)`,
`drive(bus,bus_lane)`, ¬`drive(car,bus_lane)`,
¬`drive(ambulance,bus_lane)`, ¬`park(car,bus_lane)`},

e_2 = {`emergency(ambulance)` → `drive(ambulance,bus_lane)`,
`emergency(car)` → `drive(car,bus_lane)`,
`drive(bus,bus_lane)`, ¬`drive(car,bus_lane)`,
`emergency(ambulance)`, ¬`park(car,bus_lane)`},

e_3 = {`emergency(car)` → `drive(car,bus_lane)`,
`drive(bus,bus_lane)`, ¬`drive(car,bus_lane)`,
`emergency(ambulance)`, ¬`drive(ambulance,bus_lane)`,
¬`park(car,bus_lane)`}.

The differences between the three can be better appreciated by writing them in terms of F, H, and C:

$$
\begin{aligned}
e_1 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\texttt{emergency(ambulance)}\},\\
e_2 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\neg\texttt{drive(ambulance,bus\_lane)}\},\\
e_3 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\texttt{emergency(ambulance)} \rightarrow \texttt{drive(ambulance,bus\_lane)}\},
\end{aligned}
$$

meaning that there are three possible consistent views of the world: either we negate that an ambulance can be attending an emergency (e_1), or we give up the assumption that ambulances do not drive on bus lanes (e_2), or we give up our notion that an ambulance serving an emergency can drive on bus lanes. It is easy to see that, of the three facts that we have dropped, respectively, in e_1, e_2, and e_3, any two can be accepted at the same time, whereas having all three of them would cause an inconsistency.

Obviously, e_2 is more in line with our understanding of how the real world works, although a budget-conscious government might also find e_1 interesting, and e_3 could be a viable and convenient alternative to ensure smooth bus traffic on public roads.

Our specification also presents other extensions that are not maximal, for example:

$$
\begin{aligned}
e'_1 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\texttt{emergency(ambulance)}, \neg\texttt{drive(car,bus\_lane)}\},\\
e'_2 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\neg\texttt{drive(ambulance,bus\_lane)}, \neg\texttt{drive(car,bus\_lane)}\},\\
e'_3 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\texttt{emergency(ambulance)}, \neg\texttt{drive(ambulance,bus\_lane)},\\
&\qquad\qquad \neg\texttt{drive(car,bus\_lane)}\},\\
e'_4 &= \mathsf{F} \cup \mathsf{H} \cup \mathsf{C} \setminus \{\texttt{emergency(ambulance)}, \neg\texttt{drive(ambulance,bus\_lane)}\},
\end{aligned}
$$

but only the maximal extensions e_1–e_3 are relevant for the choice of how the inconsistency can be resolved.

Using the relative degree of importance associated with each requirement (referred to as *epistemic entrenchment ordering* in belief revision terminology [Gärdenfors 1988], and as *requirements priority* in RE jargon), the issue of which extension of the current default theory should be preferred over the others at each *step of evolution* can be addressed. Here, by step of evolution we mean any modification to the requirements; a formal model for these modifications will be provided in Definitions 3.6 and 3.9 below.

Representing the degree of epistemic entrenchment of individual requirements in a specification is a nontrivial 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. Also, normally domain descriptions are more entrenched than requirements, in that requirements express desires that can be more easily renounced or delayed for future developments. We do not investigate the issue in this article, assuming that the relative importance of the

various requirements is assigned by the user (possibly assisted by some tool or formal technique[3]).

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 [Kowalski 1979]. This is precisely what the advocates of "tolerating inconsistency" [Balzer 1991; Finkelstein et al. 1994; Gabbay and Hunter 1991] 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 specification), 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 article provides the opportunity for the requirements analyst to present to the problem owners the consequences of changing the relative rank of a requirement (that is, its epistemic entrenchment) with respect to the inconsistency it can create.

We model two different types of evolution steps. The first, *revision*, is used to introduce new information into a specification, while the second, *contraction*, is used to state constraints and to withdraw previous assertions. As a result of these operations, inconsistencies may be introduced into the specification. In such cases, the user may be given the option to select interactively which of the possibly many maximal extensions should be used. The formal definition of these two operations is given below.

Definition 3.6 (*Revision*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$ and an assertion α, we define the operator $\mathsf{Rev} : \mathit{Spec} \times \mathit{Asr} \rightarrow \mathit{Spec}$ as

$$\mathsf{Rev}((\mathsf{F}, \mathsf{H}, \mathsf{C}), \alpha) = \left(\mathsf{F}^*_\alpha, \mathsf{H} \cup \left(\mathsf{F} \setminus \mathsf{F}^*_\alpha\right), \mathsf{C}^-_{\neg \mathsf{F}^*_\alpha}\right),$$

where

—*Spec* is the set of all possible specifications, and *Asr* is the set of all possible assertions (i.e., formulae in the usual propositional logic language),

—A^*_α denotes the set A, revised according to the AGM postulates [Gärdenfors 1988] so that the new assertion α is included, and all contradictory assertions already in A are removed from A, and

—A^-_B denotes the set A, contracted according to the AGM postulates, so that all the assertions in B are removed from A, together with any other assertion in A implying any of the removed ones.

[3]In Section 4, we will assume a coarse set of priorities based on the mood of the main verb of each requirement.

Remark 3.7. The A^* and A^- operations maintain consistency of A, meaning that, when applied to consistent sets, the results will still be consistent. It is thus important to start with a consistent set. In our model, we assume that all the requirements are introduced in the specification by means of the Rev operator, starting from an empty specification $\mathcal{S}_0 = (\{\}, \{\}, \{\})$ that is trivially consistent.

Rev((F, H, C), α) denotes the outcome of revising a specification, (F, H, C) with an input α (i.e., a new requirement). The result 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 $\mathsf{H} \cup (\mathsf{F} \setminus \mathsf{F}^*_\alpha)$ 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.

Example 3.8. Let us consider once again the specification $\mathcal{S}$ from Example 3.2. We want to enrich our description by stating that, if a bus lane is occupied by a bus, no other vehicle can use it (in a more refined specification, we would be considering overtaking as well). To this end, we revised the specification with the new requirement

$$\alpha = \texttt{drive(bus,bus\_lane)} \rightarrow \neg\texttt{drive(ambulance,bus\_lane)} \wedge \neg\texttt{drive(car,bus\_lane)}.$$

The addition of α to F does not cause any inconsistency, so the result of this revision is simply $\mathcal{S}' = (\mathsf{F} \cup \alpha, \mathsf{H}, \mathsf{C})$. In other words, α is simply added to F while the other buckets are unchanged. However, if we want to revise $\mathcal{S}'$ stating that there is in fact an emergency, as expressed by

$$\beta = \texttt{emergency(ambulance)},$$

we have that $\mathcal{S}'' = \mathsf{Rev}(\mathcal{S}', \beta)$ is given by

F (facts)	H (hypotheses)	C (constraints)
emergency(ambulance)→ drive(ambulance,bus_lane) emergency(car)→ drive(car,bus_lane) drive(bus,bus_lane)→ ¬drive(ambulance,bus_lane) ∧¬drive(car,bus_lane) emergency(ambulance)	¬drive(car,bus_lane) ¬drive(ambulance,bus_lane) drive(bus,bus_lane)	¬park(car,bus_lane)

where as a consequence of the introduction of `emergency(ambulance)`, the conflicting assertion `drive(bus,bus_lane)` has been demoted to a hypothesis. These two assertions are in fact inconsistent, as `emergency(ambulance)` implies `drive(ambulance, bus_lane)` and `drive(bus,bus_lane)` implies ¬`drive(ambulance, bus_lane)`.

The second type of operation we model, contraction, involves stating a constraint, thus reducing the space of possible behaviors of the system. Contraction is also used to retract an existing requirement from the specification, while recording the fact that it is no longer desired to hold (this is unlike the AGM contraction operator, where the contracted information is definitively lost: see Appendix A.1). Once again, contraction 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 specification $(\mathsf{F}, \mathsf{H}, \mathsf{C})$ is defined as follows:

Definition 3.9 (*Contraction*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$ and an assertion α, we define the operator $\mathsf{Con} : \mathit{Spec} \times \mathit{Asr} \rightarrow \mathit{Spec}$ as

$$\mathsf{Con}((\mathsf{F}, \mathsf{H}, \mathsf{C}), \alpha) = \left(\mathsf{F}^{-}_{\neg \mathsf{C}^{*}_{\neg\alpha}}, \mathsf{H} \cup \left(\mathsf{F} \setminus \mathsf{F}^{-}_{\neg \mathsf{C}^{*}_{\neg\alpha}}\right), \mathsf{C}^{*}_{\neg\alpha}\right),$$

where we use the * and $^{-}$ operators as in Definition 3.6.

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. First, 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 $\mathsf{H} \cup (\mathsf{F} \setminus \mathsf{F}^{-}_{\neg \mathsf{C}^{*}_{\neg\alpha}})$, 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.

Example 3.10. Let us consider again our budget-conscious government from Example 3.5, and let us suppose that it has been decreed that ambulances should not be allowed to drive in bus lanes at all. In our model, this can be represented as a contraction of $\gamma =$ `drive(ambulance,bus_lane)` from the specification $\mathcal{S}''$.

According to Definition 3.9, we must first revise C by $\neg\gamma$, that is, $\neg$`drive(ambulance,bus_lane)`. The revised C is

$$\mathsf{C} = \{\neg\texttt{park(car,bus\_lane)}, \neg\texttt{drive(ambulance,bus\_lane)}\}.$$

Then, $\neg\mathsf{C}$ is contracted from F. At this stage, we have an inconsistency, since F implies `drive(ambulance,bus_lane)`:

(a) `emergency(ambulance)`$\wedge$
(b) `emergency(ambulance)` $\rightarrow$ `drive(ambulance,bus_lane)` $\models$
(c) `drive(ambulance,bus_lane)`.

Thus, we have a choice: either we discard (a), or we discard (b). We can imagine that even a budget-conscious government would not deny that ambulances can have emergencies (after all, that is what they are for!), so (b) is probably

less entrenched than (a). Thus, (b) will be demoted and moved to H, giving rise to the new specification in the table below.

F (facts)	H (hypotheses)	C (constraints)
emergency(car) → drive(car,bus_lane) drive(bus,bus_lane) → ¬drive(ambulance,bus_lane) ∧¬drive(car,bus_lane) emergency(ambulance)	¬drive(car,bus_lane) ¬drive(ambulance,bus_lane) drive(bus,bus_lane) emergency(ambulance) → drive(ambulance,bus_lane)	¬park(car,bus_lane) ¬drive(ambulance,bus_lane)

Definitions 3.6 and 3.9 ensure 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 extension of the specification will contain r_1. If, however, r_2 is later contracted, r_1 can reappear in a maximal extension 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.

As appeared in Example 3.8, we need to make a distinction between two types of facts when analyzing specifications: facts that are provisionally true (e.g., `emergency(ambulance)`) and facts that are always true (e.g., `drive(bus,bus_lane)`→ ¬`drive(ambulance,bus_lane)`). The latter represent inherent properties of the domain, while the former cover two types of situations that we may need to consider for analysis purposes. The first one describes an instance of a circumstance that may arise (e.g., `emergency(ambulance)`, indicating the temporal extent during which an ambulance is serving an emergency). The second one is a speculation of a hypothetical condition for exploring "what if" analysis (e.g., `emergency(ambulance)` → ¬`drive(bus,bus_lane)`). These types of facts are used to investigate the possible consequences of new situations (e.g., new regulation asking buses to leave their lane if an ambulance is serving an emergency). To cater for these types of analysis, we introduce the concept of a *scenario*.

Definition 3.11 (*Scenario*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$, a scenario s is a set of asserted or negated atoms such that the following properties hold:

(1) $\forall a \in s, a \in H(\mathsf{F} \cup \mathsf{H} \cup \mathsf{C})$,
(2) $\forall \neg a \in s, a \in H(\mathsf{F} \cup \mathsf{H} \cup \mathsf{C})$,
(3) $\forall a \in s, (\mathsf{F} \cup \mathsf{C}) \not\models a \wedge (\mathsf{F} \cup \mathsf{C}) \not\models \neg a$, and
(4) $\forall \neg a \in s, (\mathsf{F} \cup \mathsf{C}) \not\models a \wedge (\mathsf{F} \cup \mathsf{C}) \not\models \neg a$,

where $H(A)$ denotes the Herbrand base for A (i.e., the set of all ground atoms occurring in A).

When writing specifications, most predicates are likely to be written as conditional clauses, describing causal dependencies like `emergency(ambulance)`$\rightarrow$ `drive(ambulance,bus_lane)`. On the other hand, when analyzing specifications, we will be more often interested in the actual behavior that is specified for a certain set of circumstances. To this purpose, we introduce the following notion of consistency under a certain scenario:

Definition 3.12 (*S-Consistency*). A specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$ is said to be *S-consistent* under a given scenario s (denoted by $\Sigma(\mathcal{S}, s)$) if $s \cup \mathsf{F} \cup \mathsf{C}$ is consistent.

Example 3.13. A specification can appear consistent under a certain scenario, and expose inconsistencies under a different one. Consider for example the specification resulting from the first revision in Example 3.8, shown again for clarity in the following table:

F (facts)	H (hypotheses)	C (constraints)

`emergency(ambulance)` $\rightarrow$
`drive(ambulance,bus_lane)`
`emergency(car)` $\rightarrow$
`drive(car,bus_lane)`
`drive(bus,bus_lane)` $\rightarrow$
$\neg$`drive(ambulance,bus_lane)`
$\wedge\neg$`drive(car,bus_lane)`
	$\neg$`drive(car,bus_lane)`
	$\neg$`drive(ambulance,bus_lane)`
	`drive(bus,bus_lane)`
		$\neg$`park(car,bus_lane)`

This specification is S-consistent with respect to the scenario

$$s_1 = \{\texttt{emergency(ambulance)}\}$$

but is not S-consistent under the scenario

$$s_2 = \{\texttt{emergency(car)}, \texttt{drive(bus,bus\_lane)}\}.$$

In fact, under s_2 both `drive(car,bus_lane)` and $\neg$`drive(car,bus_lane)` are entailed by the specification, thus exposing an inconsistency that was latent in the specification.

The Rev and Con operators always keep specifications internally consistent. To expose latent inconsistencies, those that only manifest themselves under a certain scenario, we define the set of "problematic scenarios" $\mathcal{PS}$, as follows:

Definition 3.14 (*Problematic Scenarios*). Given a specification $\mathcal{S}$, the set of the *problematic scenarios* $\mathcal{PS}(\mathcal{S})$ is

$$\mathcal{PS}(\mathcal{S}) = \{s \mid s \text{ is a scenario for } \mathcal{S} \wedge \neg\Sigma(\mathcal{S}, s)\}.$$

When computing problematic scenarios, we are interested mostly in the smallest scenario that exposes a certain inconsistency, to foster both computational efficiency and economy of presentation to the user. Minimal problematic scenarios are defined as follows:

Definition 3.15 (*Minimal Problematic Scenario*). A problematic scenario s for a specification $\mathcal{S}$ is *minimal* if

$$\forall s' \subset s,\ \Sigma(\mathcal{S}, s').$$

Minimal problematic scenarios are the best characterization for those unforeseen cases where a set of circumstances not explicitly ruled out by constraints in C materializes, but the stakeholders do not agree on what the behavior of the system should be in that case. By presenting minimal problematic scenarios back to the user, together with a trace of the inconsistency, the latent problem is exposed, and further investigation and discussion among the stakeholders can take place.

For practical purposes, it is convenient to partition scenarios into two sets, one given by the user and another, derived from the first, containing additional atoms. Formally, these two sets are defined as follows:

Definition 3.16 (*Partial Scenario, Completion*). Given a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$, a *partial scenario* s_i is a (possibly empty) scenario for $\mathcal{S}$ such that

$$\exists a \in H(\mathsf{F} \cup \mathsf{H} \cup \mathsf{C}) : (\mathsf{F} \cup s_i \cup \mathsf{C}) \not\models a \ \wedge \ (\mathsf{F} \cup s_i \cup \mathsf{C}) \not\models \neg a.$$

Given a partial scenario s_i such that $\Sigma(\mathcal{S}, s_i)$ holds, a *completion* s_i^j of s_i is any nonempty scenario of $(\mathsf{F} \cup s_i, \mathsf{H}, \mathsf{C})$.

The intuition behind the definition above is that a partial scenario is a user-provided set of assignment of truth values to some of the atoms in the specification, whereas an automatically generated completion assigns truth values to some of the atoms whose value was not provided in the partial scenario. Notice that both partial scenarios and completions are themselves valid scenarios, as is the set-theoretic union of a partial scenario with any one of its completions. Note also that we do not ask that completions provide truth value assignments for *all* the unbound atoms in the specification. Thus, minimal problematic scenarios can be represented as pairs (s_i, s_i^j). We will see in Section 5 how this representation turns out to be useful in practice to simplify user interaction on certain kinds of consistency analysis.

Example 3.17. Let us consider again the specification $\mathcal{S}$ from Example 3.13:

F (facts)	H (hypotheses)	C (constraints)

`emergency(ambulance)` → `drive(ambulance,bus_lane)`
`emergency(car)` → `drive(car,bus_lane)`
`drive(bus,bus_lane)` → ¬`drive(ambulance,bus_lane)` ∧¬`drive(car,bus_lane)`
	¬`drive(car,bus_lane)`
	¬`drive(ambulance,bus_lane)`
	`drive(bus,bus_lane)`
		¬`park(car,bus_lane)`

The following are some of the valid partial scenarios for $\mathcal{S}$:

$$s_0 = \{\},$$
$$s_1 = \{\texttt{emergency(ambulance)}\},$$
$$s_2 = \{\neg\texttt{emergency(ambulance)}, \texttt{emergency(car)}\}.$$

On the contrary, the set $\{\neg\texttt{park(car,bus\_lane)}\}$ is not a valid partial scenario, since it is not a scenario for $\mathcal{S}$ (it violates condition 4 of Definition 3.11).

$s_0^1 = s_1$ and $s_0^2 = s_2$ are valid completions of s_0, as is any other partial scenario. s_2 is not a valid completion of s_1, since both mention the atom

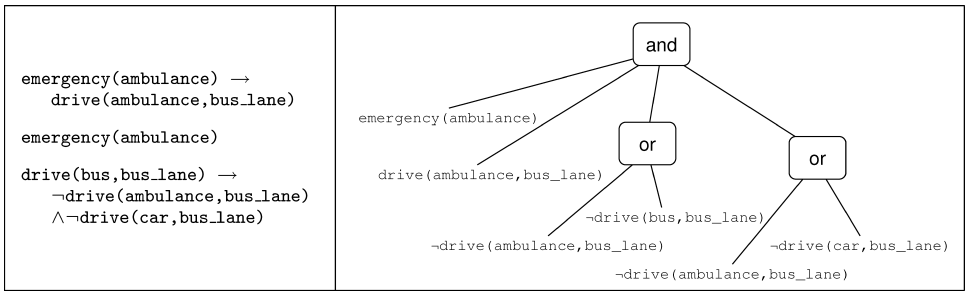


Fig. 3. A set of requirements (left) and the representation of its model as a CNF tree (right).

`emergency(ambulance)`. The following are some of the valid completions of s_1:

$$s_1^1 = \{\texttt{drive(car,bus\_lane)}\},$$
$$s_1^2 = \{\neg\texttt{drive(car,bus\_lane)}\},$$
$$s_1^3 = \{\texttt{drive(car,bus\_lane)}, \neg\texttt{drive(bus,bus\_lane)}\}.$$

The set {`drive(ambulance,bus_lane)`} is not a valid completion of s_1, since it is not a valid scenario for $(\mathsf{F} \cup s_1, \mathsf{H}, \mathsf{C})$. In fact, $\mathsf{F} \cup s_1 \models$ `drive(ambulance,bus_lane)`, thus violating condition 3 of Definition 3.11.

3.2 Notes on the Implementation

After having introduced formally our framework for the treatment of inconsistency, we can turn our attention to how its most important operations can be implemented in practice.

Our prototypical tool CARL maintains two representations of the requirements: (1) as a relational database for long-term persistency, also storing administrative details and house-keeping information about each requirement, and (2) as a set of bounded-depth, in-memory trees of conjunctive normal form (CNF) expressions, one for each bucket. In the tree representation, exemplified in Figure 3, the root always consists of a single n-ary conjunction (and) node, whose children are either (positive or negative) atoms, or n-ary disjunction (or) nodes, whose children are in turn positive or negative atoms. A number of simplification rules are applied every time the tree is updated, so as to maintain its structure and minimality. The tree is guaranteed to be equivalent to the conjunction of all the requirements in the set.

As we have seen, the user can insert new requirements into the specification by invoking the Rev() operator. To operationalize Rev($(\mathsf{F}, \mathsf{H}, \mathsf{C}), \alpha$), CARL first converts α into CNF, thus reducing it into a canonical form α'. If α' is tautologically false (e.g., `emergency(ambulance)`$\wedge\neg$`emergency(ambulance)`), or tautologically true, it is rejected altogether, with an appropriate error message. If α' is already in F, nothing needs to be done, and CARL informs the user about the fact.[4] If α' is found in H or C, CARL informs the user that it will be now accepted as

[4]Notice that the requirement may still be added to the database, for example, if the preexisting copy was requested by a different stakeholder.

a fact—that is, it will be moved from H or C to F—and asks for confirmation before proceeding.

If confirmation is given (or not needed), CARL adds α' to the F tree, and checks, via a variant of the standard semantic tableaux algorithm (see, for example, Reeves and Clarke [1990]), whether the resulting set is consistent. If it is, nothing else needs to be done, by Definition 3.6 and AGM postulates (^{*3}), (^{*4}), and ($^-3$) (see Appendix A). Otherwise, all maximal extensions of the new F that include α' are enumerated, and presented to the user (as natural language descriptions, as we will see in Section 4), in decreasing order of cumulative priority. The user is then asked to choose which one, among the various maximal extensions, should be used to continue the analysis—or, equivalently, which one among the conflicting requirements should be demoted to the state of hypothesis, as we have seen in Example 3.8. Let us stress once more that by so doing the inconsistency is *not* eliminated, but rather recorded and tolerated until it can be resolved at some future time. Once a maximal extension M has been chosen, H and C are adjusted according to Definition 3.6, and a log record of the operation is stored in the database for future reference and traceability. The treatment of $\mathsf{Con}((\mathsf{F},\mathsf{H},\mathsf{C}),\alpha)$ is similar.

Scenario operations are implemented as follows. At any stage, the user can enter a partial scenario, that is, a set of plain facts (each corresponding to a single positive or negative atom in $H(\mathsf{F}\cup\mathsf{H}\cup\mathsf{C})$). These partial scenarios can be assigned a name, saved, reopened, and edited at any time. When asked to perform an S-consistency check, CARL presents to the user a list of the partial scenarios that have been saved, supplemented with the empty partial scenario $s_0 = \{\}$. The user can then choose one or more partial scenarios against which S-consistency has to be checked. For each selected partial scenario s_i, CARL enumerates all possible completions s_i^j, and checks whether $\mathsf{F}\cup\mathsf{C}\cup s_i\cup s_i^j$ is consistent. If it is not (i.e., $\Sigma(\mathcal{S}, s_i\cup s_i^j)$ is false), a latent inconsistency has been discovered, and $s_i\cup s_i^j$ are the conditions under which the inconsistency is exposed. Of course, choosing the empty partial scenario results in all possible scenarios being checked.

In particular, for each s_i the tool computes first the set U of *unbound* atoms, that is, 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 completions are generated, by considering all the possible subsets u of U, and assigning all the combinations of truth values to members of u.[5] Scenarios which lead to an inconsistency are then presented to the user for further analysis and investigation. These can also be saved, becoming themselves new partial scenarios, to be used in the subsequent evolution of the requirements.

[5]To obtain reasonable performances, particular care is taken to avoid generating scenarios whose outcome is already known, that is, any scenario v with $v\subseteq u$ such that $u\cup\mathsf{F}\cup\mathsf{C}$ is already known 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. The latter condition has the effect of restricting the computation to the minimal problematic scenarios of Definition 3.15.

Finally, the user can ask CARL to show the consequences, or *implied facts* entailed by the requirements (optionally augmented by any currently selected partial scenario). Due to the way the CNF tree representation is maintained, these can be accessed directly by taking all the first-level children of the root that consist of a single atom. As an advanced feature, the user can also ask to be shown the *complex implications* of the requirements that are given by the or children of the root. However, these complex implications are usually more difficult to interpret in a meaningful way; hence we do not consider this to be standard usage of the tool.

3.3 Summary

To summarize, in this section we have provided a formal definition of a model for the controlled evolution of a set of requirements. In our model, multiple stakeholders work on a specification by revising and contracting requirements; if any of these operations causes an inconsistency, a set of alternative maximal extension is generated. The stakeholders can then analyze the situation, and choose a consistent subsets of the requirements to continue working with. In any case, their decision is recorded (either in H or in C), and can be changed at a later stage without loss of information.

Moreover, an automatic scenario analysis can be performed at any time on a specification, in order to discover combinations of events in the real world that could trigger latent inconsistencies in the specification. The stakeholders can then analyze the problematic scenarios and decide which modifications to the specification are needed to deal with the problems that were found.

In order to meet our usability goals, however, all those interactions between the stakeholders and the support system must be made as simple and intuitive as possible. This is the subject of the next section.

4. BRINGING LOGIC TO THE USER

In a typical industrial or commercial setting, 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 themselves, some kind of common language must be used to express the requirements in the course of the RE process.

We advocate that natural language can serve this role, provided that some restrictions are placed on its use in order to encourage the expression of precise requirements. Controlled natural language of this sort has long been suggested as an effective means for recording and discussing requirements [Dalianis 1992; Fantechi et al. 1994; Gervasi and Nuseibeh 2002; Hars 1996; Juristo et al. 2000; Macias and Pulman 1993; Mich 1996; Rolland and Proix 1992]. In order to allow the stakeholders to express their requirements in natural language, some form of automatic translation of requirements expressed in restricted NL into logic formulae and back is needed. Several studies exist on the translation of natural language into logic of different kinds, including Ali [1994], Fantechi et al. [1994],

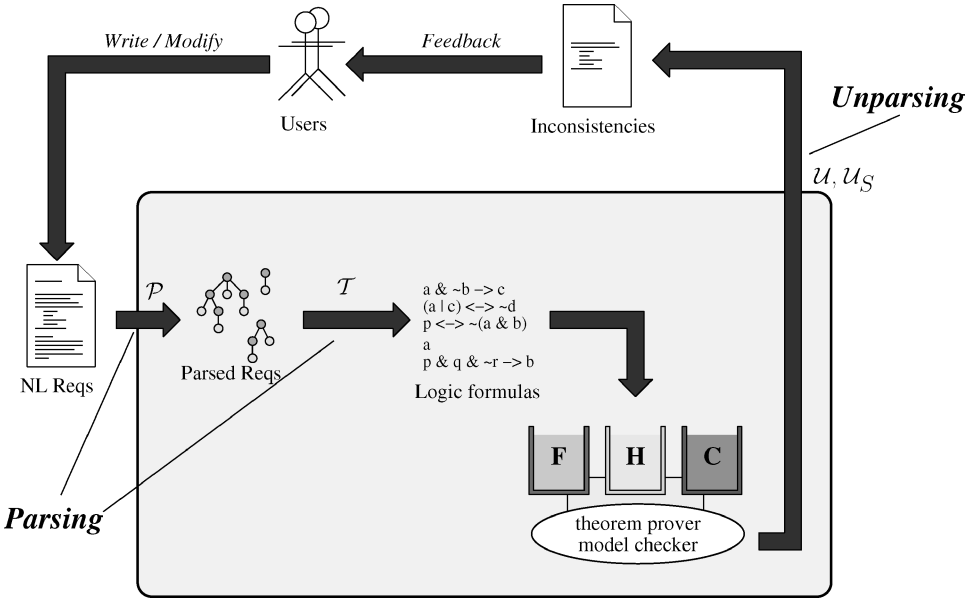


Fig. 4. Processing of requirements in the CARL system. NL Requirements are first parsed to produce a set of parse trees, then the trees are translated into logic formulae and analyzed to discover and report any inconsistency.

Fuchs and Schwitter [1995], Rolland and Proix [1992], and Webber [1983]. In most cases, these studies have targeted special-purpose logics (e.g., Linear Temporal Logic), which has resulted in very restrictive controlled languages. In contrast, in this work we maintain an information extraction perspective, and try to accept a less restricted language.

In order to support the various operations that have been defined in Section 3, we need to define a *parsing process* to translate NL sentences into logic formulae, and an *unparsing process* to translate back those formulae (or more precisely, a subset thereof) into NL sentences. The entire process is depicted in Figure 4 (the reader may also want to refer to Figure 2 for a global view on the functions and domains that will be used in the formal definitions in the following).

The first step of the parsing process consists of a preprocessing stage devoted to typographical adjustments, tokenization, and morphosyntactic analysis. During this step, multiple-word terms and domain-specific terms and acronyms defined in a user glossary are converted into single tokens. All words that do not appear either in a standard English dictionary or in the user glossary are at this stage marked as potentially erroneous, and the user is prompted to confirm their correctness (thereby adding them to the user glossary). The preprocessed text is then parsed, 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 the Rev or Con operators for consistency checking and adding to the specification. The whole process is exemplified in Figure 5 (but see also Section 5 for more complex

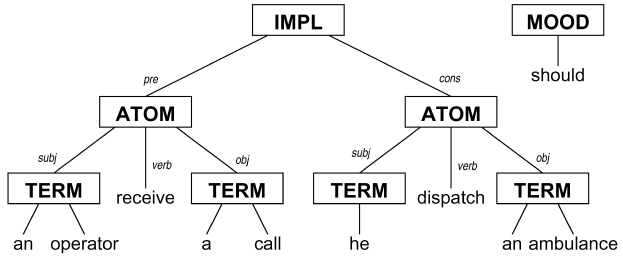
Original requirement	`When an operator receives a call, he should dispatch an ambulance.`
After morpho-syntactic analysis	`When/CC/WRB an/DT operator/NN receive/VB/Z a/DT call/NN he/PP should/MD dispatch/VB an/DT ambulance/NN`
Parse tree, as produced by the CICO algorithm ($\mathcal{P}$)	
Equivalent logic formula ($\mathcal{T}$)	`receive(operator,call)` $\rightarrow$ `dispatch(operator,ambulance)` (with priority set to "optional")

Fig. 5. An example of how NL requirements are translated into predicate logic.

examples), where the tags used in morphosyntactic annotation are derived from those in the Penn Treebank set [Marcus et al. 1993] (e.g., `/WRB` stands for adverb, `/NN` for noun, `/VB` for verb, etc.). The intermediate nodes in the parse tree have the following meaning: a TERM node represents an entity in the domain of discourse, or a referent to such a term; 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—be it causal, temporal, or otherwise—between an antecedent (premise) and a consequent (consequence).

Morphosyntactic analysis on the text is performed with the help of TreeTagger [Schmid 1994], 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. Moreover, part of the information that is normally conveyed through the various affixes (e.g., the mood of verbs) is stripped from the base form and moved to tags, where it can be more easily analyzed in later stages.

The parsing proper is performed by using the CICO algorithm, described more fully in Ambriola and Gervasi [1999] and Gervasi [2000, 2001]. The algorithm employs domain-based parsing, 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, the same algorithm can also be used as a generic syntax-based parser, by taking (a controlled 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 Gervasi [2001] for details).

A parsing rule consists of three parts: a template M for the fragment that is being matched, an action A that records the intended semantics of the fragment (as a node in the parse tree), and a substitution S that replaces the matched fragment in the text. Parsing rules are also called *MAS rules*, from the three components above. Matchings can be either literal (i.e., a specific keyword is found) or based on the tags assigned to a term.

For example, the template `a/DT/0 neg/NOT/0 adj/JJ/0 n/NN` matches an optional (as denoted by `/0`) article, followed by an optional negation, followed again by an optional adjective, and finally followed by a mandatory noun. Thus, the template matches fragments like *the ambulance* or *nonmedical emergencies*. Notice also that, due to the fuzziness of the scoring system, the same rule would also match variations like *a desperately confused operator* (*desperately* would be ignored, and *a confused operator* would be matched) and *an ambulance, broken* (parsed as *a broken ambulance*). The corresponding action for this model could be `TERM $a $neg $adj $n`, meaning that we want to record occurrences of this template as a `TERM` node in the parsing tree, having as children the subtrees corresponding to all the constituents of the fragment. Finally, we could specify a substitution of `$ID/TERM`, meaning that we want the whole fragment replaced by a reference to its corresponding node in the parse tree (by using the special variable `$ID`), and that we want this reference to have type `/TERM` for the purpose of subsequent matchings. The full rule for this case would then be

$$\frac{\dfrac{\text{a/DT/0 neg/NOT/0 adj/JJ/0 n/NN}}{\text{TERM \$a \$neg \$adj \$n}}}{\text{\$ID/TERM}}\,.$$

While we certainly do not make any claim of completeness, the parser handles a number of common linguistic phenomena, that is, reference, characterization through adjectives and relative clauses, negation, syndetic and asyndetic collation, etc. It also exhibits remarkable robustness against perturbation in the source text, so that even statements that do not conform exactly to the restrictions placed on the language are usually parsed correctly. We call the (fuzzy) language accepted by the parser *Sent*. The parsing rules used at this stage are shown in Table I. The set of all possible parsing trees is called *Tree*; one of the elements of *Tree* is shown in Figure 5.

Formally, the application of the parsing algorithm to a sentence is defined as follows:

Definition 4.1 (*Parse Tree*). Given a sentence r, the corresponding *parse tree* is given by the function $\mathcal{P} : \mathit{Sent} \rightarrow \mathit{Tree}$, where $\mathcal{P}$ is obtained by applying to r the CICO parsing algorithm as described in Gervasi [2001], with the set of MAS rules shown in Table I.

Table I. The MAS Parsing Rules Used in CARL

Template	Action	Substitution
a/TERM THAT $neg/NOT vb/VB		$a AND $a $neg $vb
a/TERM THAT IS/VB $neg/NOT adj/JJ		$a AND $a IS $neg $adj
a/TERM THAT IS/VB adj/JJ		$a AND $a IS $adj
a/TERM THAT vb/VB		$a AND $a $vb
a/TERM neg/NOT vbing/VB/G		$a AND $a $neg $vbing/-G
a/TERM vbing/VB/G		$a AND $a $vbing/-G
vb/VB/N		$vb/-VB/-N/JJ
a/SENT AND b/SENT	AND $a $b	$ID/SENT/AND
a/SENT b/SENT/AND	AND $a $b	$ID/SENT/AND
a1/JJ a2/JJ	CONJ $a1 $a2	$ID/JJ/CONJ
a/SENT ONLY IF b/SENT	EQV $b $a	$ID/SENT
a/SENT ONLY WHEN b/SENT	EQV $b $a	$ID/SENT
existential/EX BE/VB a/TERM	EXIST $a	$ID/SENT
IF a/SENT THEN/0 b/SENT	IMP $a $b	$ID/SENT
WHEN a/SENT b/SENT	IMP $a $b	$ID/SENT
a/SENT IF b/SENT	IMP $b $a	$ID/SENT
a/SENT WHEN b/SENT	IMP $b $a	$ID/SENT
aux/MD neg/NOT v/VB	MOOD $aux	$neg $v
aux/MD v/VB	MOOD $aux	$v
DO neg/NOT v/VB	MOOD absolute	$neg $v
DO v/VB	MOOD absolute	$v
a/SENT OR b/SENT	OR $a $b	$ID/SENT/OR
a/SENT b/SENT/OR	OR $a $b	$ID/SENT/OR
prep1/PREP prep2/PREP	PCONJ $prep1 $prep2	$ID/PREP/CONJ
prep/IN term/TERM	PREP $prep $term	$ID/PREP
to/TO term/TERM	PREP $to $term	$ID/PREP
a/TERM BE/VB neg/NOT/0 b/JJ compl/PREP/0	SENT $neg $b $a $compl	$ID/SENT
subj/TERM neg/NOT/0 verb/VB obj/TERM/0 compl/PREP/0	SENT $neg $verb $subj $obj $compl	$ID/SENT
a/TERM BE/VB neg/NOT/0 b/TERM compl/PREP/0	SENT $neg ISA $a $b $compl	$ID/SENT
a/DT/0 neg/NOT/0 adj/JJ/0 n/NN	TERM $a $neg $adj $n	$ID/TERM
it/PP	TERM _ _ _ $it	$ID/PPREF/TERM
EITHER a/SENT OR b/SENT	XOR $a $b	$ID/SENT
a/SENT UNLESS b/SENT	XOR $a $b	$ID/SENT
a/TERM BE/VB EITHER b/JJ OR c/JJ	XOR2 $b $a $c $a	$ID/SENT
a/TERM BE/VB EITHER b/TERM OR c/TERM	XOR2 $b $a $c $a	$ID/SENT
a/TERM v/VB EITHER b/TERM OR c/TERM	XOR3 $v $a $b $v $a $c	$ID/SENT

Example 4.2. Consider the sentence r from Figure 5:

When an operator receives a call, he should dispatch an ambulance.

The parse tree returned by the parsing algorithm is shown in graphical form in the same figure, and can also be written in textual notation as

$\mathcal{P}(r) =$ IMPL(ATOM(TERM(an/DT, _, _, operator/NN),
receive/VB/Z,
TERM(a/DT, _, _, call/NN)),
ATOM(TERM(_, _, _, he/PP),
dispatch/VB,
TERM(an/DT, _, _, ambulance/NN))
)

where _ denotes the empty matching of an optional part of the template. Notice also that the pronoun *he* is considered a TERM on its own at this stage; it will be substituted at a later stage by an appropriate reference.

Finally, the last stage of the parsing process is devoted to the translation of parse trees into logic formulae. This translation 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[6] dispatch an ambulance.

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.

[6]Modal verbs are not directly translated into logic, but are used to assign a priority, or importance to the speaker, to the resulting formula. Priorities are used to compare alternative maximal extensions according to the importance of the information that is retained, as discussed on pages 285–286.

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 satisfy 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 a nearby ambulance.

is translated as

```
medical(emergency) ∧ nearby(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 documents; however, they prove sufficient for many applications. If needed, the language accepted by the parser can be extended by writing appropriate MAS rules.

Formally, the translation into logic of a SENT node s is given by induction on the structure of parse trees. We define four semantic interpretation functions:

—$\mathcal{N} : \mathit{Tree} \rightarrow \mathit{Term}$ is a function that associates a node (of type TERM or PREP) to a logic term that represents the real-world object denoted by the node;
—$\mathcal{Q} : \mathit{Tree} \rightarrow \mathit{Asr}$ is a function that associates a node to a logic formula that qualifies the generic terms that are referred in the subtree rooted at the node;
—$\mathcal{F} : \mathit{Tree} \rightarrow \mathit{Asr}$ is a function that associates a node to a logic formula that represents the relationships that the node establishes among its constituents;
—$\mathcal{T} : \mathit{Tree} \rightarrow \mathit{Asr}$ is the main function that defines the translation of a whole parse tree into predicate logic.

The naming function $\mathcal{N}$ is applied to basic nodes representing terms or complements (i.e., terms preceded by a preposition) to obtain the name of the atom that represents the term. It is defined on nodes of type TERM and PREP as follows:

Definition 4.3 (*Naming Function*).

$$\mathcal{N}(\text{TERM}(art, neg, adj, noun)) = noun$$
$$\mathcal{N}(\text{PREP}(prep, term)) = \mathcal{N}(term)$$

Although not shown in the definition, the naming function also selects the proper reference for TERMs that represent pronouns, and returns the naming function of the referred node instead, heuristically defined as the nearest among the precedent potential foci that are valid in a particular logic context.

The qualification function $\mathcal{Q}$ collects the restrictions posed on terms by the use of adjectives, for example, the term *an emergency call* refers only to those calls that concern an emergency. These qualifications are expressed as predicates and collected[7] through the whole tree:

Definition 4.4 (*Qualification Function*).

$$\begin{aligned}
\mathcal{Q}(\text{TERM}(art, neg, adj, noun)) &= \begin{cases} adj(noun) & \text{if } neg \text{ is empty,} \\ \neg adj(noun) & \text{otherwise,} \end{cases} \\
\mathcal{Q}(\text{PREP}(prep, term)) &= \mathcal{Q}(term), \\
\mathcal{Q}(\text{PCONJ}(prep_1, prep_2)) &= \mathcal{Q}(prep_1) \cup \mathcal{Q}(prep_2), \\
\mathcal{Q}(\text{SENT}(neg, vb, subj, obj, compl)) &= \mathcal{Q}(subj) \cup \mathcal{Q}(obj) \cup \mathcal{Q}(compl), \\
\mathcal{Q}(\text{AND}(s_1, s_2)) &= \mathcal{Q}(s_1) \cup \mathcal{Q}(s_2), \\
\mathcal{Q}(\text{OR}(s_1, s_2)) &= \mathcal{Q}(s_1) \cup \mathcal{Q}(s_2), \\
\mathcal{Q}(\text{NOT}(s)) &= \mathcal{Q}(s), \\
\mathcal{Q}(\text{XOR}(s_1, s_2)) &= \mathcal{Q}(s_1) \cup \mathcal{Q}(s_2), \\
\mathcal{Q}(\text{XOR2}(p_1, t_1, p_2, t_2)) &= \mathcal{Q}(t_1) \cup \mathcal{Q}(t_2), \\
\mathcal{Q}(\text{XOR3}(vb_1, t_{11}, t_{12}, vb_2, t_{21}, t_{22})) &= \mathcal{Q}(t_{11}) \cup \mathcal{Q}(t_{12}) \cup \mathcal{Q}(t_{21}) \cup \mathcal{Q}(t_{22}), \\
\mathcal{Q}(\text{IMP}(s_1, s_2)) &= \mathcal{Q}(s_1) \cup \mathcal{Q}(s_2), \\
\mathcal{Q}(\text{EQV}(s_1, s_2)) &= \mathcal{Q}(s_1) \cup \mathcal{Q}(s_2).
\end{aligned}$$

The logic form function $\mathcal{F}$ is applied to a subtree representing a sentence to compute its equivalent logic formula:

Definition 4.5 (*Logic Form Function*).

$$\begin{aligned}
\mathcal{F}(\text{SENT}(neg, vb, subj, obj, compl)) &= \begin{cases} vb(\mathcal{N}(subj), \mathcal{N}(obj), \mathcal{N}(compl)) & \\ \qquad\qquad \text{if } neg \text{ is empty,} & \\ \neg vb(\mathcal{N}(subj), \mathcal{N}(obj), \mathcal{N}(compl)) & \\ \qquad\qquad \text{otherwise,} & \end{cases} \\
\mathcal{F}(\text{AND}(s_1, s_2)) &= \mathcal{F}(s_1) \wedge \mathcal{F}(s_2), \\
\mathcal{F}(\text{OR}(s_1, s_2)) &= \mathcal{F}(s_1) \vee \mathcal{F}(s_2), \\
\mathcal{F}(\text{NOT}(s)) &= \neg \mathcal{F}(s), \\
\mathcal{F}(\text{XOR}(s_1, s_2)) &= \mathcal{F}(s_1) \leftrightarrow \neg \mathcal{F}(s_2),
\end{aligned}$$

[7]An internal renaming is performed if a single noun occurs multiple times in conjunction with different attributes, so that the various instances can be told apart in the logic form of the requirement.

$$\begin{aligned}
\mathcal{F}(\text{XOR2}(p_1, t_1, p_2, t_2)) &= p_1(\mathcal{N}(t_1)) \leftrightarrow \neg p_2(\mathcal{N}(t_2)),\\
\mathcal{F}(\text{XOR3}(vb_1, t_{11}, t_{12}, vb_2, t_{21}, t_{22})) &= vb_1(\mathcal{N}(t_{11}), \mathcal{N}(t_{12})) \leftrightarrow \neg vb_2(\mathcal{N}(t_{21}), \mathcal{N}(t_{22})),\\
\mathcal{F}(\text{IMP}(s_1, s_2)) &= \mathcal{F}(s_1) \rightarrow \mathcal{F}(s_2),\\
\mathcal{F}(\text{EQV}(s_1, s_2)) &= \mathcal{F}(s_1) \leftrightarrow \mathcal{F}(s_2).
\end{aligned}$$

Finally, the final translation function collects all the qualifications in a parse tree with root r and composes them with the logic form of the whole tree:

Definition 4.6 (*Logic Translation Function*).

$$\mathcal{T}(r) = \begin{cases} \mathcal{Q}(r) \rightarrow \mathcal{F}(r) & \text{if } \mathcal{Q}(r) \neq \emptyset \\ \mathcal{F}(r) & \text{otherwise} \end{cases}$$

Based on the properties of the parsing algorithm given in Gervasi [2001], and on the above definitions, we state the following:

Conjecture 4.7. Given a sentence r in the fuzzy language defined by the parsing rules of Table I, $\mathcal{T}(\mathcal{P}(r))$ is a logic formula that corresponds to the intuitive meaning of r.

The equivalence of the translated logic form to the intended meaning of the original NL text, of course, can only be conjectured, and not proved. However, as we will show in Section 5, even a relatively simple translation schema as the one presented above can provide good results on most types of common requirements. The interested reader can refer to Appendix B for a comparison between the translation strategy we presented here and human performance at the same task.

The last remaining problem concerns the translation of logic formulae back into natural language sentences, in order to provide meaningful feedback to the user. While the task may seem hard, a careful analysis of the kind of feedback that we intend to produce reveals that only two types of translation are needed.

First, when reporting inconsistencies and proposing alternative maximal extensions to choose from, only assertions corresponding to entire requirements are considered. Since linkage information connecting assertions to their original text is maintained both by the parser and by the inference engine, in this case it is sufficient to retrieve the original statement corresponding to each assertion.

Second, when presenting scenarios, only simple atoms are to be unparsed, according to Definition 3.11. To this end, a simple unparsing schema based on the rules in Table I suffices. A number of grammatical and lexicographical features are also taken into account, in order to synthesize "natural looking" sentences. We will not go into the linguistic details in this article; in general terms, the translation of these classes of logic formulae into NL text is given by the following definition:

Definition 4.8 (*Unparsing*). Given a logic formula $l \in Asr$, a set of NL requirements $\mathcal{R}$, and a specification $\mathcal{S} = (\mathsf{F}, \mathsf{H}, \mathsf{C})$ obtained by successive applications of the $\mathsf{Rev}(\mathcal{S}, \alpha)$ and $\mathsf{Con}(\mathcal{S}, \alpha)$ operators, where $\alpha = \mathcal{T}(\mathcal{P}(s))$ and $s \in \mathcal{R}$,

the *unparsed version* of l is given by the function $\mathcal{U} : Asr \rightarrow Sent$, defined as

$$\mathcal{U}(l) = \begin{cases} s \in \mathcal{S} \text{ s.t. } \mathcal{T}(\mathcal{P}(s)) \equiv l & \text{if } l \in \mathsf{F} \cup \mathsf{H} \cup \mathsf{C}, \\ \textsc{Unparse}(l) & \text{if } l \text{ is an atomic predicate and } l \notin \mathsf{F} \cup \mathsf{H} \cup \mathsf{C}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where UNPARSE() represents the inverse application of the MAS rules in Table I for ATOM (together with a few aestethic adjustments of the text).

Moreover, given a set of logic formulae $\{l_1, \ldots, l_n\}$, we define the set-oriented version of the unparsing function as

$$\mathcal{U}_S(\{l_1, \ldots, l_n\}) = \mathcal{U}(l_1) \cdot \ldots \cdot \mathcal{U}(l_n),$$

where $\cdot$ denotes the append operation for sentences.

It should be noted that in the first case of the definition above, several different s can satisfy the condition stated. In this case, we consider them to be purely syntactic variations of the same fact, all equally acceptable for feedback purposes, and we could nondeterministically choose a natural language version to return. While in theory the choice of which version to present is immaterial, in practice it is preferable to present a deterministically chosen version to the user, in order to maintain consistency of the representation and simplify the analysis process. Moreover, the algorithm implementing the $\mathcal{U}_S$ function in our prototype CARL also sorts sentences so that the first occurrence of a term does not appear in the role of a subject, if possible, and so that other occurrences appear as near as possible to the first one. This improves the perceived naturalness of the resulting text (see Section 5 for examples).

We can now state our second conjecture:

Conjecture 4.9. Under the conditions given in Definition 4.8, the intuitive meaning of $\mathcal{U}(l)$ corresponds to l.

Remark 4.10. The parsing function $\mathcal{T}(\mathcal{P}())$ and the unparsing function $\mathcal{U}()$ are naturally meant to be dual. In fact, for any requirement r in the fuzzy language described by the rules in Table I, we have that

$$\mathcal{U}(\mathcal{T}(\mathcal{P}(r))) \equiv r,$$

where equivalence is understood up to purely syntactical variations. Moreover, given a set of requirements $\mathcal{R}$, we have that

$$\forall l \in H_R(\mathcal{R}), \mathcal{T}(\mathcal{P}(\mathcal{U}(l))) \equiv l,$$

where $H_R(\mathcal{R})$ is the Herbrand base for the assertions resulting from $\mathcal{R}$, that is:

$$H_R(\mathcal{R}) = \bigcup_{r \in \mathcal{R}} H(\mathcal{T}(\mathcal{P}(r))).$$

These observations support our conjectures about the equivalence between the natural language form and the logic form of the requirements that comprise a specification.

Example 4.11. Let us consider the requirement r below:

$r =$ *When an operator receives a call, he should dispatch an ambulance.*

As shown in Figure 5 and discussed in Example 4.2, the parsing process produces the parse tree $\mathcal{P}(r)$, whose translation into logic is

$$\mathcal{T}(\mathcal{P}(r)) = \texttt{receive(operator,call)} \rightarrow \texttt{dispatch(operator,ambulance)}.$$

As per Conjecture 4.7, we assume this formula to be equivalent to the NL requirement. In the course of the analysis, we may have to warn the user about an inconsistency, and to present her with a number of possible alternative extensions to choose from. In this case, we may have to translate back the formula into natural language. In our case, we have

$$\mathcal{U}(\texttt{receive(operator,call)} \rightarrow \texttt{dispatch(operator,ambulance)})$$
$$= \textit{When an operator receives a call, he should dispatch an ambulance.}$$

since the first case of Definition 4.8 is taken. Alternatively, we may have to present a particular scenario to the user. In this case, the possible atoms that may appear in the scenario are

$$H_R(\{r\}) = \{\texttt{receive(operator,call)}, \texttt{dispatch(operator,ambulance)}\}.$$

Consider, for example, the scenario $s_1 = H_R(\{r\})$. In translating it into natural language, the second case of Definition 4.8 is taken, producing the following text:

$$\mathcal{U}_S(s_1) = \begin{array}{l}\text{An operator receives a call;}\\ \text{the operator dispatches an ambulance.}\end{array}$$

In both cases, we claim that the NL representations of the logic formulae preserve their intended meaning, in keeping with Conjecture 4.9.

5. AN EXAMPLE

To show how the techniques we presented in the previous sections are used in practice, we resort to a classic example concerning a computer-aided dispatch system employed by the London Ambulance Service (LAS). The general goal of the LAS is to service emergency phone calls, by efficiently dispatching ambulances if the emergency requires medical attention, or by forwarding calls to other emergency services otherwise. The LAS is also responsible for maintaining its fleet of ambulances in good operating condition.

The LAS system has been used as a common case study in a number of works, including our previous article [Zowghi et al. 2001]; an introduction and related references can be found in Finkelstein and Dowell [1996]. In particular, we follow here the steps of Hunter and Nuseibeh [1998], in which a subset of the requirements for the LAS system was considered.

The example has been run on CARL, our prototypical tool implementing (i) functions for parsing and unparsing NL sentences and translating them into propositional logic, based on the parsing algorithm in Gervasi [2001]; (ii) the Rev and Con operators, based on those presented in Zowghi et al. [1997]; (iii) a theorem prover based on semantic tableaux for checking consistency; (iv) a scenario generator; (v) a simple model checker for S-consistency. CARL also provides a graphical user interface and a repository to access and store

requirements between sessions. The repository can also be shared among many stakeholders, to support collaborative work.

In Hunter and Nuseibeh [1998], 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 in discriminating 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[8]:

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 a nearby available ambulance.
IRC_3	When an operator receives a phone call concerning a nonmedical 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 thus assume that the following requirements are expressed by the OM:

Operations Manager	
OM_1	When an operator receives a phone call, he should dispatch a nearby available ambulance.
OM_2	When an operator receives a phone call, if an ambulance is not nearby or not available, then the operator should not dispatch that 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 calls. However, the LM must ensure that only vehicles that are "in order" are used to carry the service on. We thus consider 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 maintenance.
LM_3	The operator does not dispatch an amb ulance if it is not available.

All these NL requirements can be entered, one at a time, in CARL's repository (see Figure 6), by invoking a revision operation. In addition to an identifier and

[8]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 "nearby" ambulance.

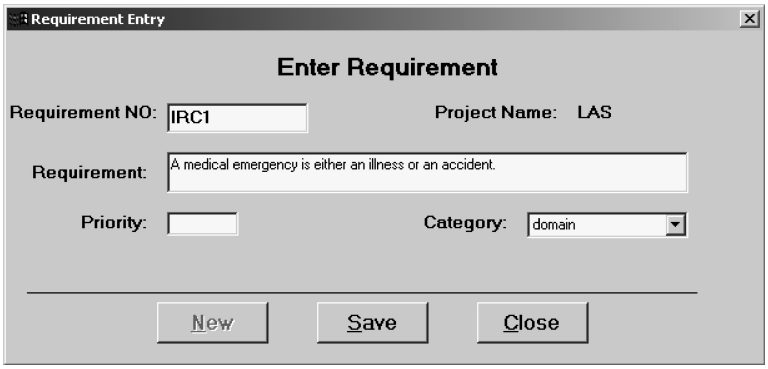


Fig. 6. Entering a NL requirement in CARL.

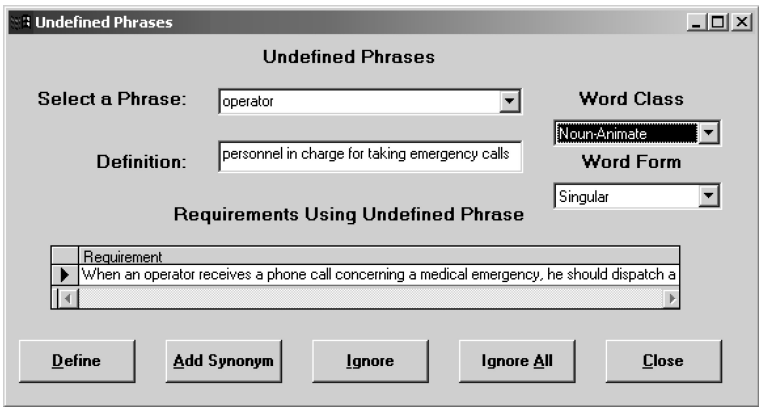


Fig. 7. Entering a glossary term in CARL.

to the requirement itself, the user can optionally enter a priority (overriding the default one that is derived from the mood of the main verb in the sentence) and a category for the requirement. The set of possible categories is user-defined; CARL does not use these labels except for display purposes.

Upon entering a requirement, the user is asked to define all unknown terms (e.g., acronyms and words not found either in a standard English dictionary or in the user glossary). The user can also define additional terms, for example, in case of multiword terms like `phone call`, to override the meaning and grammatical role extracted from the dictionary, or simply to document their intended meaning for documentation purposes (see Figure 7).

Automatic translation of the requirements above, according to the parsing and translation techniques outlined in Section 4, yields the set of logic formulae shown in Table II. As an advanced feature, the user could ask CARL to provide detailed feedback about the translation. The feedback provided would include the tagged form of the NL requirement, the corresponding logic formula, and the unparsed version of the same requirement. While tags and logic formulae are not intended for the normal user, the unparsed version could also be used as a paraphrase, to verify that CARL has indeed interpreted the sentence

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

Incident Room Controller		
IRC_1	medical(emergency) → (illness(emergency)↔ ¬accident(emergency))	[a]
IRC_2	(medical(emergency) ∧ nearby(ambulance) ∧ available(ambulance)) → ((receive(operator,phone_call) ∧ concern(phone_call,emergency)) → dispatch(operator,ambulance))	[o]
IRC_3	(¬medical(emergency)) → ((receive(operator,phone_call) ∧ concern(phone_call,emergency)) → (¬dispatch(operator,ambulance) ∧ transfer(operator,phone_call,service)))	[0]
Operations Manager		
OM_1	(nearby(ambulance) ∧ available(ambulance)) → (receive(operator,phone_call) → dispatch(operator,phone_call))	[o]
OM_2	receive(operator,phone_call) → (¬nearby(ambulance) ∨¬available(ambulance)) → (¬dispatch(operator,ambulance))	[o]
Logistics Manager		
LM_1	¬have(ambulance,crew) → ¬available(ambulance)	[a]
LM_2	(¬(nearby(ambulance) ∧available(ambulance))) →¬dispatch(operator,ambulance)	[a]
LM_3	last(year) → (¬revise(ambulance,year) → (¬available(ambulance) ∧ check(technician,ambulance,maintenance)))	[i]

according to the user's intentions. As an example, Figure 8 presents an example of feedback, where it is shown that the NL requirement originally entered as

> When an operator receives a phone call, he should dispatch a nearby available ambulance.

has been interpreted by CARL as

> If an ambulance is nearby and the ambulance is available, then if an operator receives a phone call, the operator dispatches[9] that ambulance.

None of the requirements in Table II cause any inconsistency *per se*; thus all of them are entered into the F bucket through the Rev operator. 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 CARL 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 understanding of the problem, or to check with the stakeholders whether the behavior resulting from

[9]Notice that the priority information provided by the modal verb *should* is missing in the paraphrase, but shown on the main list of requirements.

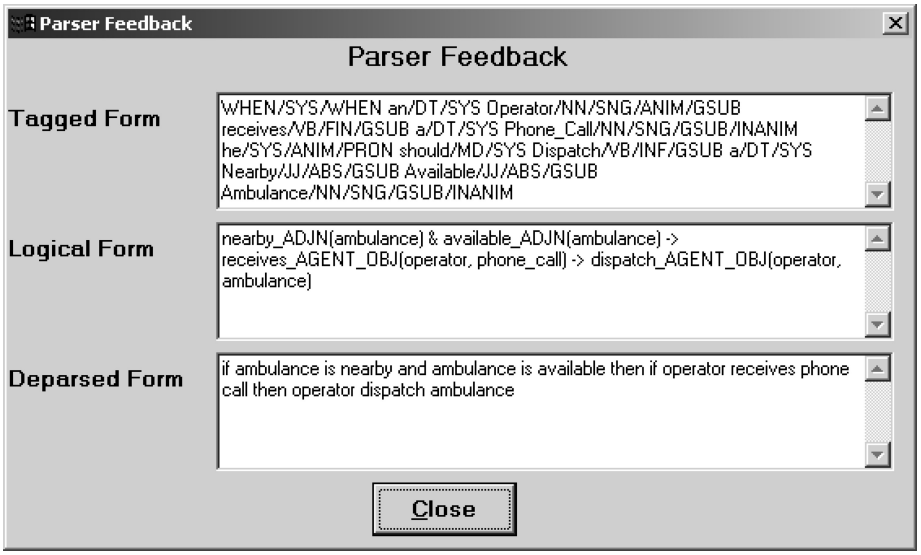


Fig. 8. The feedback provided by CARL on how NL requirements are translated into logic form and interpreted by the tool.

the combination of the requirements from all the stakeholder is acceptable to each of them.

For example, let us investigate what would happen if an ambulance without crew was considered available. To this end, we can create a new partial scenario and enter the sentence

```
An ambulance is available and it does not have a crew.
```

When asked to verify the consistency of this partial scenario, CARL identifies an explicit conflict (see Figure 9), and suggests possible ways of resolving it. In this case, there are two possibilities: either we renounce to the partial scenario, or the requirement LM_1 ("If an ambulance does not have a crew, it is not available") has to be sacrificed. Even if we decide for the latter option, the requirement need not be discarded: instead, it is simply demoted to a hypothesis (see Figure 10), and will return in force as soon as the (admittedly odd) partial scenario is recalled.

Even if we do not specify a scenario, inconsistencies could arise between different stakeholders' viewpoints. Extending Hunter and Nuseibeh [1998], suppose we identify two more stakeholders in the LAS system, the maintenance technician (MT) in charge of properly maintaining ambulances in good working conditions, and the human resources secretary (HR) who schedules the shifts for the ambulance crews. The MT cannot operate on ambulances that have crews on board; hence he could state the following requirement:

Maintenance Technician	
MT_1	Ambulances are available only when they have no crews.
MT_2	When an ambulance is available, a technician should check it for maintenance.

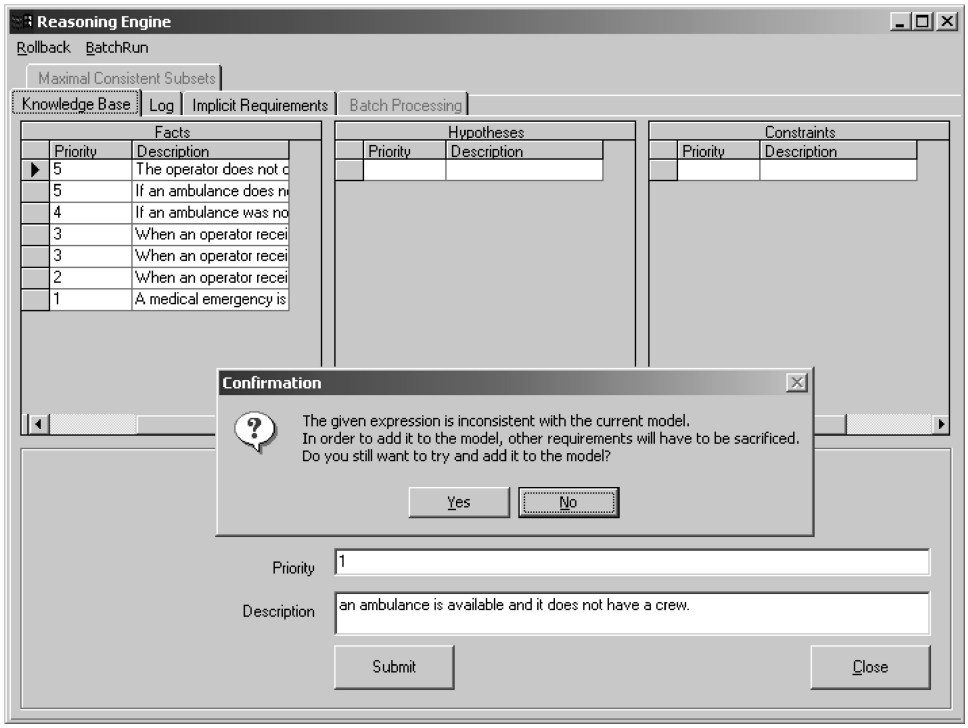


Fig. 9. A case of inconsistency: CARL has identified a conflict between the current specification and a new fact.

On the other hand, as far as HR is concerned, once a shift has been scheduled and a crew assigned to an ambulance, it is available:

Human Resources	
HR_1	A crew includes a driver, a medic, and a paramedic.
HR_2	When an ambulance has a crew, it is available.

If we try to revise the specification with these requirements that are translated into logic form as

Maintenance Technician		
MT_1	`available(ambulance)` $\leftrightarrow \neg$ `have(ambulance,crew)`	[a]
MT_2	`available(ambulance)` $\rightarrow$ `check(technician,ambulance,maintenance)`	[o]
Human Resources		
HR_1	`include(crew,driver,medic,paramedic)`	[a]
HR_2	`have(ambulance,crew)` $\rightarrow$ `available(ambulance)`	[a]

CARL reports again an inconsistency, akin to the one shown above. This time, the conflicting requirements are LM_1, MT_1, and HR_2, as the reader can easily verify. It is interesting to notice that any two of those requirements can be satisfied at the same time, whereas considering all three requirements together

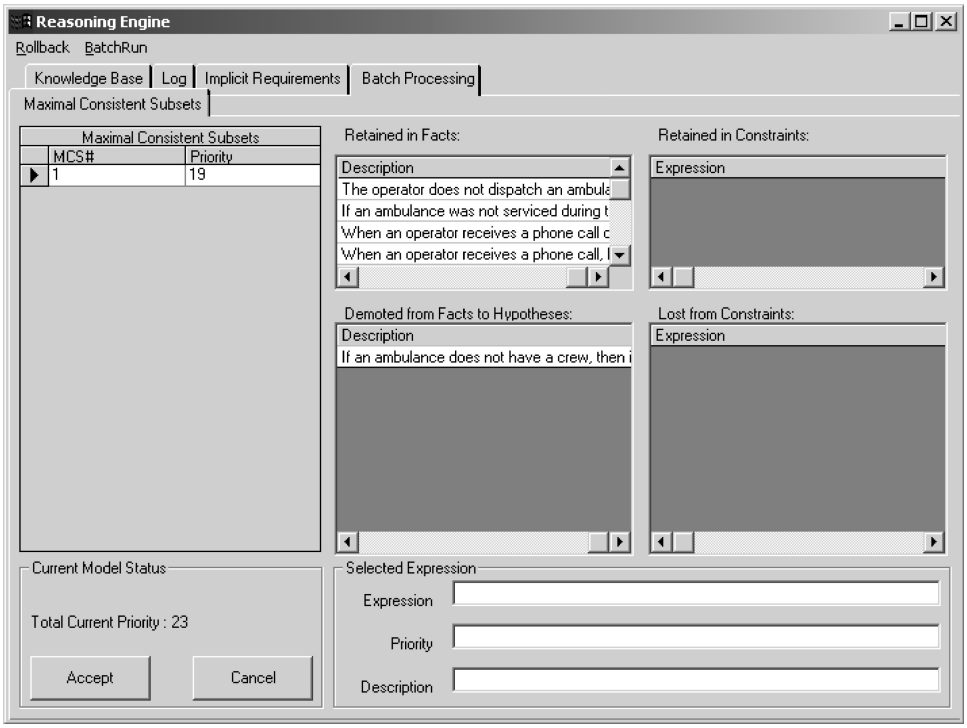


Fig. 10. A case of inconsistency: CARL suggests that a requirement previously considered a fact should be demoted to a hypothesis, since a conflicting statement has been added to the specification.

causes an inconsistency. This case emphasizes the complex nature of inconsistency, which is a property of set of requirements, and not a binary relation as some earlier work (e.g., Easterbrook and Nuseibeh [1996]) has postulated.

In our case, discussion with the three stakeholders involved might reveal that the term *available* has been used with different meanings: the LM intended *available for dispatching*, whereas the MT intended *available for maintenance*. Also, the second HR requirement as formulated is too strong, in that *available* is intended as *available as far as HR is concerned*, a concept different from both the LM and MT ones. In fact, an ambulance staffed with a crew might still not be available for dispatching (e.g., because it does not have enough fuel). In normal conditions, HR_2 could be considered a reasonable default, but not a sure fact: demoting it to a hypothesis would thus be appropriate.

CARL's glossary feature can help in resolving inconsistencies that are caused by *terminological ambiguity*, like the one above. In our glossary, `available` might have been defined as *ready to be dispatched to service an emergency call*, thus supporting the LM use of the term, and hinting that MT_1, MT_2, and HR_2 should be revised. Possible corrective actions include using different adjectives for the different meanings (e.g., `available` and `serviceable`), or adding a specifier (e.g., `available for dispatching` vs. `available for maintenance`).

As a more interesting example, let us turn back to the simple case where only IRC, OM, and LM are involved, and let us consider what happens in the

"normal" case for processing a phone call received by the Incident Room Controller, as expressed by the following partial scenario:

```
There is a medical emergency;
an operator receives a phone call concerning the emergency;
an ambulance is nearby 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)
nearby(ambulance)
available(ambulance)
```

is provisionally revised into the specification, without causing any inconsistency, and thus complementing what is in the F bucket. The absence of conflicts assures us that all the stakeholders agree on what the processing of the normal case should be. Moreover, we can inspect through CARL what other consequences can be deduced for this scenario, by checking for implied facts. Indeed, the system identifies these other facts as true:

—`dispatch(operator,ambulance)` from IRC_2 and also from OM_1: the operator actually dispatches a nearby available ambulance;

—`have(ambulance,crew)` from LM_1: the ambulance will be dispatched with a crew on board.

By presenting to the user the corresponding unparsed NL sentences, CARL can improve the confidence that a system implemented according to these requirements will behave as expected.

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(year_2004)` $\wedge$ $\neg$`revised(ambulance,year_2004)`. This scenario does not produce any inconsistency, and yields as consequences the following facts:

—$\neg$`available(ambulance)` from LM_3;

—`check(technician,ambulance,maintenance)` also from LM_3;

—$\neg$`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 Hunter and Nuseibeh [1998], is effective for confirming the analyst's or the stakeholders' expectations for certain cases. However, it does not help in checking the behavior of the system for *unexpected* scenarios. To this end, a more exhaustive analysis is needed: the one provided by the search for problematic scenarios via completions, starting with the empty partial scenario.

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

```
¬medical(emergency)
concern(phone_call,emergency)
receive(operator,phone_call)
nearby(ambulance)
available(ambulance)
```

The complete problematic scenario is automatically unparsed to NL form, and presented to the user as follows:

```
An operator receives a phone call;
the phone call concerns an emergency;
the emergency is not medical;
an ambulance is nearby;
the ambulance is available.
```

As can be observed, from this scenario both `dispatch(operator,ambulance)` and `¬dispatch(operator,ambulance)` can be derived. In fact, IRC_3 and OM_1 are the causes of inconsistency: OM_1 tacitly assumes that a phone call will *always* result in the dispatching of a nearby 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. Two maximal extensions are identified, and presented to the user as possible alternative ways of tolerating this particular inconsistency:

$\mathsf{F} \setminus \{IRC_3\}$, and
$\mathsf{F} \setminus \{OM_1\}$.

In other words, to remove the inconsistency under this scenario we have to give up either IRC_3 or OM_1. As usual, CARL also provides suggestions regarding the most optimal course of action, based on the priorities assigned to each requirement. In this case, the two candidates were both stated in the optative mood, with no specific priority assigned at the time they were entered. Hence, CARL suggests that any of them could be demoted, without giving preference to either.

In this example, our understanding of the problem would probably be sufficient to take a decision. But in a real case, the analyst might want to consult the stakeholders owning the inconsistent requirements (i.e., the Incident Room Controller and the Operations Manager), showing them the problematic scenario identified by CARL and the various alternatives, 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. As an important side effect, since OM_1 is in H, the Operations Manager knows that in certain cases his requirement will not be satisfied (i.e.,

when IRC_3 preempts it), and thus is encouraged not to rely on it unconditionally. In this way, a false sense of security is avoided, as is the buildup of excessive expectations—both of which are among the most common causes of failure in requirements elicitation processes.

6. DISCUSSION AND COMPARISON

In this section we discuss some of the limitations of our approach alongside with relevant mitigating factors, and compare the goal, scope, and spirit of our work to what other researchers have proposed in recent times.

First, propositional logic can be considered not expressive enough to model in sufficient detail and precision complex system behavior. In our experience, propositional logic was found to be adequate to express high-level requirements, but not the details of how the system should behave. These two types of requirements can be exemplified by considering the requirement that an ambulance must have been serviced. In propositional logic, this could be expressed simply as a high-level requirement, for example,

```
serviced(ambulance),
```

whereas a more detailed version could be expressed, in first-order logic, as a detailed requirement, for example:

$$\begin{aligned}&\forall a \in Ambulances,\\ &\quad \forall d \in DefectReports(a),\\ &\quad \exists m \in MaintenanceLog(a) : date(m) \geq date(d).\end{aligned}$$

It can be observed that while this second expression of the requirement is more complex, it provides much more information, in particular linking the concept of "maintaining an ambulance" with those of defect reports and maintenance log. However, in the early stages of requirements evolution, simpler requirements of the first type tend to prevail, and are more subject to negotiation between the stakeholders. Being able to reveal inconsistencies at this stage is thus useful even if all the details of the requirements cannot be stated precisely. Moreover, 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. This fact manifests itself in first-order logic theorem provers as the need for user input when trying to find an inductive hypothesis in complex proofs. As a consequence, when using first-order logic, it is not possible to provide totally push-button tools like CARL. However, push-button tools are the kind of instruments that are more readily accepted and deployed in industrial practice, and we believe this to be an extremely desirable feature of our approach. One possible solution to this problem is to restrict the language to a subset with known computational advantages such as Horn clause logic or Datalog [Abiteboul and Simon 1991; Ullman 1989]. Another alternative is to restrict it to first-order representations over finite domains, which can be instantiated down to the propositional level [Jackson 2000; Russo and Nuseibeh 2001]. This would allow proofs to continue to be carried out automatically, thus preserving our tenet that the user

should not be forced to plunge into the complexity of formal reasoning with logic.

Indeed, other researchers have explored the use of more sophisticated forms of logic. These studies have mainly concerned the usage of logic and model checking as tools for the specification and formal verification of properties of software systems, rather than in support of requirements engineering. Moreover, few of the studies have investigated the important issue of how usable these techniques are in a typical production context, that is, when experts in formal languages like modal action logic [Ryan 1993], multivalued logic [Chechik et al. 2003], QC-logic [Hunter and Nuseibeh 1998], or abductive logic [Russo et al. 2002] are not readily available. Even some recent studies specifically targeted at requirements engineering (e.g., Choi et al. [2002]; Eshuis et al. [2002]) have not addressed this issue. In contrast, our work has focused on providing lightweight and easy to use yet rigorous tools in support of the early phases of requirements elicitation and negotiation. We highlight two advantages of our approach with respect to the ones cited above: (i) our use of classical propositional logic makes the framework simple and elegant; we have explicitly avoided the complexity of introducing additional truth values (e.g., Chechik et al. [2003]; Huth and Pradhan [2004]) and a special notion of paraconsistency; instead, we developed two defeasible operators that act on a conveniently partioned knowledge base. (ii) The use of a natural language interface empowers even nonexpert stakeholder to profitably use our framework; all input to and output from our prototype tool is in plain yet rigorous English, thus helping the stakeholders in choosing a course of action when inconsistencies arise.

We believe that our approach is better suited to the initial phases of development, when the principal concerns are elicitation, clarification, and negotiation of the requirements. We envision that more thorough analysis can be carried on, if so desired, at a later stage, after the requirements have been agreed upon.

Another important difference with respect to other approaches is that we consider a dynamically evolving requirements base, whereas some previous work (e.g., Huth and Pradhan [2004]; Chechik et al. [2003]) has concentrated on static requirements that are inconsistent per se, or on cases where merging several static viewpoints expose latent inconsistency. Thus, we *manage* inconsistency during requirements evolution, rather than *finding* inconsistency in a set of already specified requirements. The same goal of managing inconsistency during evolution has been stated in related work [Garcez et al. 2003] where event-based descriptions and neural networks have been applied.

The second limitation concerns the kind of restricted NL defined in Table I, which is not expressive enough for many purposes. This, however, is not a major problem in itself, since both the parsing rules and the translation functions are modular and compositional, and allow easy expansion with more complex grammatical constructs, as needed by specific applications or domains.

In fact, in related research [Gervasi 2000; Gervasi and Ambriola 2002], the same parser has been used to analyze completely different languages, ranging from analyzing functional software specification, to information extraction from real estate ads, to identification of rethorical structures in literary text. More

specifically, the language recognized by CARL has been extended by adding a few domain-specific constructs in the context of an industrial case study, whose results will be reported in a companion article.

The third limitation concerns scalability. We have not addressed the issue of the scalability of the approach to industrial-strength cases in this article. Since parsing and translation are done one requirement at a time, as soon as requirements are entered into the system, the resources needed for that step are essentially constant. Moreover, the linguistic processing of a single requirement is fast enough (usually, well below 1 s) that it does not add significantly to the time needed to type it; hence, from a user interface point of view, this processing step introduces no perceivable overload. Transformation into conjunctive normal form for insertion into the tree-based representation of the various buckets has a complexity that grows linearly with the size of the logic-form representation of the requirement. Since the size of the logic-form representation is bound by the size of the natural language form, and this in turn does not normally exceed the standard size of a sentence, this step too does not give rise to scalability concerns.

However, theorem proving and model checking may require resources that, in the worst case, grow exponentially with the total number of requirements in a specification. CARL, being a research prototype rather than an industrial-strength tool, implements rather simplistic versions of both of them, but more efficient theorem provers and model checkers exist and could be used instead. Current state-of-the-art algorithms for model checking on contemporary hardware can handle thousands of propositional variables, or even millions if the model can be reduced to a satisfiability problem on structured formulae. This is a huge number of atoms for high-level requirements like the ones we are considering in this work; we thus expect that our technique, when used in conjunction with such algorithms, can scale reasonably well to real-life problems.

Moreover, the worst-case scenario rarely materializes in practice (e.g., an inconsistency caused by some fact derived through a derivation chain that encompasses *all* the requirements in a specification). Also, while the Rev() and Con() operators that are used more frequently during requirements evolution have efficient implementations, the most computationally expensive tasks (e.g., generating all the completions for the empty partial scenario) are only invoked on demand. Thus, it is possible to run such operations overnight, without disrupting the normal flow of interaction with the stakeholders.

Despite the mitigating factors cited above, further work is needed to overcome all these limitations. Moreover, other issues should be addressed, like integrating CARL with a requirements management tool to provide traceability, revision control, and other capabilities relevant for process management purposes. Although these features would improve the effectiveness of the tool, we consider them totally orthogonal to the analysis of inconsistencies that has been the primary focus of this work. Still, our framework provides valuable support for traceability and revision control:

—Since all changes to the requirements are implemented as applications of the Rev() and Con() operators, the state of the requirements specification at

a certain stage can be seen as the result of a repeated application

$$\mathcal{S} = \mathsf{Rev}(\mathsf{Rev}(\mathsf{Con}(\ldots \mathsf{Rev}(\mathcal{S}_0, \alpha) \ldots, \beta), \gamma), \delta).$$

CARL maintains a record of the applications of operators and of the extensions chosen when inconsistencies arise. That record could be supplemented with rationale information to provide an accurate history of the evolution of the requirements.

—By transforming natural language requirements into predicate logic formulae, a number of significant relationships among requirements can be automatically detected. For example: requirements that contain the same atom; requirements that entail other requirements; requirements that take part in the same inconsistency proof; requirements that describe actions taken under a given scenario, etc.

The automatic generation and management of these relationships would relieve analysts from the tedious and error-prone job of explicitly stating all the dependencies, as required by current commercial tools, instead allowing them to concentrate on the more important ones: those that cannot be derived syntactically because they have a purely semantic nature.

We intend to further investigate these issues as part of our future work.

7. CONCLUSIONS

This article has addressed an important and challenging issue in RE, one that concerns identifying and analyzing logical inconsistencies in natural language requirements. Inconsistencies of this kind have been 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 natural language can be effectively parsed and translated into logic. We have also demonstrated how these translated logical statements of requirements can be analyzed for inconsistency. Our approach to modeling and analyzing requirements is novel because it combines the expressiveness of natural language with preciseness, rigor, and formality of logic for handling inconsistencies in requirements specifications. We have also developed a number of tools and techniques in support of this approach: parsing and unparsing of NL sentences, translation of these sentences into logic and back to NL, operators for rational revision and contraction of specifications, theorem proving, and model checking to discover explicit or hidden inconsistencies in specifications and scenarios.

In fact, we have addressed two of the three problems whose solution has been deemed "essential" by Easterbrook and Chechik [2002]:

> Temporal logics can be hard to work with, and most people have difficulty in finding the correct logical expression for all but the simplest properties.[10]

[10]This position is indeed consistent with findings from our field study reported in Appendix B.

and

> The counter-examples produced by most model-checking engines do not mean anything to the stakeholders, and need to be translated back into the original modelling language.

(The third problem cited, that of state explosion in model-checking approaches, is not specific to requirements engineering, and is the main focus of research in the model-checking community.) Not only we have provided our stakeholders with a more natural interface, based on natural language rather than on logic, but we have also integrated model-checking techniques with theorem proving in the context of a default logic, thus faithfully modeling the complex interplay of evolution, negotiation, verification, and validation in early requirements analysis.

We believe that this work has achieved significant steps toward providing automated support for identifying, analyzing, and handling inconsistencies 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 analyzing NL requirements has opened up a new and fruitful area of research, for which this work is a starting point.

It is our hope that, as a result of our research, practitioners will have a better access to sophisticated tools and techniques for the management of inconsistencies in evolving requirements.

APPENDIX

A. BELIEF REVISION

A.1 Overview

The area of *belief revision* or *belief change* offers a theoretical foundation for rational changes of belief, focusing on revisions that occur when one receives new information that is possibly inconsistent with the present state of belief. The most well-known philosophical study of belief change in modern times is commonly referred to as the *AGM theory of epistemic change*. The AGM theory of belief revision is named after its originators, Alchourron, Gärdenfors, and Makinson, who developed the idea and published it jointly in 1985 [Alchourrón et al. 1985]. This framework was elaborated in Gärdenfors [1988] and since then it has become one of the standard frameworks for modeling belief and information change.

We do not intend to provide all the technical and philosophical details of the AGM paradigm here; for a comprehensive study of belief revision and AGM, the reader is referred to Gärdenfors [1988]. AGM theory adopts a simple way of modeling the *epistemic state* of a reasoner by a *set* of logical sentences closed under deduction, the intended meaning being that it contains precisely those sentences that she *believes to be true*. In order to incorporate new information which is inconsistent with the existing knowledge base, the reasoner must *decide* what information she is prepared to give up. Belief revision attempts to model rational decision making that concerns changes to a knowledge base.

Not every set of sentences is supposed to represent *rational* epistemic states. The AGM framework represents the body of information that may be rationally held by an individual as *belief sets*. The following *rationality criteria* determine what may be regarded as an idealized rational belief change process:

(1) All the *logical consequences* of beliefs in an epistemic state should be included in the epistemic state.
(2) Epistemic states should stay *consistent* if possible.
(3) When epistemic states are being revised, loss of information should be minimized.
(4) More important beliefs should be retained in favor of those that are less important.

The basic idea used in the formulation of the AGM framework is that when we, for some reason, change our beliefs, we would like to retain as much as possible of our old beliefs in the new belief state. This is known as the *principle of minimal change* [Gärdenfors 1988]. Beliefs are generally considered to be valuable (useful in arguments, expensive to acquire or infer), so unnecessary loss of beliefs is irrational. When we receive new information which is consistent with our current belief state, this requirement does not cause any problems, since we can then preserve all the old beliefs and add the new one. If we are rational, we must also accept all the logical consequences of the new belief. On the contrary, if the new information is inconsistent with the current state of belief, then some of the old beliefs must be retracted in order to preserve the consistency of the belief state.

In the AGM framework, information states are regarded as theories, and changes to the information content of an information state are taken as transformations on theories. The AGM framework provides three types of operations on belief states or transformations on theories. For each belief state K and proposition α, we may have one of the following three belief change operations:

—*Expansion.* Expanding K by α, written as K^+_α, means to add α to K without retraction of any existing beliefs and close under logical entailment. This may produce an inconsistent belief state. More formally, the expansion of a theory K with respect to a sentence α is defined as: $K^+_\alpha = Cn(K \cup \{\alpha\})$, where $Cn(K) = \{\alpha \mid K \models \alpha\}$.

Formally, it is assumed that $^+ : \mathit{Belief} \times \mathit{Ass} \rightarrow \mathit{Belief}$ is any function that, for any K and H belief sets, and for any assertion α, obeys the following postulates:

($^+1$) K^+_α is a belief set;
($^+2$) $\alpha \in K^+_\alpha$ (*the expanded set always includes* α);
($^+3$) $K \subseteq K^+_\alpha$ (*nothing is lost with expansion*);
($^+4$) $\alpha \in K \implies K^+_\alpha = K$ (*if* α *was already in* K*, expansion does nothing*);
($^+5$) $K \subseteq H \implies H^+_\alpha \subseteq K^+_\alpha$ (*monotonicity of expansion with respect to* $\subseteq$).

It can be shown that the only function satisfying those properties is in fact the logical closure under entailment of $K \cup \{\alpha\}$.

—*Revision.* Revising K with respect to α, written as K^*_α, means adding α to K with the possible removal of existing beliefs such that the result is a consistent belief state. Formally, a *revision function* is any function which satisfies the AGM revision postulates below:

(^{*1})	K^*_α is a belief set;	
(^{*2})	$\alpha \in K^*_\alpha$	(*the revised set always contains* α);
(^{*3})	$K^*_\alpha \subseteq K^+_\alpha$	(*nothing is added that would not be added by expansion*);
(^{*4})	$\neg\alpha \notin K \implies K^+_\alpha \subseteq K^*_\alpha$	(*if* α *does not cause a contradiction, revision is no weaker than expansion*);
(^{*5})	$K^*_\alpha = \bot \leftrightarrow \models \neg\alpha$	(K^*_α *becomes inconsistent only when trying to revise by a tautological falsity*);
(^{*6})	$\models \alpha \leftrightarrow \beta \implies K^*_\alpha = K^*_\beta$	(*revision does not depend on term syntax; logically equivalent terms produce the same revised set distributive property with respect to* $\wedge$);
(^{*7})	$K^*_{\alpha\wedge\beta} \subseteq (K^*_\alpha)^+_\beta$	
(^{*8})	$\neg\beta \notin K^*_\alpha \implies (K^*_\alpha)^+_\beta \subseteq K^*_{\alpha\wedge\beta}$	(*if no contradiction arises, the distributive property holds by equality*).

Postulates (^{*1})–(^{*8}) do not uniquely characterize a specific function, thus leaving room for different revision policies to be applied.

—*Contraction.* A contraction of K with respect to α, written as K^-_α, involves the removal of a set of sentences from K such that α is no longer implied. Formally, a *contraction function* is any function which satisfies the AGM contraction postulates listed below:

($^-1$)	K^-_α is a belief set;	
($^-2$)	$K^-_\alpha \subseteq K$	(*contraction does not add any belief to* K);
($^-3$)	$\alpha \notin K \implies K^-_\alpha = K$	(*if* α *was not in* K*, contracting it has no effect*;
($^-4$)	$\not\models \alpha \implies \alpha \notin K^-_\alpha$	*unless* α *is tautologically true, the contracted set will not contain* α).
($^-5$)	$\alpha \in K \implies K \subseteq (K^-_\alpha)^+_\alpha$	(*if* α *is in* K*, we do not lose information if we first contract and then expand* K *by it*);
($^-6$)	$\models \alpha \leftrightarrow \beta \implies K^-_\alpha = K^-_\beta$	(*contraction does not depend on term syntax; logically equivalent terms produce the same revised set distributive property with respect to* $\wedge$);
($^-7$)	$K^-_\alpha \cap K^-_\beta \subseteq K^-_{\alpha\wedge\beta}$	
($^-8$)	$\alpha \notin K^-_{\alpha\wedge\beta} \implies K^-_{\alpha\wedge\beta} \subseteq K^-_\alpha$	(*if* α *is not in* K *after contracting by* $\alpha \wedge \beta$*, the distributive property holds by equality*).

Rationality postulates basically specify constraints that the respective operators should satisfy. In essence, these postulates are motivated by the rationality criteria outlined above. They require that the outcome of a belief change

operation be a logically closed consistent theory, that the change operation be successful (i.e., in the case of expansion and revision, the new information should be consequences of the resulting theory and in contraction should not) that the outcome be independent of the syntactic form of the input, and that the operation involve *minimal change*.

One of the major result of the AGM framework is that the postulates for contraction are complementary to the revision postulates if the revision K^*_α is defined by using the *Levi Identity* [Levi 1977]: $K^*_\alpha = (K^-_{\neg\alpha})^+_\alpha$, which means that revision by α is equivalent to contracting by $\neg\alpha$ in order to remove any inconsistent beliefs and then expanding the result with α. Moreover, contraction can be defined in terms of revision using the *Harper Identity* [Harper 1977]: $K^-_\alpha = K \cap K^*_{\neg\alpha}$.

A.2 Epistemic Entrenchment

There is a special class of ordering (i.e., a transitive and reflexive relation) called *Epistemic Entrenchment* (EE) that is defined over the entire logical language. An EE ordering, which will differ from belief state to belief state, models the relative epistemic importance of the sentences in the belief set. The notion of EE is essentially motivated by the fourth rationality criterion described above. Gärdenfors [1988] stated that the epistemic entrenchment of a sentence in a belief state is determined by how useful it is in deliberation and inquiry. Some pieces of knowledge and belief about the world are of more importance than others when one is making decisions or is reasoning and planning. The EE of a sentence in a belief set is therefore related to its explanatory power and its overall informational value rather than to its probability. Furthermore, the context for which beliefs are ordered is very important because the same set of beliefs may have different EE ordering when used in different contexts.

The effect of this ordering of sentences over belief revision is the requirement that a revision/contraction operation should preserve more entrenched beliefs in preference to less entrenched ones. If x and y are sentences of a belief set K, $x \leq y$ means that y is at least as entrenched as x. The strict part of this order is represented as $x < y$ to mean that y is more entrenched than x. What this means informally is that, for any two formulas x and y such that $x < y$, whenever we have a choice between giving up x or giving up y (in order to preserve consistency), the former is chosen in order to minimize the epistemic loss. This procedure conveys all the required information to uniquely determine the result of a rational revision.

The major result of the epistemic entrenchment theory is that EE is fundamentally equivalent to the previously postulated notion of belief revision of theories. That is, rational contraction functions may be constructed from orderings of epistemic entrenchment, and the EE ordering may be constructed from rational contraction functions. One other result that is also important is that, when revising or contracting a theory K with respect to a conjunction $x \wedge y$, one must give up the conjunct that is less epistemically entrenched, or if both have the same EE, then both should be given up.

A.3 Belief Bases

While the AGM framework provides a useful abstraction for the belief change process, it does not lend itself to implementation in a straightforward way. It has been argued [Makinson 1985] that, when belief sets are revised or contracted, what is being applied is applied not to the belief set but rather to a finite, manageable, and reasonably close to irredundant *base* of the belief set. Several studies (e.g., Nebel [1991]) have therefore focused on *belief bases*, which are finite sets of sentences, instead of infinite deductively closed theories, as representations of belief states. The belief base approach considers priority relations (known as *epistemic relevance*) on the belief base, instead of entrenchment relations defined on the entire language, in determining the outcome of a belief change step. Formally, this notion can be modeled by representing an arbitrary set B of sentences in the language $\mathcal{L}$ that we call a *belief base*. Hence B is a base for a belief set K iff $Cn(B) = K$. In place of revision and contraction functions that are abstractly defined on belief sets, these functions are defined for belief bases assuming that belief sets are related to these belief bases [Gärdenfors and Rott 1995]. These new functions are appropriately called *base revision* and *base contraction*. An immediate result of employing belief bases is that one may end up having two (or more) different bases B_1 and B_2 such that $Cn(B_1) = Cn(B_2)$ but where revisions or contractions performed on these bases may result in different new states. What this means is that, by using belief bases instead of deductively closed belief sets, the belief changes will become syntax-dependent. It has been shown that inclusion of an explicit representation of disbelief in the epistemic state can solve this problem [Ghose 1995]. Another limitation of the belief base approach is the fact that operations performed on belief bases are not reversible. That is, beliefs that are discarded from the base are irretrievably lost and cannot be considered in future revision steps.

Representation schemes that are computationally viable are of special interest in RE. Moreover, requirements for a computer-based system are typically a finite collection of facts from a finite application domain. Therefore, in this article, requirements specifications are represented as belief bases and hence only the AGM-rational operators (i.e., operators which satisfy the relevant AGM postulates) for belief bases are considered.

B. FIELD TESTS OF CONJECTURES 4.7 AND 4.9

Conjectures 4.7 and 4.9, concerning the fidelity of the translation procedure between controlled natural language and predicate logic, cannot be ultimately *proved*, since a proof would require direct access to a person's internal state of conscience. In other words, we would like to verify the correspondence between a stakeholder's *intentions*, the natural language *expression* of those intentions, and the predicate logic *encoding* of the same for the purpose of formal reasoning. Given the inaccessibility of intentions, no direct proof can be carried out.

However, it is possible to organize blind tests to measure some attributes of the translation procedure. In particular, we have conducted experiments on two related attributes, with the goal of ascertaining

(1) how close the automatic translation is to what people knowledgeable in predicate logic would do manually if asked to perform the same task, and
(2) whether inferences performed through formal reasoning on the results of the translation are close to what experienced people would infer from the corresponding natural language sentences.

We emphasize again that the results of these experiments should not be taken as providing a definitive proof to our conjectures, as none can be provided. While our results lend further credibility to the conjectures, they firmly remain such.

B.1 Experimental Setup

A sample of 15 people was randomly selected from a population of academics and professional software developers, all of whom had a scientific background (having at least 1 year of professional experience after having obtained either a Master of Science or a Doctorate in Computer Science). The subjects were asked to answer the questionnaire shown below in Section B.4, which was administered in electronic form via e-mail. No time limit was placed on the subjects, and they checked the questionnaire at their most convenient time. Most of them reported that they had spent from 5 to 30 minutes on it. No information on the purpose of the test was provided, nor were they aware of the contents and background of the present study. No reward or other motivating factor was promised, nor provided, to respondents.

Responses were collected by email, and compared with those produced by CARL. In the first test (pairing test), propositional logic formulae 7 and 8 were equivalent. This was intentional, as we wanted to test if there was any preference among the respondents in associating them with the supposedly correspondent natural language sentences D and B, respectively.

B.2 Analysis of Results

The questionnaire included two sets of questions, one for each of the tasks we outlined above. The first set (*Pairing test*) was intended to verify whether the translation from natural language to logic performed by CARL according to Definitions 4.1–4.6 was consistent with the human respondents' expectations. The second test (*Inference test*) was intended to verify whether inferences performed by CARL using the logic translation and the usual inference rules of propositional logic were consistent with how the human respondents performed inference on the corresponding natural language sentences.

There was a certain variability in the answers we collected, due in part to differing interpretations placed on NL sentences and in part to plain errors on the part of the respondents (e.g., not recognizing that formulae 7 and 8 in the Pairing test have the same model, and are thus equivalent for semantic purposes). As a reference, we considered the *mode* of answers to each question—in other words, we assumed the most popular choice among respondents to be representative of what qualified humans in general would judge a "correct" translation into logic.

Table III. Summary of Results from the Field Test on Conjectures 4.7 and 4.9

	Pairing test									
Question	A	B	C	D	E	F	G	H	I	J
CARL response	5	7,8	12	7,8	3	9	—	20	13	14
Mode of respondents	5	8	12	7,8	10	9	—	20	18	14
Agreement with CARL	56%	100%	78%	100%	14%	100%	86%	44%	29%	100%
Agreement between respondents and CARL: 80% (mode), 72% (single)										

	Inference test							
Question	1	2	3	4	5	6	7	8
CARL response	Y	Y	N	Y	N	N	N	Y
Mode of respondents	Y	Y	N	Y	N	N	N	Y
Agreement with CARL	100%	57%	100%	100%	100%	71%	100%	57%
Agreement between respondents and CARL: 100% (mode), 83% (single)								

A summary of the results can be observed in Table III. As the table shows, in 8 out of 10 questions in the Pairing test CARL did as well as the majority of the respondents. Given that no better definition of what constitutes a correct translation can be given, this is a satisfactory result. The two exceptions are sentences E and I. In sentence E, the probable cause of the disagreement is the strategy used to translate selection of instances by adjectives. CARL collects all adjectives (Definition 4.4) and uses them as a premise for the whole clause (Definition 4.6). Thus, sentence E ("When an operator receives a phone call, he should dispatch a nearby available ambulance") is translated as "if there is an ambulance nearby, and the ambulance is available, then if an operator receives a phone call, he should dispatch that ambulance" (answer 3). On the contrary, the majority of the respondents favored answer 10, which could be read as "if an operator receives a phone call, then there is an ambulance nearby, it is available, and the operator should dispatch it." We believe that in this case the majority of the respondents were wrong, and that CARL's translation was more accurate.

The other exception, sentence I, can be attributed to a different meaning attached to the clause "only if." In fact, CARL interprets "p only if q" as equivalence, thus producing $p \leftrightarrow q$, while 57% of the respondents interpreted it as implication, thus indicating $q \rightarrow p$ as the correct translation (a formula that CARL produces for "p if q," "if q then p," etc.). We hypothesize that this result comes from the tendency most human beings have to reason deductively and only consider the positive case (hence a preference for $\rightarrow$), while devoting less attention to abductive reasoning and negative cases (hence ignoring the $\leftarrow$ side).

In the Inference test, there was total agreement between CARL and the mode of the respondents. In other words, in all cases CARL's judgement coincided with that of the majority of the respondents.

It should be noted that the degree of consensus among respondents in the first test was much lower that in the second one. We regard this as evidence that even experts have difficulties in doing this kind of translation manually, as there were many errors (in the sense of different minority opinions) among the responses for the Pairing test. Despite that, reasoning on the inference test was almost invariably correct. We interpret this phenomenon as a proof that

even if people indicated a "wrong" translation into logic for a given sentence, when reasoning about it they drew the same conclusions that could be drawn from a *different* (what we regard as "correct") translation.

Summarizing the results, we observed that CARL agreed with the majority of respondents in 80% of the cases for the Pairing test, and in 100% of the cases for the Inference test. If we do not consider majority, but compare each response on its own, we have that CARL's answer was considered the correct one in 72% of the cases in the Pairing test, and in 83% of the cases in the Inference test. It is worthwhile to stress that CARL's answers were the most agreed upon in all our data set—that is, no human respondent in our group was able to obtain a larger agreement on his choices than CARL. These results confirm our belief that Conjectures 4.7 and 4.9 do actually hold.

B.3 Threats to Validity

In the Pairing test, we provided a fixed set of possible translations to the respondents, whereas the real translation task was about writing a corresponding formula from scratch. Hence, we have not really tested the ability to translate a NL sentence into a propositional logic formula, but rather the ability to decide whether a proposed translation was correct or not. In this sense, what we tested was the decision problem associated with our original task. However, several almost correct translations of each sentence were provided, and respondents knew that some sentences could have no corresponding translation (e.g., sentence G). In these conditions, the decision problem was very similar to considering several plausible alternative translations in the direct translation task. We thus consider the results of our test applicable to Conjecture 4.7.

In the Inference test, we could have tested with entire sets of consequences, or with longer chains of inferences. Indeed, the consistency proofs that CARL produces internally are usually rather large. Instead, we chose to present the respondents with simple inferences, with only one or two derivation steps. This does not invalidate the test, since we wanted to investigate the basic *mechanisms* of inference on natural language sentences, with an hypothesis that rigorous reasoning on more complex inferences is obtained by composing short derivations. Notice that in general other psychological mechanisms could intervene in cases where complex inferences were needed: for example: *intuition* (unrationalized qualitative reasoning). However, for the purpose of carrying out rigorous consistency proofs, we wanted to restrict ourselves to rational reasoning, and this is what was tested in our experiment.

Finally, we have no guarantee that our sample is representative of the general population of professional requirements engineers. In fact, including members from academia, it had probably a bias toward a more formally oriented population than that normally found in industry. However, this does not change the general conclusion: instead, it makes our point even stronger, in that members of a less formally oriented population would have more difficulty in accomplishing the translation and reasoning task, thus making CARL even more useful in such a context.

In conclusion, even if the small size of our sample does not allow us to draw an authoritative statistical conclusion about our conjectures, all the evidence we collected is in support of them.

B.4 Questionnaire Provided

Below we include *verbatim* the questionnaire that was provided to the respondents.

Pairing test. Some of the following sentences (A–J) are equivalent to one or more of the propositional logic formulae below (1–20); other sentences may not have a corresponding logic formula. Please write down the pairs that you believe to be correspondent (e.g., A-3, B-6, B-8, C-none, . . .).

Natural language sentences

A. A medical emergency is either an illness or an accident.
B. If an ambulance does not have a crew, then it is not available.
C. The operator receives a phone call concerning an emergency.
D. If an ambulance is available, then it has a crew.
E. When an operator receives a phone call, he should dispatch a nearby available ambulance.
F. The operator does not dispatch an ambulance if the ambulance is not nearby and available.
G. If the phone call concerns an emergency, the operator should dispatch a nearby ambulance.
H. If the phone call concerns a medical emergency, the operator should dispatch an ambulance.
I. An ambulance is available only if it is nearby.
J. If the operator dispatches an ambulance, then it has a crew.

Propositional logic formulae

(1) emergency(medical) $\rightarrow$ illness(medical) $\vee$ accident(medical)
(2) dispatch(operator,ambulance) $\leftrightarrow$ have(ambulance,crew)
(3) (nearby(ambulance) $\wedge$ available(ambulance)) $\rightarrow$ (receive(operator,phone_call) $\rightarrow$ dispatch(operator,ambulance))
(4) medical(emergency) $\rightarrow$ (receive(operator,phone_call) $\rightarrow$ dispatch(operator,ambulance))
(5) medical(emergency) $\rightarrow$ (illness(emergency) $\leftrightarrow \neg$accident(emergency))
(6) concern(phone_call,emergency) $\rightarrow$ receive(operator,phone_call)
(7) available(ambulance) $\rightarrow$ have(ambulance,crew)
(8) $\neg$have(ambulance,crew) $\rightarrow \neg$available(ambulance)
(9) $\neg$(nearby(ambulance) $\wedge$ available(ambulance)) $\rightarrow \neg$dispatch(operator,ambulance)
(10) receive(operator,phone_call) $\rightarrow$ (nearby(ambulance) $\wedge$ available(ambulance) $\wedge$ dispatch(operator,ambulance))
(11) $\neg$nearby(ambulance) $\wedge$ available(ambulance) $\rightarrow$ dispatch(operator,ambulance)
(12) receive(operator,phone_call) $\wedge$ concern(phone_call,emergency)

(13) available(ambulance) ↔ nearby(ambulance)

(14) dispatch(operator,ambulance) → have(ambulance,crew)

(15) receive(operator,phone_call) → concern(phone_call,emergency)

(16) receive_phone_call(operator) ∧ concern_phone_call(emergency)

(17) ¬nearby(ambulance) →(¬available(ambulance) →¬dispatch(operator,ambulance))

(18) nearby(ambulance) → available(ambulance)

(19) have(crew,ambulance) ∧ nearby(ambulance) → dispatch(operator,ambulance)

(20) medical(emergency) → (concern(phone_call,emergency) → dispatch(operator,ambulance))

Inference test. Judge whether the following inferences on natural language sentences are tenable or not (e.g., 1-Yes, 2-No, ...).

(1) (a) When an operator receives a phone call concerning a medical emergency, he should dispatch a nearby available ambulance. (b) The operator receives a phone call concerning an emergency. (c) The emergency is a medical emergency. **Thus,** (d) The operator should dispatch a nearby available ambulance.

(2) (a) A medical emergency is either an illness or an accident. (b) A given medical emergency is about an illness. **Thus,** (c) the medical emergency is not about an accident.

(3) (a) When an operator receives a phone call concerning a non-medical emergency, he should not dispatch an ambulance. (b) The operator receives a phone call concerning an attempted manslaughter. **Thus,** (c) the operator should dispatch an ambulance.

(4) (a) When an operator receives a phone call, he should dispatch a nearby available ambulance. (b) The operator receives a phone call concerning an attempted manslaughter. (c) An ambulance is nearby. (d) The ambulance is available. **Thus,** (e) the operator should dispatch the ambulance.

(5) (a) If an ambulance was not serviced in 2002, then it is not available. (b) An ambulance was serviced in 2002. **Thus,** (c) it is available.

(6) (a) An ambulance is available if it has a crew and it was serviced in 2002. (b) When an operator receives a phone call concerning a medical emergency, he should dispatch an available ambulance. (c) An ambulance has a crew. (d) The ambulance was serviced in 2002. **Thus,** (e) the operator should dispatch the ambulance.

(7) (a) If an ambulance is not available, a technician must check it for maintainance. (b) An ambulance is available. **Thus,** (c) the technician must not check it for maintanance.

(8) (a) An ambulance is available if it has a crew. (b) When an operator receives a phone call concerning a medical emergency, he should dispatch an available ambulance. (c) The operator receives a phone call concerning an emergency. (d) The emergency is a medical emergency. (e) An ambulance has a crew. **Thus**, (f) The operator should dispatch that ambulance.

REFERENCES

ABITEBOUL, S. AND SIMON, E. 1991. Fundamental properties of deterministic and nondeterministic extensions of Datalog. *Theoret. Comput. Sci. 78*, 1, 137–158.

ALCHOURRÓN, C. E., GÄRDENFORS, P., AND MAKINSON, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *J. Symbol. Logic 50*, 510–530.

ALI, S. 1994. A logical language for natural language processing. In *Proceedings of the 10th Biennial Canadian Artifical Intelligence Conference* (Banff, Alta., Canada). 187–196.

AMBRIOLA, V. AND GERVASI, V. 1999. Experiences with domain-based parsing of natural language requirements. In *Proceedings of the Fourth International Conference on Applications of Natural Language to Information Systems*, G. Fliedl and H. C. Mayr, Eds. OCG Schriftenreihe (Lecture Notes), no. 129. OGC, Klagenfurt, Austria, 145–148.

BALZER, R. 1991. Tolerating inconsistency. In *Proceedings of the 13th International Conference on Software Engineering (ICSE-13)*. IEEE Computer Society Press, Los Alamitos, CA, 158–165.

BERRY, D. M., KAMSTIES, E., AND KRIEGER, M. M. 2003. From contract drafting to software specification: Linguistic sources of ambiguity—a handbook. (unpublished as of July 2005, but available on-line at `http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf`).

BOEHM, B. W. 1976. Software engineering. *IEEE Trans. Comput. 25*, 12 (Dec.), 1226–1241.

CHAOS. 1995. Software development report by the Standish group. Available online at `http://www.standishgroup.com/chaos.html`.

CHECHIK, M., DEVEREUX, B., EASTERBROOK, S., AND GURFINKEL, A. 2003. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Methodol. 12*, 4 (Oct.), 371–408.

CHOI, Y., RAYADURGAM, S., AND HEIMDAHL, M. P. E. 2002. Toward automation for model-checking requirements specifications with numeric constraints. *Requirements Eng. J. 7*, 4 (Dec.), 225–242.

CLARKE, E. M., EMERSON, E. A., AND SISLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programm. Lang. Syst. 8*, 2, 244–263.

DALIANIS, H. 1992. A method for validating a conceptual model by natural language discourse generation. In *Advanced Information Systems Engineering*. Lecture Notes in Computer Science, vol. 593. Springer-Verlag, Berlin, Germany.

DALY, E. 1977. Management of software development. *IEEE Trans. Softw. Eng. 3*, 3 (May), 229–242.

DAVIS, A. M., JORDAN, K., AND NAKAJIMA, T. 1997. Elements underlying the specification of requirements. *Ann. Softw. Eng. 3*, 63–100. (Special issue on Software Requirements Engineering)

EASTERBROOK, S. AND CHECHIK, M. 2002. Guest editorial: Special issue on model checking in requirements engineering. *Requirements Eng. J. 7*, 4 (Dec.), 221–224.

EASTERBROOK, S. AND NUSEIBEH, B. 1996. Using viewpoints for inconsistency management. *IEE Softw. Eng. J. 11*, 1, 31–43.

ESHUIS, R., JANSEN, D. N., AND WIERINGA, R. 2002. Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Eng. J. 7*, 4 (Dec.), 243–263.

FANTECHI, A., GNESI, S., RISTORI, G., CARENINI, M., VANOCCHI, M., AND MORESCHINI, P. 1994. Assisting requirement formalization by means of natural language translation. *Form. Meth. Syst. Des. 4*, 3, 243–263.

FINKELSTEIN, A. AND DOWELL, J. 1996. A comedy of errors: The London Ambulance Service case study. In *Proceedings of the 8th International Workshop on Software Specification & Design*. IEEE Computer Society Press, Los Alamitos, CA, 2–4.

FINKELSTEIN, D., GABBAY, A., HUNTER, J., KRAMER, B., AND NUSEIBEH, A. 1994. Inconsistency handling in multiperspective specifications. *IEEE Trans. Softw. Eng. 20*, 8, 569–577.

FUCHS, N. E. AND SCHWITTER, R. 1995. Specifying logic programs in controlled natural language. In *Proceedings of the Workshop on Computational Logic for Natural Language Processing* (University of Edinburgh, Edinburgh, Scotland).

GABBAY, D. AND HUNTER, A. 1991. Making inconsistency respectable: A logical framework for inconsistency in reasoning, Part 1—a position paper. In *Proceedings of Fundamentals AI Research*. Lecture Notes in Computer Science, vol. 535. Springer-Verlag, New York, 19–32.

GARCEZ, A., RUSSO, A., NUSEIBEH, B., AND KRAMER, J. 2003. Combining abductive reasoning and inductive learning to evolve requirements specifications. *IEE Proc.—Softw. 150*, 1 (Feb.), 25–38.

GÄRDENFORS, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA.

GÄRDENFORS, P. AND ROTT, H. 1995. Belief revision. In *Handbook of Logic in Artificial Intelligence and Logic Programming Volume IV: Epistemic and Temporal Reasoning*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Oxford University Press, Oxford, U.K. 35–132.

GERVASI, V. 2000. Environment support for requirements writing and analysis. Ph.D. dissertation. University of Pisa, Pisa, Italy.

GERVASI, V. 2001. The Cico domain-based parser. Tech. rep. TR-01-25. Dipartimento di Informatica, University of Pisa, Pisa, Italy.

GERVASI, V. AND AMBRIOLA, V. 2002. Quantitative assessment of textual complexity. In *Complexity in Language and Text*, L. Merlini Barbaresi, Ed. PLUS-University of Pisa, Pisa, Italy, 197–228.

GERVASI, V. AND NUSEIBEH, B. 2002. Lightweight validation of natural language requirements. *Softw.: Pract. Exper. 32*, 2 (Feb.), 113–133.

GHEZZI, C. AND NUSEIBEH, B. 1998. Introduction to the special section on managing inconsistency in software development. *IEEE Trans. Softw. Eng. 24*, 11 (Nov.), 906–907.

GHOSE, A. K. 1995. Practical belief change. Ph.D. dissertation. Department of Computing Science, University of Alberta, Edmonton, Alta., Canada.

HARPER, W. L. 1977. Rational conceptual change. In *PSA 1976 East Lansing, Mich.: Philosophy of Science Association*, vol. 2. Philosophy of Science Association, East Lansing, MI, 462–494.

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

HUNTER, A. AND NUSEIBEH, B. 1998. Managing inconsistent specifications: Reasoning, analysis and action. *Trans. Softw. Eng. Method. 7*, 4 (Oct.), 335–367.

HUTH, M. AND PRADHAN, S. 2004. Consistent partial model checking. *Electron. Notes Theoret. Comput. Sci. 73*, 45–85. (Proceedings of the Workshop on Domains VI.)

IBANEZ, M. 1996. European user survey analysis. Tech. rep. ESI report TR95104. European Software Institute, Zamudio, Spain. Web site: `www.esi.es`.

JACKSON, D. 2000. Automating first-order relational logic. In *Proceedings of the 8th International Symposium on Foundations of Software Engineering* (FSE8).

JURISTO, N., MORENO, A. M., AND LÓPEZ, M. 2000. How to use linguistic instruments for object-oriented analysis. *IEEE Softw. 17*, 3 (May/June), 80–89.

KOWALSKI, R. 1979. *Logic for Problem Solving*. North Holland Elsevier, New York, NY.

LEVI, I. 1977. Subjunctives, dispositions and chances. *Synthese 34*, London, U.K., 423–455.

MACIAS, B. AND PULMAN, S. G. 1993. *Natural Language Processing for Requirements Specification*. Chapman and Hall, London, U.K., 57–89.

MAKINSON, D. 1985. How to give it up: A survey of some formal aspects of the logic of theory change. *Synthese 62*, 4, 347–363.

MARCUS, M. P., SANTORINI, B., AND MARCINKIEWICZ, M. A. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computa. Linguist. 19*, 313–330.

MICH, L. 1996. NL-OOPS: From natural language requirements to object oriented requirements using the natural language processing system LOLITA. *J. Natural Lang. Eng. 2*, 2, 161–187.

MICH, L., FRANCH, M., AND NOVI INVERARDI, P. 2004. Market research for requirements analysis using linguistic tools. *Requirements Eng. J. 9*, 1, 40–56.

NEBEL, B. 1991. Belief revision and default reasoning: Syntax-based approaches. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*. 417–428.

NEILL, C. J. AND LAPLANTE, P. A. 2003. Requirements engineering: The state of the practice. *IEEE Softw. 20*, 6 (Nov./Dec.), 40–45.

NUSEIBEH, B. 1996. To be and not to be: On managing inconsistency in software development. In *Proceedings of the IEEE International Workshop on Software Specification and Design* (IWSSD-8). IEEE Computer Society Press, Los Alamitos, CA, 164–169.

POOLE, D. 1988. A logical framework for default reasoning. *Art. Intell. 36*, 27–47.

POOLE, D., GOEBEL, R., AND ALELIUNAS, R. 1987. Theorist: A logical reasoning system for defaults and diagnosis. In *The Knowledge Frontier: Essays in the Representation of Knowledge*, N. Cercone and G. McCalla, Eds. Springer Verlag, Berlin, Germany, 331–352.

REEVES, S. AND CLARKE, M. 1990. *Logic for Computer Science*. Addison-Wesley, Reading, MA.

ROLLAND, C. AND PROIX, C. 1992. A natural language approach for requirements engineering. In *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, P. Loucopoulos, Ed. Lecture Notes in Computer Science, vol. 593. Springer-Verlag, Berlin, Germany, 257–277.

RUSSO, A., MILLER, R., NUSEIBEH, B., AND KRAMER, J. 2002. An abductive approach for analysing event-based requirements specifications. In *Proceedings of the 18th International Conference on Logic Programming* (Copenhagen, Denmark).

RUSSO, A. AND NUSEIBEH, B. 2001. On the use of logical abduction in software engineering. In *Handbook of Software Engineering and Knowledge Engineering*, S. K. Chang, Ed. World Scientific Publishing, Singapore.

RYAN, M. D. 1993. Defaults in specifications. In *Proceedings of the IEEE International Symposium on Requirements Engineering* (RE93). 142–149.

SADRI, F. AND KOWALSKI, R. 1986. An application of general theorem proving to database integrity. Tech. rep. Department of Computing, Imperial College, London, U.K.

SCHMID, H. 1994. Probabilistic part–of–speech tagging using decision trees. In *Proceedings of the Conference on New Methods in Language Processing* (Manchester, U.K.). 44–49.

TSAI, J. P., WEIGERT, T., AND JANG, H. C. 1992. A hybrid knowledge representation as a basis of requirements specification and specification analysis. *IEEE Trans. Softw. Eng. 18*, 12, 1076–1100.

ULLMAN, J. D. 1989. *Principles of Database and Knowledge-Base Systems*, Vols. I and II. Computer Science Press, Rockville, MD.

VAN LAMSWEERDE, A. 2000. Formal specifications: A roadmap. In *The Future of Software Engineering*. ACM Press, New York, NY, 149–159. (Proceedings of the 22nd IEEE International Conference on Software Engineering.)

WEBBER, B. L. 1983. So what can we talk about now. In *Computational Models of Discourse*, M. Brady and B. Berwick, Eds. MIT Press, Cambridge, MA, 331–370.

ZAVE, P. AND JACKSON, M. 1997. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Method. 6*, 1, 1–30.

ZOWGHI, D. 1999. A logic-based framework for the management of changing software requirements. Ph.D. dissertation. Macquarie University, Sydney, Australia.

ZOWGHI, D. AND GERVASI, V. 2004. On the interplay between consistency, completeness, and correctness in requirements evolution. *Informat. Softw. Techn. 46*, 11 (Sept.), 763–779.

ZOWGHI, D., GERVASI, V., AND MCRAE, A. 2001. Using default reasoning to discover inconsistencies in natural language requirements. In *Proceedings of the 8th Asia-Pacific Software Engineering Conference*. 133–140.

ZOWGHI, D., GHOSE, A., AND OFFEN, R. 1997. Computer-assisted requirements evolution tool: Formal foundations and architecture. In *Proceedings of the 2nd Australian Workshop on Requirements Engineering* (AWRE97, Sydney, Australia). 47–59.

ZOWGHI, D., GHOSE, A., AND PEPPAS, P. 1996. A framework for reasoning about requirements evolution. In *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence* (PRICAI'96). 157–168.

ZOWGHI, D. AND OFFEN, R. 1997. A logical framework for modelling and reasoning about the evolution of requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering* (RE97). 247–259.

Received May 2002; revised December 2003, January 2005; accepted February 2005