Agile Requirements Engineering via Paraconsistent Reasoning*

Neil A. Ernst[a], Alexander Borgida[b], Ivan J. Jureta[c], John Mylopoulos[d]

[a] *Department of Computer Science, University of British Columbia*
nernst@cs.ubc.ca
[b] *Department of Computer Science, Rutgers University*
borgida@cs.rutgers.edu
[c] *Fonds de la Recherche Scientifique – FNRS & Department of Business Administration, University of Namur*
ivan.jureta@fundp.ac.be
[d] *Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento*
jm@disi.unitn.it

Abstract

Innovative companies need an agile approach towards product and service requirements, to rapidly respond to and exploit changing conditions. The agile approach to requirements must nonetheless be systematic, especially with respect to accommodating legal and nonfunctional requirements. This paper examines how to support lightweight, agile requirements processes which can still be systematically modeled, analyzed and changed. We propose a framework, REKOMBINE, which is based on a propositional language for requirements modeling called Techne. We define operations on Techne models which tolerate the presence of inconsistencies. This paraconsistent reasoning is vital for supporting delayed commitment to particular design solutions. We evaluate these operations with an industry case study using two well-known formal analysis tools. Our evaluations show that the proposed framework scales to industry-sized requirements models, while still retaining (via propositional logic) the informality that is so useful during early requirements analysis.

Keywords: paraconsistency, agile methods, software requirements, requirements evolution

Contents

*This is an expanded and updated version of an earlier conference paper [1].

1. Introduction

It is increasingly uncommon for software systems to be fully specified before implementation begins. This is because uncertainty about the right requirements is inescapable. Furthermore, it is highly desirable to avoid premature commitment by being able to change/revise requirements throughout the development lifecycle. Being flexible in this fashion is a source of competitive advantage for a business; for example, by delivering the correct product before competitors. The notion that one should engage in what has been called "big design up front" as part of the design activity is no longer defensible [2], since inevitably early plans must be abandoned, or at best revised. A variety of studies and experience reports (most recently [3]) have shown that requirements changes are very expensive to accommodate and constitute the most frequent cause of project failures.

There is a shift, instead, to methods of software development which avoid premature commitment to decisions. The central tenet of these methods, including most Agile methodologies, is that requirements are discussed iteratively. These requirements are often manifested as very brief *user stories*, which serve as conversation starters with business representatives. A major concern with

such lightweight requirements "engineering" is that non-functional requirements, such as security, are often neglected since system functionality is the focus [4].

While this lightweight approach to Requirements Engineering (RE) has become popular in many segments of industry, the IEEE standard for software requirements [5] uses words like *"correct"* and *"unambiguous"* to describe its recommended practice for RE. Thus, many previous approaches to the problem of system specification have methodological constraints insisting that conflicts and obstacles be resolved before solutions are identified. It is rarely, if ever, possible to achieve these criteria. We argue that the above shift demands a much more flexible approach to requirements modeling and analysis.

In this paper we introduce a framework, RE-KOMBINE, which supports this shift to flexibility. The framework represents possibly contradictory requirements as assertions about the current state of the requirements model, allowing us to reason *paraconsistently* about requirements problems. In paraconsistent reasoning local inconsistencies are not propagated globally, "polluting" all inferences, as in standard logic. This paraconsistent reasoning accommodates flexible, agile decision-making. The chief advantage of our approach is that it permits derivation of useful knowledge about problems of interest in the moment, while postponing decisions about currently inconsistent states of the problem until a decision must be made.

Example 1. *To illustrate the usefulness of deferring conflict resolution, consider the requirements fragment in Fig. 1. The figure represents part of the business requirements ("use mobile terminals") together with the imposed requirements from an applicable security standard (PCI-DSS, which we discuss fully in Section 4.1). The red, X-headed relation between requirements* $\mathsf{t}_{\mathsf{wep}}$ *and* $\mathsf{d}_{4.1.1}$ *represents a conflict. In this case, the conflict is between the business-motivated use of the Wireless Encryption Protocol (WEP) and the security problems with WEP. Domain assumptions (in square boxes) are taken to be asserted truths (in contrast to goals, which are desired, or tasks, which are proposed parts of implementations). Existing approaches to requirements analysis either (i) insist that the conflict be resolved before proceeding with further reasoning (e.g., KAOS [6]), or (ii) represent the conflict as trade-offs for higher-level goals (e.g., [7]).*

Assuming we have an existing WEP solution in place, it also must satisfy the PCI-DSS. The conflict in this model emerges as a logical inconsistency. Because our framework is paraconsistent, its reasoning can isolate and work around the conflict for the time being, by informing the requirements engineer of conclusions that can be satisfied regardless. This is useful if, for example, the need for compliance is not immediate, and time and money can be better spent elsewhere. We also flag the conflict for the requirements engineer, who later has the choice of proceeding with one of the conflicting choices, or insisting that the model be revised (e.g., remove our WEP equipment). In the example, the final system might instead make use of Bluetooth for wireless transmission. ■

In practice, then, the approach for a requirements engineer is as follows (italicized text indicates steps not addressed in this paper).

1. *Elicit requirements from stakeholders*

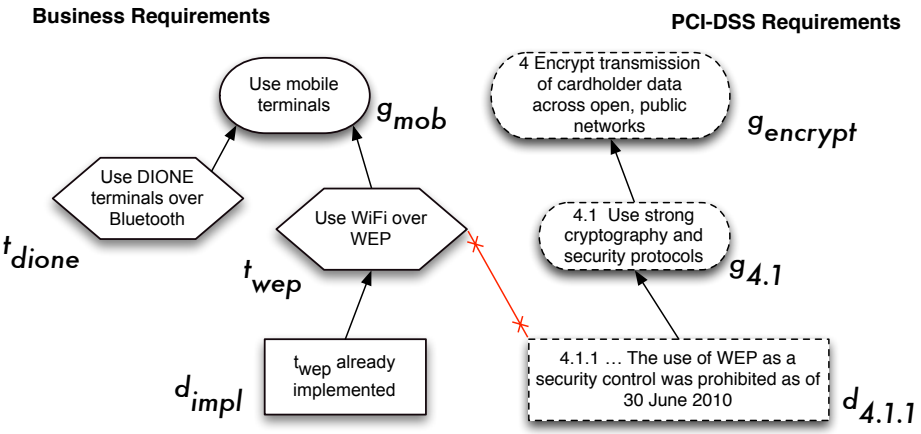

Figure 1: Fragment of the requirements model from the payment card case. (Section 4.1)

2. Create requirements model using RE-KOMBINE, written in the T1 language. The model will include goals, domain assumptions, existing and possible tasks, and the structure of the problem space.
3. Query RE-KOMBINE for possible solutions, if any, to this model.
   (a) If RE-KOMBINE indicates a solution exists, understand whether this is a controversial or uncontroversial solution (i.e., whether paraconsistent reasoning was necessary). If needed, choose one of the options in the conflict through further interaction with the stakeholders. If it is necessary to remove the conflict after choosing one option, then remove one or all other options in the conflict, which will remove the conflict. The decision to do so is indicated by assessment of the cost of revision, using, e.g., real options theory [8].
   (b) *If no solutions exists, repeat elicitation.*
4. *Implement specification indicated by answer to queries on RE-KOMBINE.*
5. Update the requirements model when and where the existing problem changes, e.g., if new requirements are discovered or new domain assumptions constrain the solutions.

Here is an example showing how this might work.

Example 2. *Lucene*[1] *is a popular text retrieval library. One recent feature, tracked as LUCENE-1458*[2]*, is the addition of a flexible indexing approach to storing text. The requirements might be represented as shown in Figure 2.*

Note the conflict between the implemented text-storage approach t_{utf8}, supporting g_{bytes}, and the need for t_{utf16} for Unicode support g_{uni}, which says that if we want both flexible indexing (g_{flex}) and need internationalization support

[1] `http://lucene.apache.org`
[2] `https://issues.apache.org/jira/browse/LUCENE-1458`

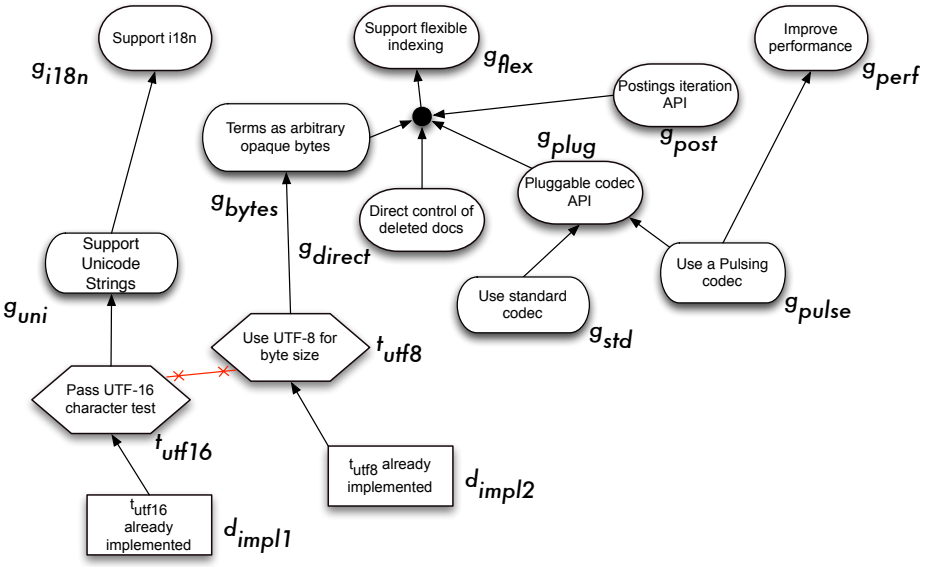

Figure 2: The hypothetical Lucene scenario, showing conflict between storage format and internationalization.

(g_{i18n}) we have an inconsistent model. At this point we would like to signal the requirements engineer of this inconsistency, leaving it to him/her to resolve the problem. This might mean changing the design, dropping support for internationalization, or deferring the decision until the development of the rest of the feature was completed (e.g., satisfying g_{post}). ■

What actually happened in this scenario was that the need for internationalization (a non-functional requirement) was never explicitly connected to the change in storage format for the new feature, and only a few days prior to release was the conflict understood. This example shows the inter-twining of the two motivations for our work: 1) the importance of requirements, because the portability/internationalization non-functional requirement did not seem to be an important factor in the ad-hoc requirements approach of Lucene's developers; 2) the usefulness of delayed commitment in resolving fundamental contradictions in design approaches, such as the Unicode issue. For Lucene, in particular, the ability to develop experimentally is important, and we would not want to insist on Unicode support until the rest of the feature's design was properly understood.

In [9], we introduced the notion of a Requirements Engineering Knowledge Base/Theory RETH for maintaining a requirements model. In this paper, we build on the notion of an RETH to focus on the case where problems are changing and possibly inconsistent (i.e., contain contradictory assertions). We adopt a design science research methodology (as explained in Wieringa et al. [10]), focusing on the problem investigation and solution validation aspects, and leaving evaluation

of the solution to future work. Our research questions are

RQ1 What formal mechanisms are necessary to support reasoning with inconsistent requirements models?

RQ2 Is there a scalable algorithm for automating analyses of such models?

To answer these questions, this paper makes the following contributions:

- motivates the problem by identifying the importance of accommodating variability by supporting paraconsistency in software development, and modifying the way in which the requirements are queried;
- proposes a framework, RE-KOMBINE, for finding solution specifications to possibly inconsistent, goal-oriented requirements problems, including:
  - language T1, in Section 3.1;
  - consequence relation over T1 formulas $\mid\!\sim$;
  - formally specifying operators querying/modifying RETH using T1 and $\mid\!\sim$, in Section 3.3;
  - implementing these operators on top of existing technologies;
- explicitly introduces paraconsistent reasoning into a prototype tool (Section 3.4);
- evaluates the framework and its scalability with a retrospective industrial case study (Section 4).

This paper is an updated and expanded version of an earlier conference paper [1]. The most important extensions over [1] are: (1) a detailed discussion of paraconsistent consequence ($\mid\!\sim$) to explain several semantic choices; (2) a formal introduction of the Techne language T1 used in the earlier paper;(3) a formal definition of the candidate solution concept; (4) a greatly expanded set of paraconsistent operators to include those creating RETH and those finding conflicts.

Notation—We use the following conventions in our paper, indexed or primed as necessary.

- Capital Roman letters ($A, B, C, \ldots$) represent typed sets;
- Capital Greek letters (Π, Δ, Θ) represent arbitrary sets of formulas;
- Small Greek letters ($\phi, \psi, \delta, \ldots$) represent arbitrary formulas;
- Small Roman letters represent specific atoms (e.g., a_1, a_2, b_3);
- $\perp$ reads False.

2. Paraconsistency and Agile Requirements Evolution

2.1. Inconsistency and Conflict

The ability to represent conflicts between requirements is an essential part of any requirements modeling language. In formal logic, a theory $\mathcal{T}$ is said to be *inconsistent* if one can derive $False/\bot$, from $\mathcal{T}$. Classical logic trivializes in the sense that anything can be derived from an inconsistent $\mathcal{T}$ (*ex falso quodlibet*). This makes inconsistent RETH based purely on classical logic useless for solving requirements problems

There are several ways to interpret the presence of a *conflict* relation between requirements ϕ and ψ. The conflict might mean that neither requirement can be satisfied. Logically this might be recorded as $\phi \rightarrow \bot$ and $\psi \rightarrow \bot$. The conflict could also mean that at most one, but not both can be satisfied, recorded as $\phi \wedge \psi \rightarrow \bot$. Finally, it could be more drastic, and suggest that the entire model must be resolved to remove the conflict. Part of searching for solutions to requirements problems is to find ways to ensure at most one of ϕ or ψ, where ϕ and ψ are in conflict, is satisfied.

In the Requirements Engineering (RE) research community, the term "conflict" has typically been used to denote social disagreement over the nature of the system requirements. Robinson et al. [11] define it as "requirements held by two or more stakeholders that cause an inconsistency". The term "inconsistency" denotes the technical, formal existence of a "broken rule" [12]. Zowghi and Gervasi [13] show that "consistency" is directly related to requirements "completeness": a more complete requirements document is often less consistent (since more competing requirements are introduced).

In this paper, the conflict relation is formally between two or more requirements, and not between stakeholders. Any conflicts between stakeholders, such as a disagreement over terminology, are the purview of other techniques (e.g., negotiation and model merging). A potential conflict is the presence in a requirements engineering knowledge base RETH of rules of the form $\phi \wedge \psi \rightarrow \bot$, while inconsistency is the assertion/choice of literals/atoms that force RETH to classically entail $\bot$. Ultimately, we not only want to permit conflict (and possibly inconsistency) to be present; we also want to specify what we ought to do when inconsistency is detected.

Classical propositional logic and first-order logic cannot tolerate inconsistency, in the sense that no useful reasoning can be done in its presence; yet, in the RE domain, tolerating inconsistency is important. Nuseibeh *et al* [14] give a few important reasons:

1. to facilitate distributed collaborative working;
2. to prevent premature commitment to design decisions;
3. to ensure all stakeholder views are taken into account;
4. to focus attention on problem areas [of the specification].

Perhaps the most useful reason for the case of evolving requirements problems is the second one. Avoiding premature commitment, in the sense of Thimbleby

[15], means to wait until the "last responsible moment" to make decisions regarding the system. Not only does this apply to deciding *how* to satisfy our goals, but also in the choice of those goals themselves. Tolerating inconsistency therefore allows us to continue to make progress on design (and even implementation) while fire-walling the conflicting parts of the system. Section 4.1 will show how this becomes crucial in our case study.

In our case, part of tolerating inconsistency in the RETH involves *paraconsistent reasoning*. A paraconsistent logic, broadly, is one which does not trivialize in the presence of inconsistency. Section 3.3 will show how we define operators on the knowledge base that continue to give meaningful answers even when inconsistency is present.

We start from Zave & Jackson's *"solution to the requirements problem"* framework [16]: **given** requirements R, and domain assumptions D, **find** specification S, satisfying $D \cup S \vdash R$ under the condition that $D \cup S$ is consistent. This is the classical logical consequence relation, $\vdash$. Implicit in this formulation is that one can distinguish formulas/atoms representing requirements from ones representing specifications, and that D contains formulas that somehow connect these.

We modify this to deal with goal-oriented specifications (of the kind illustrated in Figures 1 and 2), and add some details as follows: we are **given** disjoint sets of (1) goals G, (2) tasks T, and (3) domain assumptions D, together with a set R of (4a) refinements that indicate how goals and possibly tasks may be achieved, and (4b) conflicts between elements of $T \cup G \cup D$; plus a subset *desired*(G) of desired goals; we must **find** specifications S that are minimal subsets of T such that

$$D, R, S \vdash \textit{desired}(G) \tag{1}$$

and $D \cup R \cup S$ is classically consistent.

Example 3. ***(example 1 revisited.)*** *We can review the difficulty with consistency in Example 1, by formalizing the requirements problem:* $R = \{t_{dione} \rightarrow g_{mob}, t_{wep} \rightarrow g_{mob}, d_{impl} \rightarrow t_{wep}, d_{4.1.1} \rightarrow g_{4.1}, g_{4.1} \rightarrow g_{encrypt}, t_{wep} \wedge d_{4.1.1} \rightarrow \bot\}$, $D = \{d_{4.1.1}, d_{impl}\}$. *Note that* d_{impl} *appears in* D*, reflecting the fact that this business had already implemented a WEP wifi solution.*

If one now desires to find a minimal subset S *of the task specifications* $T = \{t_{wep}, t_{dione}\}$ *available in* R *which achieves goals* $\textit{desired}(G) = \{g_{mob}, g_{encrypt}\}$*, there is no classical solution to* $D \cup R \cup S \vdash \textit{desired}(G)$*, in the style of Zave and Jackson, since* $D \cup R$ *is inconsistent.*

To deal with possible inconsistencies, we replace $\vdash$ *by* $\mid\!\sim$*, which is a special paraconsistent consequence relation*[3]*. The remainder of this section is dedicated to exploring what is 'passed through'* $\mid\!\sim$ *in order to reason appropriately on inconsistent requirements problems.*

For example, if we use a $\mid\!\sim$ *that looks for maximal subsets of its theory then solution* $S_2 = \{t_{dione}\}$ *can satisfy* $D \cup R \cup S \mid\!\sim G$*, because* d_{impl} *is omitted from*

[3]Not all paraconsistent logics will do!

D. In this example, the new solution specification says "implementing Bluetooth avoids the conflict with the PCI-DSS of the original approach".

In contrast, note that if one wanted only to find a solution for the less ambitious set of goals $G' = \{g_{mob}\}$*, there is a perfectly reasonable solution* $S' = \{t_{dione}\}$*. One might want to classify solutions that rely in an essential way on paraconsistent reasoning as "controversial", and call other solutions "uncontroversial". In this example there are only controversial solutions to the goals* $\{g_{mob}, g_{encrypt}\}$*, because the WEP insecurity is in conflict while our network requires WEP.* ■

2.2. Possible logics for paraconsistency

Example 2 illustrates how paraconsistency can be a fairly natural concept in solving the requirements problem. We survey here a number of possible approaches to defining $\mid\!\sim$ for dealing with paraconsistency in requirements. Some of these have been proposed by other researchers, while others are among the choices we favor.

2.2.1. Maximal Consistent Subsets

A favorite approach in artificial intelligence and database research for reasoning from an inconsistent theory Δ, dating back at least to the work in logic of Rescher and Manor [17], is to consider the set of maximally consistent subsets of Δ:

$$\mathsf{MC}(\Delta) = \{S \mid S \subseteq \Delta, S \not\vdash \bot, \nexists S' \text{ such that } S' \subseteq \Delta, S' \not\vdash \bot, S \subset S'\}$$

Two approaches to defining an $\mid\!\sim$ from this are

credulous: $\Delta \mid\!\sim \varphi$ iff *there exists* $\Pi \in \mathsf{MC}(\Delta)$ such that $\Pi \vdash \varphi$;

skeptical: $\Delta \mid\!\sim \varphi$ iff *for all* $\Pi \in \mathsf{MC}(\Delta)$ one has $\Pi \vdash \varphi$.

While in other areas the skeptical approach makes more sense, in RE the credulous approach is preferable since we are interested in exploring possible *alternative* solutions.

There is however an aspect of the above approach which is undesirable from the point of view of Requirements Engineering: In any specific situations we are looking for a consistent set of tasks and goals which solve the requirements problem. If we allow implications to be excluded, then we might miss inconsistencies between these atoms. In other words, in finding maximally consistent subsets one can avoid implications such as $\alpha \to \bot$, which represent direct conflicts, or ones of the form $\beta \to \gamma$, which help reveal inconsistencies. Since the set of all such implications (Implications(Δ)) in a Horn-logic theory corresponding to a requirements problem is consistent in any case, we impose the additional requirement that $\mid\!\sim_{MC}$ choose only Π that contain Implications(Δ).

2.2.2. Minimal Repairs

A different approach to deal with inconsistency is to consider *potential repairs* that restore consistency to Δ, and then reason in these consistent theories. Such repairs involve formulas that are removed from or added to Δ. It makes sense to consider repairs that make as small a change as possible and keep as many of the old formulas as possible, thereby minimizing rework. Following Fagin et al. [18], we say that a repair theory T_1 of Δ is a *smaller change* than T_2 if T_1 restores consistency to Δ, and either T_1 has fewer deletions than T_2[4] or T_1 has the same deletions as T_2 but fewer insertions than T_2. Furthermore, we also want this to be a minimal change, in that there is no theory T' that also repairs Δ with a smaller change than T_1.

In this approach formulas in Implications(Δ) are treated as "integrity constraints", which persist. (This idea was suggested by [18] as a semantics for database updates.) It has been shown [19] that the set of all such minimal repairs is homomorphic to $\mathsf{MC}(\Delta)$, so we arrive at the same solution as in the previous subsection. The above approach is clearly syntax-dependent in the sense that it depends on how the formulas in the theory Δ are expressed. For example, the formula $p \wedge q$ in Δ would have to be removed as a single unit, as opposed to having the choice of removing either p or q in case Δ contained $\{p, q\}$ instead of $(p \wedge q)$. To avoid syntax-dependence, researchers, including Lembo *et al* [19], usually consider dealing instead with the logical closure Δ^* of the theory. Of course, if Δ is inconsistent then its logical closure is the set of all formulas, and is of no interest. Instead, one considers the set $\mathbf{lcl}(\Delta)$ of logical consequences of consistent subsets of Δ, once again insisting on including all Implications(Δ) in such subsets.

In the case of our language $T1$ (see below), corresponding to propositional Horn theories, there are no conjunctions being asserted in Δ, only atoms, and so the only additions in $\mathbf{lcl}(\Delta)$ are intermediate goals that can be derived by Horn rules. For example, the use of this approach would allow one to start from inconsistent $\Delta = \{a, b, a \wedge b \rightarrow \bot, a \rightarrow c, b \rightarrow d, c \wedge d \rightarrow g\}$, and use a consistent subset of its logical closure such as $\{c, d, a \wedge b \rightarrow \bot, a \rightarrow c, b \rightarrow d, c \wedge d \rightarrow g\}$ to achieve G. This solution is undesirable from the point of view of Requirements Engineering, since c and d are subgoals, not tasks. Therefore this approach does not yield a new $\mid\sim$ for our purposes.

2.2.3. Multi-valued logics

An alternative approach to deal with inconsistency is to resort to a logic with non-standard truth values. An example of this is Belnap's 4-valued logic [20], where an atom can be assigned one of the truth values $\{TF, T, F, \emptyset\}$. These are thought of as corresponding to subsets of standard $\{True, False\}$, and make sense in a situation where one accumulates evidence towards both the truth and falsity of a formula. Thus TF corresponds to conflicting evidence, while $\emptyset$ corresponds to no evidence. In this logic there is no inconsistency per se, since

[4] That is, $T - T_1 \subset T - T_2$

even if one has $\{a, b, a \rightarrow \neg b, b \rightarrow \neg a\}$[5], the effect is that both a and b receive truth value TF, but no arbitrary formulas can be derived with truth value T, as in a standard logic.

To arrive at a consequence relation, $\mid\sim_4$, based on this idea, we follow a solution also used by Sebastiani et al [7]: create a new theory where for every atom such as a there are two atoms EvidenceFor_a and EvidenceAgainst_a; these will be assigned standard truth values, so that both being $True$ corresponds to truth value TF assigned to a, while both being $False$ corresponds to $\emptyset$. An implication like $a \rightarrow \neg b$ is then replaced by EvidenceFor_a $\rightarrow$ EvidenceAgainst_b and EvidenceAgainst_a $\rightarrow$ EvidenceFor_b. Definite clauses such as $a \wedge b \rightarrow c$ are translated into a series of implications: EvidenceFor_a $\wedge$ EvidenceFor_b $\rightarrow$ EvidenceFor_c, EvidenceAgainst_a $\rightarrow$ EvidenceAgainst_c, and EvidenceAgainst_b $\rightarrow$ EvidenceAgainst_c. Finally, the assertion of a in a theory then corresponds to asserting EvidenceFor_a. If one calls the resulting theory $transform(\Delta)$, then note that all Horn clauses are definite and no negation appears, so that the theory is always consistent.

One can then search for minimal (abductive) sets of tasks S such that $\{\text{EvidenceFor\_}t \mid t \in S\}$ classically entail EvidenceFor_g, where g is a desired goal. Of course, S together with the implications might also entail EvidenceAgainst_g. So, in this sense, contradictions are still present, but they do not "infect" reasoning about all other goals. One can therefore consider defining consequence relation $\mid\sim_4$ such that $\Delta \mid\sim_4 \varphi$ iff $transform(\Delta) \vdash \text{EvidenceFor\_}\varphi$ where $transform(\Delta)$ is the above-sketched mapping of the Horn theory Δ into rules and atoms involving evidence for and against the atoms.

We remark that Sebastiani et al. have extended this for the i^* goal model, which includes so-called "soft-goals" and "contribution links", with the addition of intermediate-strength truth values WeakEvidenceFor and WeakEvidenceAgainst. This allows support for non-functional requirements introduced in [21].

2.2.4. Default Rules

Zowghi and Offen [22] and Ghose [23] both use default logic as part of their approach to requirements modeling and evolution. Zowghi and Offen approach things from a verification perspective. Their central concern is to ensure that the requirements specification is complete and consistent following change. To evolve a specification, Zowghi and Offen define a partial order over the requirements in order to select the requirements that should be removed to maintain consistency. Like us, Ghose is concerned with avoiding premature commitment. However, Ghose insists on obtaining from an oracle the possible critical states of system behaviour, which he calls trajectories. With this predictive oracle, the solutions to the requirements problem captured in his language can then be optimized. The oracle is capturing significant contextual variation in the assumptions.

In both these works, default rules are used to "complete in a consistent

[5]The two implications are equivalent to $a \wedge b \rightarrow \perp$

manner" potentially incomplete requirements. However, there is no discussion on how to take an inconsistent theory Δ and replace (some) formulas by default rules in Reiter's default logic, say, in order to get only consistent extensions. If we make all non-implications in Δ be defaults, then the result is equivalent to that discussed in Section 2.2.1 – finding maximal consistent subsets. And if in fact we made only elements of D be default, we could ask for minimal subsets of tasks that solve the requirements problem, as desired.

We point out that their approach is intended to work for general First Order Theories, while our work is intended for the much more restricted case of Horn propositional theories. Finally, it is important to note that both papers focus on the logic of evolution, which can be quite complicated since arbitrary formulas can be retracted.

2.2.5. Labeled Quasi-Classical Logic

Besnard & Hunter's quasi-classical logic (QCL hereafter) [24] has a paraconsistent consequence relation defined over a subset of proof rules from classical logic, without *ex falso quodlibet*. QCL accepts standard definitions of connectives and associated equivalences from classical logic, so that the form of the premises does not affect the conclusion in a proof. To enable the tracking of formulas in deductions, and thereby inform the resolution of inconsistencies during RE and more generally in software specification, Hunter & Nuseibeh [25] added labels (as in Gabbay's labeled deductive systems [26]) on formulas of QCL, and let the labels move through proof rules of QCL. The result is Labeled QCL, which Hunter & Nuseibeh used to permit the resolution of inconsistency during specification development. This work prefigures ours in using a paraconsistent logic for continuing to reason in the presence of inconsistency, although different inferences are drawn using the consequence relation in Labeled QCL.

3. RE-KOMBINE

We now define a framework, RE-KOMBINE, for managing the inconsistency in requirements problems. We do so by viewing the requirements as a Requirements Engineering Theory (RETH) – a set of requirements formulas, which is manipulated by several operators, including ones allowing us to check what can be deduced from it by logical inference. We will use a paraconsistent inference relation $\mid\sim$. We discussed in Section 2.2 a number of candidates for $\mid\sim$ which we have considered, and now proceed with it as a parameter of our modeling language and our operator definitions. Our framework consists of:

- a language, T1,
- various operators on RETH specified logically, and
- a tool implementing these operators.

We evaluate this framework in Section 4.

3.1. Requirements Modeling Language T1

Recall that our RETH needs to model the requirements problem stated in the Introduction. To model this requirements problem, we use a variant of the Techne [27, 9] requirements modeling language, called Techne 1 (T1). The reason why we call it T1 is that it is the smallest sublanguage of the original Techne, which still allows us to model a requirements problem analogous to Zave & Jackson's.

In T1, we have the following concepts and relations:

- Goals (G), replace Zave & Jackson's Requirements, and refer to desirable conditions and behaviors of the system-to-be and/or its environment.
- Domain assumptions (D), refer to conditions assumed to hold in the system-to-be itself or its environment, and which the system-to-be will not be able to change through its operation.
- Tasks (T), replace Zave & Jackson's Specification (S), and refer to what we can do to realize goals.
- Refinements and conflicts, which we call Implications (and denote *Impl*) hereafter, capture the refinement and conflict relations among the terms in the preceding three sets, using the language T1.

The requirements problem statement starts from the above, and a set of desired goals $\mathit{desired}(G)$, and seeks a minimal subset of tasks S such that $S \cup D \cup \mathit{Impl} \mathrel{|\!\sim} \mathit{desired}(G)$.
The language T1 is specified by the following grammar, whose start symbol is *wff*:

$$\mathit{prop} ::= p \mid q \mid r \mid s \mid \ldots \tag{2}$$

$$\mathit{sortedProp} ::= \mathit{prop}^{\mathsf{G}} \mid \mathit{prop}^{\mathsf{D}} \mid \mathit{prop}^{\mathsf{T}} \tag{3}$$

$$\mathit{rule} ::= \bigwedge_{i=1}^{n} \mathit{sortedProp}_i \rightarrow \mathit{sortedProp} \mid \bigwedge_{i=1}^{n} \mathit{sortedProp}_i \rightarrow \bot \tag{4}$$

$$\mathit{wff} ::= \mathit{sortedProp} \mid \mathit{rule} \tag{5}$$

A T1 RETH Δ is a finite subset of the language T1.

The following extends the sorting of atomic propositions to a "type system" that covers all wffs and reflects the ontology of Goals, Domain assumptions, Tasks, and Implications.

$$\begin{aligned} \textsc{Type} ::= & \mathsf{G} \mid \mathsf{D} \mid \mathsf{T} \mid \mathsf{Refinement\ implication} \\ & \mid \mathsf{Realization\ implication} \mid \mathsf{Conflict\ implication} \end{aligned} \tag{6}$$

The following are the typing rules for wffs using the standard notation of programming languages, where the colon ':' refers to the typing judgement, so that $\psi : \tau$ is interpreted as "expression ψ has type τ."

$$\frac{}{p^{\mathsf{G}} : \mathsf{G}} \qquad \frac{}{p^{\mathsf{D}} : \mathsf{D}} \qquad \frac{}{p^{\mathsf{T}} : \mathsf{T}}$$

$$\frac{\phi_1 : \mathsf{X}_1, \ldots, \phi_n : \mathsf{X}_n}{\bigwedge_{i=1}^{n-1} \phi_i \rightarrow \phi_n : \text{Refinement implication}} \quad \exists\ j,\ 1 \leq j \leq (n-1),\ \mathsf{X}_j = \mathsf{G}$$

$$\frac{\phi_1 : \mathsf{X}_1, \ldots, \phi_n : \mathsf{X}_n}{\bigwedge_{i=1}^{n-1} \phi_i \rightarrow \phi_n : \text{Realization implication}} \quad \forall\ j,\ 1 \leq j \leq (n-1),\ \mathsf{X}_j \in \{\mathsf{T}, \mathsf{D}\}$$

$$\frac{\phi_1 : \mathsf{X}_1, \ldots, \phi_n : \mathsf{X}_n}{\bigwedge_{i=1}^{n} \phi_i \rightarrow \bot : \text{Conflict implication}} \quad \forall\ j,\ 1 \leq j \leq n,\ \mathsf{X}_j \in \{\mathsf{G}, \mathsf{T}, \mathsf{D}\}$$

The informal reading of the implication types is as follows:

- Refinement implication is needed in order to capture the refinement relation, where a goal or task or domain assumption is refined by two or more other goal, task, or domain assumption instances. At least one of the refining instances has to be a goal. This is because, when we refine, we are moving across levels of abstration, from less detailed, to more detailed information, but we still have not reached the point where we bottom out on tasks and domain assumptions only.
- Realization implication is used when we have knowledge of what to do, and under which assumptions, in order to operationalize a goal, task, or domain assumption. When we have a realization relation to a goal, this means that we are relating that goal to a way to satisfy it, via tasks and domain assumptions. A convenient way to think about realizations is that they tell us how the possible designs of the system-to-be might satisfy the goals. We do not, however, require that all realizations target goals, since tasks can be abstract enough to require adding detail on how to realize them. Realization can also target a domain assumption, as some tasks might be required in order to maintain a domain assumption.
- Conflict implication is used to specify the conflict relation, between any propositions which cannot be satisfied together.

The above reflects the ideas in our work on the Techne language: requirements problems are structured, representing notions ranging from high-level requirements ("sell more products") to low-level tasks ("use Moneris payment terminals"). Hence the refinement implication. A key part of solving requirements problems is to find ways to refine requirements so they are eventually reduced into tasks, and to record conflicts between requirements.

3.2. What we can and cannot say using T1

We discuss in this section how relations in existing requirements modeling languages can be defined in T1. Our aim is to illustrate the versatility of the language, despite its simplicity. We start by introducing the *minimal consistent inference* relation (MCI).

Definition 1. *Suppose* $Impl(\Delta)$ *returns all implications in* Δ*. A proposition* q *in a T1* RETH Δ *will be said* to be in the MCI relation *to propositions* $\{p_1, \ldots, p_n\} \subseteq \Delta$*,* $n \geq 1$ *if and only if:*

1. $(p_1 \wedge \ldots \wedge p_n \rightarrow q) \in Impl(\Delta)$*;*
2. $\nexists \Pi.\Pi \subset \{p_1, \ldots, p_n\}$ *and* $\Pi \cup Impl(\Delta) \mid\sim q$*;*
3. $\nexists \gamma.\gamma \in Impl(\Delta), \{\gamma\} \cup \{\bigwedge_{i=1}^{n} p_i\} \vdash \bot$*.*

The first condition requires that there be an implication between the propositions. The second, minimality condition requires that there be no subset of the premises from which the consequence can be deduced via $\mid\sim$*. The third condition requires that the premises be consistent in one step.*

We use MCI to define the concept of *argument* in T1. An argument puts together the premises and the conclusion which stand in an MCI relation.

Definition 2. *The pair* (Π, q) *is an argument in a requirements theory* Δ *if and only if:*

1. $\Pi \subseteq \Delta$*,*
2. q *is in the MCI relation to all propositions in* $\Pi \setminus Impl(\Delta)$*.*

By restricting the types in premises and the conclusion of arguments, we can define a taxonomy of relations, and thereby illustrate that T1 can capture oft-cited relations in requirements modeling languages.

MCI is the root of the relations taxonomy, and is specialized onto the Refinement, Realization, and Conflict relation. Each of these is defined by restricting the types allowed in the premises and the conclusion, according to the type system introduced above.

It is then straightforward to observe the following:

- The goal refinement relation from Darimont and van Lamsweerde [28] can be defined in T1 as a specialization of the Refinement relation. Namely, Goal Refinement in T1 is the Refinement relation, in which all premises and the concusion are of type Goal. Recall that Darimont & van Lamsweerde defined goal refinement as the relation between a goal being refined and subgoals which refine it, the latter having to satisfy three conditions: (i) be sufficient to deduce the refined goal, (ii) be minimal, and (iii) be consistent. All three conditions are satisfied here, since Goal Refinement is a specialization of Refinement, which is in turn a specialization of MCI.

- Yu and Mylopoulos [29] introduced task decomposition in i* (i-star). It is similar to goal refinement, with two differences: (i) the requirement being refined/decomposed must be a task, and (ii) it can be refined by any combination of goals and tasks.[6] Task Decomposition can be defined in T1 as a specialization of the Realization relation, in which all premises and the conclusion are all of type Task.

- The goal operationalization relation in KAOS [30] is similar to the means-ends relation in i*. The idea of both is that tasks should be executed in order to satisfy goals. Operationalization in KAOS stands between goals and constraints, whereby a constraint is operational, in the sense that it is formulated in terms of objects and actions available to the agents in/of the system. Means-ends rather emphasizes the role of goals as reasons why tasks are executed, i.e., a task exists in a requirements database because it is a means to a goal. The Goal Operationalization relation, which captures the idea of both goal operationalization and means-ends, is a specialization of the Realization relation, in which all premises are tasks or domain assumptions, and the conclusion is a goal.

To see the kinds of conflict we can capture using our Conflict relation, we first need the concept of Alternative.

Definition 3. *Given* $\Pi \subseteq \Delta$, $\Phi \subset \Pi$ *is an* ***alternative*** *in* Π *if and only if:*

1. *There is an argument* $(\Pi, \bot)$*;*
2. $\Phi \nvdash \bot$*;*
3. $\forall \Psi.\Psi \subseteq \Pi$ *if* $\Psi \nvdash \bot$ *then* $\Phi \not\subset \Psi$*; i.e.,* Φ *is a maximally consistent subset of* Π*;*
4. Φ *does not include only implications and/or Domain assumptions. In other words,* Φ *includes at least one goal and/or task.*

The set of all alternatives in Π *is denoted* $\mathsf{Alt}(\Pi)$*.*

Using the Alternative concept, we specialize the Conflict relation as follows:

- Type-A Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if there are at least two alternatives in Π, i.e., $|\mathsf{Alt}(\Pi)| \geq 2$.

- Type-B Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if $|\mathsf{Alt}(\Pi)| = 1$. Informally, a Type-B conflict involves domain assumptions which are blocking the satisfaction of goals or the execution of tasks. If instances in Π are in Type-B Conflict, then we have $\mathsf{Block}(\Pi) \stackrel{def}{=} \Pi - \mathsf{Alt}(\Pi)$, and we call $\mathsf{Block}(\Pi)$ the set of blockers.

[6] There is no concept in i* [29] which corresponds to Domain assumption, so it is not allowed here to have Domain assumptions in a decomposition of a task.

- Type-C Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if $|\mathsf{Alt}(\Pi)| = 0$. Type-C Conflict involves a minimally inconsistent set which includes only Domain assumptions.

Example 4. *Recall our running example using WEP and the PCI-DSS (Example 1). Given the specialization of the Conflict relation above, a Type-B conflict exists between $d_{4.1.1}$ and t_{wep}. There is no Type-A conflict since there are no alternatives to t_{dione}, and no Type-C conflicts.* ■

We can also relate Conflict in T1 with notions of conflict in other requirements modeling languages.

In KAOS, the Conflict relation is also a minimally inconsistent set of requirements. Obstruction and Divergence are two relations, also in KAOS, which involve domain assumptions that block, in the sense discussed above, the satisfaction of a goal or the execution of a task.

Table 1 summarizes the translation of conflict relations identified in Robinson, Pawlowski & Volkov's survey [11] into T1. Conflicts listed in that table cannot obtain in T1 definitions that are as convenient as, e.g., Type-A, Type-B, or Type-C Conflict relations. In a propositional formalism such as T1, there is no elegant way to formally talk about instances of classes, and their deviations: a proposition stating the deviation of an instance will be different from a proposition stating normal behavior of other instances, but there is no relation which would say that the two propositions talk about instances of the same class.

3.3. Operators for (Paraconsistent) RETH

Having defined our language, we now turn to an operational definition of how to solve requirements problems. RE-KOMBINE is defined in a functional style, specifying update and query operators on the RETH, which are a modified version of those first presented by Ernst et al [9]. We describe the operators in the style of Javadoc by naming the parameters and their types (using $\wp(S)$ to represent the set of all subsets of S), as well as the assumptions and effects of the operators. Operators are transactional, leaving things untouched if an exception occurs. For simplicity we omit in the definitions the implicit parameter specifying which RETH is being considered; of course, this could be added if one wanted to work on multiple theories at the same time.

First, we need an operator to introduce new propositional symbols (for goals, tasks, and domain assumptions, i.e., those TYPEs from T1, introduced in Section 3.1), without asserting them.

Operation 1 — Declare-Atomic

@param atomName : *String*

@param sort : $\{G, T, D\}$

@effect Add typed atom to the symbol table of the RETH.

@throws an exception if the name is already used.

Table 1: Translation to T1 of the conflict relation types in Robinson, Pawlowski & Volkov's survey [11].

Relation in the survey	*Corresponding relation in* **T1**
Process-level deviation: deviation of the actual process of developing the system from the predefined process.	Either a Type-B conflict, in which the blocker is a domain assumption stating the deviation between the planned and actual development process, or a Type-C conflict where one of the domain assumptions states that deviation.
Instance-level deviation: an instance of an implemented class violates a requirement.	A Type-B conflict, which has one blocking domain assumption. That domain assumption names the instance responsible for the violation, and the alternative in the Type-B is the requirement violated by that instance.
Terminology clash (also *Structure clash*): a member of the semantic domain is being referred to using more than one symbol/-expression.	None of the conflict relations captures terminology clashes. The terminology clash is an error in the use of a requirements modeling language.
Designation clash: a symbol/expression refers to two or more different members of the semantic domain.	As for the terminology clash, a designation clash is an error in the use of the formalism, and cannot be captured in the formalism.
Conflict: a set of requirements is logically inconsistent.	Conflict relation.
Divergence (also *Obstruction*): A set of requirements is logically inconsistent when a certain sequence of events can occur.	A Type-B conflict, where the blocked requirements are an alternative and the blocker is a domain assumption describing the problematic sequence of events.
Competition: A kind of divergence where particular instances of a requirement can cause a divergence.	A Type-B conflict, which has one blocking domain assumption. That domain assumption names the instance responsible for the violation, and the alternative in the Type-B is the requirement violated by that instance.

Use the following operator to assert a formula as being part of the requirements:

Operation 2 — Assert-Formula

@param wff : *Formula*

@effect Adds wff to the theory RETH.

@returns Boolean (False iff the resulting theory is classically inconsistent).

@throws An exception is thrown if the wff is not well-formed or has undeclared atoms.

Before proceeding to solve inconsistent requirements, the user might want to discover the causes of the inconsistency. For this purpose the following operator should be used:

Operation 3 — Find-Reasons-For-Inconsistency

@returns $\wp(\wp(\mathsf{wff}))$ where each element S of the set returned is a *minimal* subset of RETH from which False can be classically derived: $S \vdash \bot$.

Note that if the RETH is consistent, then this operator returns the empty set. Using this operator when an inconsistency is first signaled allows the user to find which formulas cause inconsistency, and possibly repair things by retracting some formulas and then possibly adding them back corrected.[7] Of course, the user may choose to keep the RETH inconsistent, as we have argued.

The core operator that solves requirement problems is next.

Operation 4 — Paraconsist-Min-Goal-Achieve

@param desiredG : $\wp$(GOALS)

@return TaskSets : $\wp(\wp(\text{TASKS}))$ consisting of all sets S of tasks such that RETH $\cup$ $S \mid\!\sim$ *desired*(G), and no subset of S has this property.

@throws exception if desiredG $\cup$ Implications(RETH) $\vdash \bot$.

The exception above occurs in the case when the goals are conflicting in and of themselves: in Horn RETH, Implications(RETH) is always consistent, so if the addition of wanted goals causes inconsistency then the goals must conflict.

The Paraconsist-Min-Goal-Achieve operation supports what has been called "backward reasoning" in the RE literature [7]. Backward reasoning sets some high-level goals as desiderata, and determines a minimal set of tasks that can accomplish those goals. In this sense Paraconsist-Min-Goal-Achieve is an *abductive search*. Abduction normally works from consistent theories, since one is not interested in explanations of how g can be derived if one assumes q and $\neg q$. We rely on the appropriate choice of $\mid\!\sim$ to provide a reasonable solution to abduction in the presence of inconsistency. Note that the minimality of S in the above specification also prevents $\mid\!\sim$ from choosing S that have conflicting tasks.

Note that this operator also supports paraconsistency by allowing only a subset of the final goals to be passed in as an argument, as opposed to adding to

[7] We do not show the retraction operator, for brevity. However, one retracts only asserted formulas, not their implications, and so it is uncontroversial.

the RETH *all* top level goals from the beginning. In this way, the requirements engineer can see what consistent solutions exist for parts of the problem, and resolve inconsistency issues incrementally.

Example 5. *Consider again the requirements problem defined in Example 1 as* $R = \{t_{dione} \rightarrow g_{mob}, t_{wep} \rightarrow g_{mob}, d_{impl} \rightarrow t_{wep}, d_{4.1.1} \rightarrow g_{4.1}, g_{4.1} \rightarrow g_{encrypt}, t_{wep} \wedge d_{4.1.1} \rightarrow \bot\}$, $D = \{d_{4.1.1}, d_{impl}\}$. *If we then let* desiredG $= \{g_{mob}, g_{encrypt}\}$, *the problem is* classically *inconsistent, meaning all and any answers can be derived, since the subset* $\{d_{impl}, d_{4.1.1}, t_{wep} \wedge d_{4.1.1} \rightarrow \bot\}$ *is inconsistent. However, given an appropriate paraconsistent* $\mid\sim$, *the operation* Paraconsist-Min-Goal-Achieve *can identify an answer* $S_1 = \{t_{dione}, d_{4.1.1}\}$. *This supports our desire to continue to reason despite a conflict.* ■

When the RETH is inconsistent, paraconsistent reasoners usually ignore certain formulas (or their consequences), in order to avoid *ex falso quodlibet*. Such solutions may be somewhat unintuitive and bear examination by the requirements engineer. However, some goals can be achieved without involving any "questionable" formulas – i.e., it is as if the RETH is consistent as far as the goals concerned. We would like such cases to be distinguished, much like the concept of FREE formulas in Hunter and Nuseibeh's LQCL approach [31]. Example 1 Revisited illustrated the problem, and called for solutions to be marked as (un)controversial. Our approach to this is to find a way of marking formulas that are "relevant" to the deduction of sets S above, and then distinguishing formulas that are controversial. In this way the decision to exclude controversial formulas is left to a requirements engineer.

Definition 4. *A formula* φ *will be said to be* relevant to the solution S of $D \cup R \cup S \mid\sim G$ *if* $(D \cup R \cup S) - \{\varphi\} \not\mid\sim G$.

A formula φ *will be said to be* controversial to the solution S of $D \cup R \cup S \mid\sim G$ *if it belongs to one of the elements of* Find-Reasons-For-Inconsistency$(D \cup R \cup S)$.

Note that if we were dealing with a particular paraconsistent logic, such as Relevance Logic [32], these issue would be addressed directly in its axiomatization. However, we are using an abstract $\mid\sim$ at this point, so our definition needs to be more general, and hence potentially less accurate.

We provide support for finding such information via the next operator.

Operation 5 — Examine-Solution

@param desiredG : $\wp$(GOALS)

@param Soln : $\wp$(TASKS)

@assumes Soln is one of the solutions to the requirements problem for desiredG

@return Explanations : $\wp(\wp(\langle \mathsf{Wff}, \{\text{“}controversial\text{”}, \text{“”}\}\rangle))$ consisting of all minimal sets of formulas S in RETH $\cup$ Soln used to derive desiredG, with controversial formulas marked. (We need sets of sets since there may be multiple explanations.)

Note that if we reformulate Example 2 in a more natural way to be a specification where all nodes are goals, eliminating assumptions, there is no

formal inconsistency per se in its translation to logic. There is also no solution, because not all goals have been refined to tasks and domain assumptions. However *there is no possible solution* to achieving all top level goals, *no matter how one tries* to complete the requirements problem by adding refinements and realizations of existing leaf goals. Obviously, requirements engineers would like to know this information as early as possible. The following operator can provide it:

Operation 6 — Exist-Solution

@param desiredG : $\wp$(GOALS)

@return Boolean – True, iff there is a set $G_0 \subset G$ consisting of only unrefined goals (leafs in diagrams like Figure 1 and 2) such that $D \cup$ Implications(RETH) $\cup\, G_0 \cup T \;\not\sim$ desiredG

@throws exception if *desired*(G) $\cup$ Implications(RETH) is inconsistent.

A more helpful variant of this operator would return "the reasons" for False answers—minimal sets of formulas that cause the absence of solution. We leave this problem to future work.

In the requirements problem, we are interested in optimality with respect to the stakeholders communicating the requirements for the new system. In that context, the stakeholder may not be content with a subset-minimal implementation that satisfies the requirements mandated (as returned by Paraconsist-Min-Goal-Achieve). Rather, he or she is interested in implementations which also satisfy other, less important goals. Furthermore, while still subset-minimal with respect to tasks, we add the constraint that the set of goals achieved is maximized. This answers the question, *"If I wish to accomplish the following extra goals, in addition to certain mandatory requirements, what are the minimal sets of tasks I must perform?"*

Operation 7 — Paraconsist-Get-Candidate-Solutions

@param desiredG : $\wp$(GOALS)

@param wishedG : $\wp$(GOALS)

@return set of pairs $\langle$solnT, satG$\rangle$, where solnT is a set of tasks, and satG is a set of goals such that 1) RETH $\cup$ solnT $\not\sim$ satG; 2) satG = desiredG (the required goals) $\cup$ wishedG$_0$ (a subset of the wished-for goals wishedG), such that satG $\cup$ Implications(RETH) is consistent; 3) satG is maximal with respect to the above properties; 4) solnT is subset-minimal to achieve the above.

@throws an exception if desiredG is inconsistent with Implications(RETH).

3.4. Tool-supported RE-KOMBINE

3.4.1. Implemention based on ATMS

We have implemented the RE-KOMBINE framework and operators, where $\not\sim_{MC}$, as discussed in Section 2.2.1, is used for $\not\sim$, by using an Assumption-based Truth Maintenance System (ATMS) [33], because an ATMS naturally supports

our simple definition of well-formed formulae, abduction and paraconsistent reasoning.

In an ATMS, the RETH is represented as a set of nodes, corresponding to the atoms, with Horn clauses linking them to form a graph. Horn-rules $\alpha \rightarrow \bot$, indicating conflicts, connect the antecedent nodes to a special node called CONTRADICTION (originally called "NOGOOD" in [33]). The ATMS implementation then associates with each node a set of possible *environments* in which that node is interpreted as true (originally marked by :IN). Environments are sets of assumptions which ultimately justify that node (i.e., from which that node can be derived from assumptions via definite Horn-rules called justifications.) The *label* for some node n will take one of three values:

- if there is no justification for n, the environments are said to be empty: $\langle n : \{\}\rangle$;
- if the node is always :IN, i.e., it is an assumption (in our case, a task), then the label has an empty environment: $\langle n : \{\{\}\}\rangle$;
- in all other cases, the node is labelled with all environments, or sets of assumptions, from which it can be derived :IN, e.g., $\langle n : \{\{a, b\}, \{c, d, e\}\}\rangle$. Most importantly, as added conditions, (i) these sets are minimal – no nodes can be removed from such a context without losing the full justification; and (ii) the sets are consistent, in the sense that no contradictions ($\bot$) can be derived from them.

When constructing the ATMS, we need to distinguish rules of the form $\psi \rightarrow t$ where t is a task and ψ has no tasks in it i.e., they are all domain assumptions. Such rules should be replaced by $\psi \wedge \mathit{actual}[t] \rightarrow t$, where *actual[t]* is a new task. We have the following result supporting the correctness of this implementation, which is an immediate consequence of the material in [33]:

Proposition 3.1. *If an ATMS is constructed from $D \cup R$ in the manner given above, and all leaf nodes are made :IN, then for any goal g, its node N_g contains environment with tasks S if and only if* Paraconsist-Min-Goal-Achieve$(\{g\})$ *returns, among others, S*[8]*, if $\mid\sim_{MC}$ is used as the paraconsistent logic.*

It can happen that in a requirements model, relations between requirements create cycles in the graph. Cycles are supported in the ATMS because of short-circuit evaluation. If a node has an existing environment which subsumes a possible new environment, evaluation terminates.

Figure 3 gives an example of using the RE-KOMBINE operators for capturing requirements problems in Techne, implemented using an ATMS library with our operators implemented on top, in Common Lisp.[9] As discussed in Section 3.3, the operation declare-atomic introduces (but does not assert) goals in the model,

[8] Where *actual[t]* is replaced by t

[9] Our implementation is available at http://github.com/neilernst/Techne-TMS.

```
(defvar
g0 (declare-atomic "Comply with PCI DSS" :GOAL *rekb*)
t1 (declare-atomic "Build and Maintain Secure Network" :TASK *rekb*)
t2 (declare-atomic "Protect Cardholder Data" :TASK *rekb*)
t3 (declare-atomic "Maintain a Vulnerability Mgmt Program" :TASK *rekb*)
t4 (declare-atomic "Implement Strong Access Control Measures" :TASK *rekb*)
t5 (declare-atomic "Regularly Monitor and Test Networks" :TASK *rekb*)
t6 (declare-atomic "Maintain an Info Security Policy" :TASK *rekb*))

(assert-formula  g0 (list t1 t2 t3 t4 t5 t6) :DA *rekb*)

(min-goal-achieve g0 *rekb*)
```

Figure 3: Simple example introducing T1 formulae into RE-KOMBINE. The result of the final call to `min-goal-achieve` is the set {t1, t2, t3, t4, t5, t6}.

while assert-formula asserts in this case inference or conflict relations between g0 and its antecedents. Note the use of **rekb**, a global variable referring to the current instance of RETH and the *:GOAL* keyword symbol representing the atom's Type per T1.

The tool supports a graphical front-end with a translation engine to the operators. The user may also write directly in the textual DSL language of Listing 3. Output of the reasoning is likewise textual or graphical. In agile software development, in particular, it is important that the process artifacts be perceived as nearly invisible (hence the frequent use of index cards and whiteboards). More study is needed into the effort required to compose RE-KOMBINE models. What we have done is attempt to keep the core language as simple as possible, and, as we show in Section 4, make the tool close to interactive for most requirements models. The essential tradeoff is that speed of development needs longer-term support, particularly in more complex domains.

We are pursuing integrating RE-KOMBINE with a commercial requirements tool like DOORS or Jira, where the intention is to permit requirements to be captured easily and managed using RE-KOMBINE. The workflow is for requirements elicitation to proceed as usual, with the sole exception being that the requirements engineer or developer enters the requirements as Techne statements (which are simple propositional statements with formal relations). Then, during the prioritization phase at the beginning of a development iteration, the RE-KOMBINE tool can provide answers regarding which requirements/features/user stories to work on.

3.4.2. Implementation using SAT-solvers

Sebastiani et al [7] implemented $\mid\sim_4$, as it was described in Section 2.2.3 , by using a minWeight SAT solver to find minimal abductive solutions. To do this, it is necessary to replace rules $\{\beta_1 \rightarrow b, \ldots, \beta_n \rightarrow b\}$ that imply some atom

b of the theory $transform(\Delta)$ by their "Clark completion" $\{(\beta_1 \vee \ldots \vee \beta_n) \leftrightarrow b\}$. In using a SAT-solver, one can also add constraints, such as $\neg$EvidenceFor_p $\vee$ $\neg$EvidenceAgainst_p that prevent solutions with conflicting evidence for p.

If one uses an ordinary SAT-solver on the above theory, one can find much more quickly *non-minimal solutions* – i.e., assignments of True/False to EvidenceFor_t$_i$ and EvidenceAgainst_t$_i$ for tasks t_i which make EvidenceFor_g be True for desired goals g.

Example 6. *Sebastiani et al.'s approach divides the problem operationally into input goals and target goals, corresponding (roughly) to T1's notion of Tasks and wanted Goals, respectively. In our ongoing example, we would assign all leaf goals to the input set, and the two root goals g_{mob} and $g_{encrypt}$) to the target set. We would like to ideally find that both target goals are True for* EvidenceFor_ *and False for* EvidenceAgainst_*, but this is not the case here: both predicates will be True, indicating conflicting evidence* $\{TF\}$. ■

4. Evaluating RE-KOMBINE

Recall that our research questions were:

RQ1 What formal mechanisms are necessary to support reasoning with inconsistent requirements models?

RQ2 Is there a scalable algorithm for automating analyses of such models?

We begin with a description of our case study of variability and evolution in requirements found in the Data Security Standard (henceforth PCI-DSS) [34], an industry standard which regulates security of credit card transactions. This is a retrospective case study, based on a real standard which we merged with a requirements model from the Alvalade stadium case [35] to examine our approach. We first constructed a requirements goal model with several versions, based on changes to the PCI-DSS standard. We then applied RE-KOMBINE to the problem to illustrate how paraconsistency is essential for solving the requirements problem in a specific example (**RQ1**). We then discuss how our framework scales to industrially relevant sizes (**RQ2**), using the case study and random models.

4.1. Case Study: Payment Card Standards and Requirements Variability

PCI-DSS version 2.0 was released in October, 2010, and is currently in force. There is a two-year cycle between major revisions, with a three month announcement window immediately prior to the new standard coming into force. This provides organizations time to achieve compliance. PCI-DSS has the following sub-goals for compliance: (i) Build and maintain a secure network, (ii) Protect cardholder data, (iii) Maintain a vulnerability management program, (iv) Implement strong access control measures, (v) Regularly monitor and test networks, (vi) Maintain an information security policy.

PCI-DSS is well-suited for representation as a requirements problem. We map requirements as goals, and constraints as domain assumptions. Tasks are used to represent compliance tests in the PCI-DSS. A solution to the requirements problem is a series of tasks which can pass the compliance audit, a *compliance strategy*. For this paper, the relevance of this case study is in describing how the changes to the standard are realized in individual organizations, which also have entirely separate sets of organization-specific requirements. This was shown in Fig 1, earlier: the organization-specific goals must be related to the standard's compliance goals. We then translate this to a domain-specific language (DSL) which can be run against the ATMS or MinWeightSAT solvers.

4.1.1. Solving the PCI-DSS Requirements Problem

We now illustrate how changes in an external standard, such as PCI-DSS, might lead to inconsistency in an organizational requirements model. We focus in particular on how the RE-KOMBINE tool can use paraconsistent reasoning to support strategic, change-tolerant decisions.

Let us return to the example from Section 1, shown again in Fig. 4. We showed that the presence of the conflict relation and existing (asserted) atoms would cause classical reasoning to fail. In RE-KOMBINE, we are able to continue to search for solutions in spite of the conflict. Recall that the requirements problem has been defined as $D, R, S \vdash \mathit{desired}(G)$. The state of the RETH, that is, the nature of the requirements problem of Fig. 4, is as follows.

Set R contains all implications and conflicts, including the refinement of g_{rev} by g_{mob} and the conflict between t_{wep} and $d_{4.1.1}$. Set G contains the twin goals of $g_{encrypt}$ and g_{mob}, and $g_{4.1}$, since in this case study the business would like to achieve both compliance and business success. The set of tasks T contains t_{dione}, the use of Dione XPlorer terminals and t_{wep}, the use of WiFi. D, the domain assumptions, contains the assumptions that task t_{wep} is already satisfied, i.e., currently the state of affairs for the business network, and that we must abide by the PCI-DSS requirements, i.e., $d_{4.1.1}$ is asserted. This would be the case if the business was compliant prior to the June 2010 enforcement of the restriction on WEP.

In this fragment of the model we are seeking to find tasks T or assumptions for R such that a subset of goals in G are satisfied. One way to do this is with the following operation. Call Paraconsist-Min-Goal-Achieve with the argument desiredG $= \{g_{rev}, g_{mob}\}$. RE-KOMBINE performs an abductive search to find minimal sets of tasks or assumptions which will satisfy desiredG. In this case, possible answers, returned as TaskSets, include only $\{t_{dione}\}$. Solutions which include assumptions which lead to conflict are excluded. In this case, since the PCI-DSS is external and presumably compliance is essential, the business would choose to phase out WEP, using preferring to use a Bluetooth solution. Again, paraconsistency in RE-KOMBINE allows us to maintain the complete model, adding or retracting tasks as required.

In the PCI-DSS there are multiple types of changes we must accommodate, including variations in adopting best practices the standard identifies (such as application security practices). One type of change is a switch between

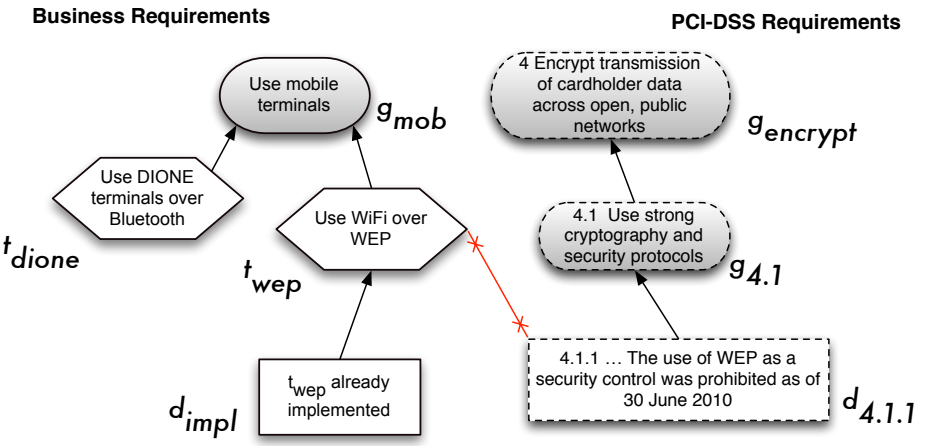

Figure 4: WEP fragment of the model from the payment card case showing a possible configuration. Grey indicates satisfied goals.

alternatives, which are variations [36]. There is little variation in the high-level requirements the PCI-DSS defines, in that a compliant organization must comply with them all. Variation does exist in three other places, however. One is that some requirements exist as *best practices*, which are recommended requirements (in that they are likely to become mandatory in the next iteration of the standard). We could manage these 'wished for' requirements using a call to Paraconsist-Get-Candidate-Solutions with a wishedG parameter (Operator 6).

With respect to domain assumptions, there are clearly variations between organizations. For instance, the context of applying the PCI-DSS to a supermarket chain will be different than for an e-commerce payment processor. There is also provision for variation *within* an organization in *proving* compliance (i.e., selecting a compliance strategy). In PCI-DSS these are known as **compensating controls**, and they define solutions for proving compliance where domain assumptions in the standard are invalid. For example (numbers in parentheses refer to the PCI-DSS standard, v2), in environments that cannot prevent multiple root logins (requirement 8.1), the organization is permitted to use SUDO (a command to give an ordinary user full but temporary privileges), just as long as the system carefully logs each access. The reason to prevent root login is that the use of a super-user account is opaque, without this control: it is not clear what physical access is behind the root account.

Fig. 5 captures this fragment of the requirements problem. Assume there are tasks satisfying leaf goals, and that there is a call to Paraconsist-Get-Candidate-Solutions with wishedG = {Use existing hardware} and desiredG = {Assign Unique ID}. Then the result is the tuple ⟨ solnT:{Log Access,Use SUDO,Use AS/400 Servers}, satG:{Use existing hardware,Assign Unique ID}⟩. This reflects the (simplistic) result that the best way to satisfy the wished-for goal to use existing hardware is to apply for the compensating control of logging access. Again, we

have placed Use AS/400 Servers into D as an assumption, since it describes the current state of affairs. In this case a conflict exists if we also assume that the use of Centralized identity management is satisfied. The paraconsistent operator has allowed us to find alternative solutions to this inconsistent state.

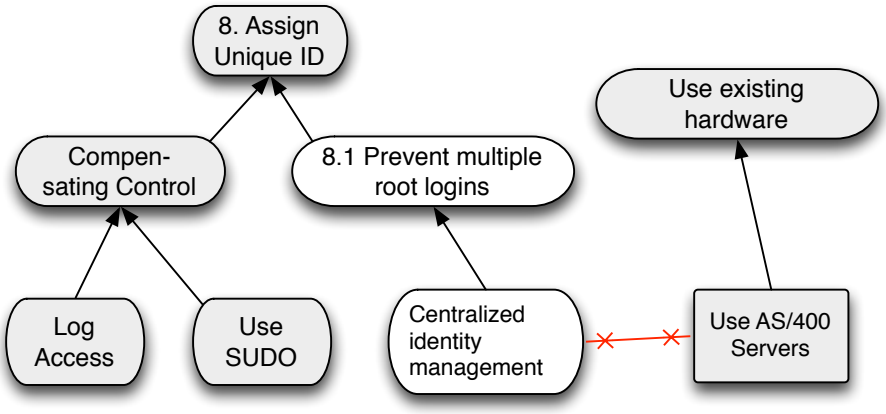

Figure 5: The graphical representation of the compensating controls example.

4.2. Demonstrating Scalability

In [9] we showed that the ATMS reasoner was scalable for industrial requirements problems. It can return decisions in less than 100 seconds on random models that were as large as six hundred requirements and two hundred relations, as shown in Figure 6. In this paper, we have extended our tool to introduce some useful pre-processing steps, including the elision of and-subtrees, which greatly reduces the number of assumptions.[10] We then ran our reasoner on the PCI-DSS case study model. We also evaluated the case study using the tool from [7], as described in Section 3.4. Applying the tool to industrial problems is predicated on a useful set of requirements propositions being generated during requirements elicitation activities, whether geared to lightweight user-story gathering or more formal use case or IEEE 930 approaches.

The examples above were derived from a complete RE-KOMBINE model of the case study. It consisted of two connected components. One was the representation of the PCI-DSS model, which has 254 requirements and 65 relationships. This is the most basic model, in which every requirements must be adhered to (thus, all relationships are AND-style). This is trivially easy to reason on. We then added subcomponents representing scenarios for variation (i.e., nodes with multiple justifications in the PCI model, representing alternatives for compliance), consisting of 41 nodes and 18 relations, and scenarios for evolution

[10]The complete model and source code is available at http://github.com/neilernst/Techne-TMS.

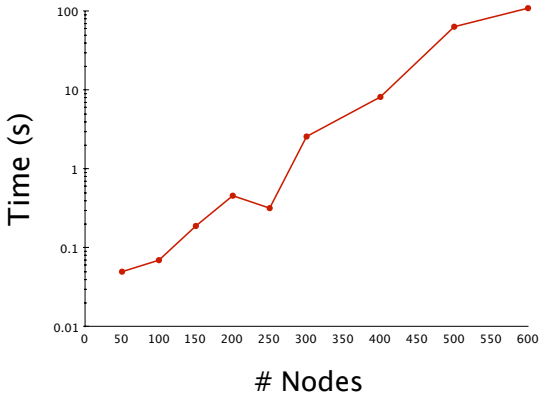

Figure 6: A log-linear chart of ATMS scalability on randomly generated requirements problems. Originally appears in [9].

(changes between version 1.1 and version 2). Finally, we created components that reflected the business objectives of a soccer stadium, based on the case study described in [35]. The final model was 342 nodes and 127 relations in size.

To compile the model in ATMS took 614 seconds, on a Macbook Pro 2.4Ghz with 8Gb RAM. This reflects the amount of time needed to generate the abductively minimal solutions for *all* possible goals in the model. Querying a set of these goals then requires a polynomial amount of comparisons to generate the answer which is trivial relative to the exponential abduction problem. This size of model compares in size with industrial examples of design requirements described in the literature (e.g., van Lamsweerde [37] listed KAOS model sizes that were, on average, 540 goals and requirements). We have found that the benefits of the ATMS are more apparent when performing incremental computations, e.g., when the model is evolved. ATMS is inherently incremental and so evolutionary changes are relatively painless.

A single call to the MinWeight SAT solver from [7], using weak conflict avoiding, did not find a single minimal solution after 30 minutes. Admittedly, the MinWeight tool is not state of the art for SAT solvers. A call to a non-minSAT solver, zChaff 2007.3.12, by comparison, returned a solution (a single satisfying instance that is not minimal) in a few milliseconds.

4.3. Threats to validity

We consider threats as described in Yin [38]. Our *construct validity* is related to how well we defined the key constructs we are examining. In this case, we look for goal oriented requirements models with inconsistencies, which are both formally defined constructs. One might take issue with how we have defined the significance of the problem of inconsistency and how relevant gross reasoning time is to the problem. *Internal validity* concerns the cause and effect nature of case study inferences. In this case we only conclude that reasoning time is acceptable based on our measures, with reference to previously established rules

for interactive applications. A threat which might be internal, but could possibly be seen as external, is whether this framework is the best approach to managing inconsistency. We have shown that it supersedes other approaches in the related work sections of this paper, but it remains an open question whether managing inconsistency formally is the preferred approach in all cases.

Finally, our *external* validity is limited to goal-oriented requirements models with similar characteristics. We would like to expand our case study to include other models, but it isn't clear how well T1 would handle requirements models which are not goal-oriented. The principles of paraconsistent reasoning are well-established in other research papers, but the tool has yet to be widely applied. We have tried to address *reliability* issues by making our source code and case study data available online.

5. Related Work

We described the work of Giorgini et al. [39] and Sebastiani et al. [7] in Section 2.2.3. Their qualitative approach can simulate some of the capability of RE-KOMBINE. A major difference in philosophy is the omission, in RE-KOMBINE, of qualitative, partial satisfaction/denial relations. RE-KOMBINE deliberately omits this notion of partial satisfaction, because in practice, this bipartite approach leads to frequent occurrences of conflicting information about a given requirement. In a dynamic environment, partial satisfaction of goals results in lack of actionable information. For example, consider the case where we know that the goal Comply with PCI-DSS is both partially satisfied and partially denied. This type of conflict can lead to analysis paralysis and a substantial cost of delay. RE-KOMBINE is tailored for automatic, binary answers over conflicting goals. Qualitative reasoning is better suited to up-front problem exploration. RE-KOMBINE's systematic, lightweight approach is more suitable when we are doing an iterative problem exploration by committing to small increments of the model.

Hunter and Nuseibeh [25] described one of the early approaches to inconsistent requirements specifications. We explained the use of their approach in Section 2.2. Our innovation is the introduction of operations on requirements problems, including the notion of minimal solutions, as a way to support design decision-making. We also feel that the simpler propositional language of Techne is more suited to the light-weight analysis common in industry.

The concepts of specification, requirements and domain assumptions also exist in formal methods research. For example, Poppleton and Groves [40] discuss the notions of *refinement* and *retrenchment*, which are used to model the transformation of a program as the specification changes. which formally implements a specification at time t_1 to a revised specification at time t_2. Retrenchment is the analysis of the old specification in light of the new environment, with an eye to identifying any structures worth preserving. Formalization in a language like Z is clearly more onerous than the one in RE-KOMBINE. The distinction is primarily in the degree of formality that the tools demand. We feel that only formalizing high-level relationships between elements in the requirements

problem is more likely to support the wide range of scenarios one might see in an agile setting.

6. Future Work

There are several open questions on the topic of reasoning from requirements knowledge bases. Paraconsistent reasoning simply means that conclusions drawn from a requirements knowledge base are based on a set of inference rules that satisfy the paraconsistency property: whenever $\perp$ is inferred, it is *not* the case that *all* formulas are also consequences of the knowledge base, unlike classical logic.

A key open question for paraconsistency for requirements knowledge bases is: What are the properties that a paraconsistent consequence relation should satisfy, independently of the specifics of the language used to represent requirements? While a paraconsistent consequence relation will not deduce everything from an inconsistent requirements knowledge base, some conclusions are clearly more appropriate than others during the requirements problem-solving process. This issue is related to the operators one may want to have when interacting with an (inconsistent) knowledge base. Consider a situation where there is a requirements knowledge base, and the engineer asks "Is there a solution?" If the answer is "No", then there are three possible reasons why:

a. A top-level goal (i.e., a member of desiredG) has no refinement into tasks and/or domain assumptions (i.e., there is no way to satisfy it). Therefore the engineer must further refine it.
b. There are conflicts between top-level goals. The engineer needs to revise the top-level goals.
c. There is no conflict-free set of tasks/leaf sub-goals/domain assumptions that refine all top-level goals. The engineer needs to either find alternative refinements for which atoms in their refinements are not in conflict, or resolve conflicts.

Observe that one can compute the answer "No" above using the classical consequence relation $\vdash$ on the requirements knowledge base (provided that its language fits the language over which the classical consequence relation is defined). But what $\vdash$ cannot do is suggest which one or more of the three options applies, or more generally, what to do once we know *that there is an inconsistency in the requirements knowledge base*. What is interesting is to have a paraconsistent consequence relation which can be used to suggest how to proceed in solving the requirements problem, rather than simply indicate that there is inconsistency, as classical consequence does.

One might also wish to distinguish domain knowledge, D_K, from domain assumptions, D_A. In the former case, we would consider such assertions potentially defeasible (i.e., retractable), while D_A would capture unchanging properties, such as physical laws.

The questions above are also related to the issue of how to state the requirements problem. For example, one of the properties that a solution to the problem

should have is that it is consistent. But being consistent means different things, as it depends on the proof theory that defines the consequence relation. Consider for example the property that $S \cup \text{Relations} \cup D_0$ should be consistent. The reflex is to define this formally as $S \cup \text{Relations} \cup D_0 \cup \nvdash \bot$, and this means that one cannot deduce $\bot$ using $\vdash$ of classical logic. But what if reasoning over the requirements knowledge base is not deductive, but argumentative: for example, an argumentative proof theory may allow a conclusion x from a set Z even if x is in conflict with some $y \in Z$, *provided* there is also a q that is evidence against y. In that case, we are not saying that there are *no* conflicts in a solution, but instead, that when there are conflicts, they stand in some specific relations that render them harmless, even in the solution.

Finally, another issue of interest is what happens when we use a richer ontology of requirements in the requirements knowledge base. In particular, what should we deduce from an inconsistent requirements knowledge base in case there are partial orders over its formulas, such as orders of preference between conflicting requirements.

7. Conclusion

In many situations, establishing the entirety of a software development project's requirements up-front is unrealistic and even undesirable. Instead, we propose a systematic approach to agile requirements evolution where it is easy to change requirements and automatically evaluate the consequences of these changes (answering **RQ1**). We further show that this reconciliation makes it possible to delay decisions about conflicting requirements until more information becomes available. Our proposal is grounded on a framework, RE-KOMBINE, for expressing requirements formally yet sufficiently flexibly as to enable deferred commitment. The paper introduces a novel definition of paraconsistency in requirements specifications using Techne as underlying propositional language. It then presents properties for defining a paraconsistent consequence operator, $\mid\sim$. Using that operator, we introduce two operations for reasoning paraconsistently on requirements models, searching for minimal solutions to the requirements problem despite the existence of contradictory or missing information. We evaluate our proposal with an industrial case study of payment card requirements, and show that the operations to scale to typical industrial design problems (answering **RQ2**). In future, we intend to continue investigating how our proposal works for even more complex, real-world problems.

References

[1] N. A. Ernst, A. Borgida, J. Mylopoulos, I. Jureta, Agile requirements evolution via paraconsistent reasoning, in: International Conference on Advanced Information Systems Engineering, Gdansk, Poland, 2012.

[2] M. Jarke, P. Loucopoulos, K. Lyytinen, J. Mylopoulos, W. N. Robinson, The Brave New World of Design Requirements: Four Key Principles, in: International Conference Advanced Informations Systems Engineering, Hammaret, Tunisia, 2010, pp. 470–482. doi:10.1007/978-3-642-13094-6.
URL http://dblp.uni-trier.de/db/conf/caise/caise2010.html#JarkeLLMR10

[3] S. Mcgee, D. Greer, Software Requirements Change Taxonomy : Evaluation by Case Study, in: Int. Conf. on Req. Engineering, Trento, Italy, 2011.

[4] B. Ramesh, L. Cao, R. Baskerville, Agile requirements engineering practices and challenges: an empirical study, Information Systems Journal 20 (5) (2010) 449–480. doi:10.1111/j.1365-2575.2007.00259.x.
URL http://doi.wiley.com/10.1111/j.1365-2575.2007.00259.x

[5] IEEE Software Engineering Standards Committee, IEEE Recommended Practice for Software Requirements Specifications, Tech. rep., IEEE (1998).

[6] A. van Lamsweerde, E. Letier, R. Darimont, Managing Conflicts in Goal-Driven Requirements Engineering, Trans. Soft. Eng. 24 (11) (1998) 908–926. doi:10.1109/32.730542.
URL http://dx.doi.org/10.1109/32.730542

[7] R. Sebastiani, P. Giorgini, J. Mylopoulos, Simple and Minimum-Cost Satisfiability for Goal Models, in: International Conference Advanced Informations Systems Engineering, Riga, Latvia, 2004, pp. 20–35.
URL http://dx.doi.org/10.1007/b98058

[8] K. Sullivan, P. Chalasani, S. Jha, Software Design as an Investment Activity: A Real Options Perspective, Real Options and Business Strategy: Applications to Decision Making, 1999, pp. 215–262.

[9] N. A. Ernst, A. Borgida, I. Jureta, Finding Incremental Solutions for Evolving Requirements, in: Int. Conf. on Req. Engineering, Trento, Italy, 2011.

[10] R. Wieringa, N. Maiden, N. Mead, C. Rolland, Requirements engineering paper classification and evaluation criteria: a proposal and a discussion, Requirements Engineering Journal 11 (1) (2006) 102–107.

[11] W. N. Robinson, S. D. Pawlowski, V. Volkov, Requirements interaction management, ACM Computing Surveys 35 (2) (2003) 132.
URL http://portal.acm.org/citation.cfm?id=857079

[12] S. M. Easterbrook, B. Nuseibeh, Managing inconsistencies in an evolving specification, in: Int. Conf. on Req. Engineering, York, England, 1995, pp. 48–55.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=512545

[13] D. Zowghi, V. Gervasi, On the interplay between consistency, completeness, and correctness in requirements evolution, Information and Software Technology 45 (14) (2003) 993–1009. doi:10.1016/S0950-5849(03)00100-9.
URL http://www.sciencedirect.com/science/article/B6V0B-492W03S-2/2/e78a8f88512c8927d98a3cb042b4eebb

[14] B. Nuseibeh, S. M. Easterbrook, A. Russo, Making inconsistency respectable in software development, Journal of Systems and Software 58 (2) (2001) 171–180. doi:10.1016/S0164-1212(01)00036-X.
URL http://www.sciencedirect.com/science/article/B6V0N-43RHW98-9/2/c3ddc41fdcf7e06032699f48ba18ad70

[15] H. Thimbleby, Delaying commitment, IEEE Software 5 (3) (1988) 78–86. doi:10.1109/52.2027.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=2027

[16] P. Zave, M. Jackson, Four Dark Corners of Requirements Engineering, ACM Transactions on Software Engineering and Methodology 6 (1997) 1–30.
URL http://www.cs.toronto.edu/~nernst/papers/zave97req.pdf

[17] N. Rescher, R. Manor, On inference from inconsistent premisses, Theory and Decision 1 (2) (1970) 179–217. doi:10.1007/BF00154005.
URL http://www.springerlink.com/content/g3069245573x45u6/

[18] R. Fagin, J. Ullman, M. Vardi, On the semantics of updates in databases (preliminary report), in: International Symposium on Principles of Database Systems (PODS), Atlanta, 1983, pp. 352–365.

[19] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. Savo, Inconsistency-tolerant semantics for description logics, in: Web Reasoning and Rule Systems, 2010, pp. 103–117.

[20] N. D. Belnap, A useful four-valued logic, D. Reidel Publishing Co., 1977, p. 737.

[21] L. Chung, J. Mylopoulos, B. A. Nixon, Representing and Using Nonfunctional Requirements: A Process-Oriented Approach, Trans. Soft. Eng. 18 (1992) 483–497.
URL http://dx.doi.org/10.1109/32.142871

[22] D. Zowghi, R. Offen, A logical framework for modeling and reasoning about the evolution of requirements, in: Int. Conf. on Req. Engineering, 1997, pp. 247–257.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=566875

[23] A. Ghose, Formal tools for managing inconsistency and change in RE, in: International Workshop on Software Specification and Design, 2000, pp.

171–181. doi:10.1109/IWSSD.2000.891138.
URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=891138

[24] P. Besnard, A. Hunter, Quasi-classical logic: Non-trivializable classical reasoning from inconsistent information, in: C. Froideveaux, J. Kohlas (Eds.), Symbolic and Quantitative Approaches to Uncertainty, Springer, 1995.

[25] A. Hunter, B. Nuseibeh, Managing inconsistent specifications: reasoning, analysis, and action, ACM Transactions on Software Engineering and Methodology 7 (4).
URL http://portal.acm.org/citation.cfm?id=292182.292187

[26] D. Gabbay, Labelled deductive systems: A position paper, in: Proc. Log. Colloquium, 1993.

[27] I. J. Jureta, A. Borgida, N. A. Ernst, J. Mylopoulos, Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling, in: Int. Conf. on Req. Engineering, Sydney, Australia, 2010.

[28] R. Darimont, A. van Lamsweerde, Formal refinement patterns for goal-driven requirements elaboration, in: SIGSOFT FSE, 1996.

[29] E. S. K. Yu, J. Mylopoulos, Understanding "Why" in Software Process Modelling, Analysis, and Design, in: Proc. 16th Int. Conf. Software Eng., 1994, pp. 159–168.

[30] A. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Science of Computer Programming 20 (1-2) (1993) 3–50. doi:10.1016/0167-6423(93)90021-G.
URL http://dx.doi.org/10.1016/0167-6423(93)90021-G

[31] A. Hunter, B. Nuseibeh, Analysing inconsistent specifications, in: Int. Conf. on Req. Engineering, Annapolis, Maryland, 1997, pp. 78–86. doi:10.1109/ISRE.1997.566844.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=566844

[32] A. R. Anderson, N. Belnap, Entailment: The Logic of Relevance and Necessity, Princeton University Press, 1975.

[33] J. de Kleer, An assumption-based TMS, Artificial Intelligence 28 (2) (1986) 127–162. doi:10.1016/0004-3702(86)90080-9.
URL http://dx.doi.org/10.1016/0004-3702(86)90080-9

[34] PCI Security Standards Council, PCI DSS Requirements and Security Assessment Procedures, Version 2.0, Tech. rep., PCI, Boston (Oct. 2010).
URL https://www.pcisecuritystandards.org/security_standards/documents.php?document=pci_dss_v2-0#pci_dss_v2-0

[35] R. O'Callaghan, Fixing the payment system at Alvalade XXI: a case on IT project risk management, Journal of Information Technology 22 (4) (2007) 399–409. doi:10.1057/palgrave.jit.2000116.
URL http://www.palgrave-journals.com/jit/journal/v22/n4/full/2000116a.html

[36] S. Liaskos, A. Lapouchnian, Y. Yu, E. S. Yu, J. Mylopoulos, On Goal-based Variability Acquisition and Analysis, in: Int. Conf. on Req. Engineering, Minneapolis, Minnesota, 2006.
URL http://www.cs.toronto.edu/~jaranda/draftpapers/VarInt.pdf

[37] A. van Lamsweerde, Goal-oriented requirements enginering: a roundtrip from research to practice, in: Int. Conf. on Req. Engineering, 2004, pp. 4–7.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335648

[38] R. K. Yin, Case Study Research: Design and Methods, 4th Edition, Vol. 5 of Applied social research methods series, Sage Publications, Inc, Beverly Hills, CA, 2009.

[39] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani, Formal Reasoning Techniques for Goal Models, Journal on Data Semantics 2800 (2003) 1 – 20.
URL http://www.springerlink.com/content/nfe4tm8u9vt5j12d

[40] M. Poppleton, L. Groves, Software evolution with refinement and retrenchment, in: International Workshop on Refinement of Critical Systems: Methods, Tools and Developments, Turku, Finland, 2003.