Agile Requirements Evolution via Paraconsistent Reasoning

Neil A. Ernst[1], Alexander Borgida[2], John Mylopoulos[3] and Ivan J. Jureta[4]

[1] Department of Computer Science, University of British Columbia
nernst@cs.ubc.ca
[2] Department of Computer Science, Rutgers University
borgida@cs.rutgers.edu
[3] Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento
jm@disi.unitn.it
[4] FNRS & Louvain School of Management, University of Namur
ijureta@fundp.ac.be

Abstract. Innovative companies need an agile approach for the engineering of their product 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 a combination of lightweight, agile requirements 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 in the requirements. 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, agility, requirements, evolution

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 [1], since inevitably the plan must be abandoned, or at best revised. A variety of studies and experience reports (most recently [2]) 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 models of software development which avoid premature commitment to decisions. The central tenet of these models, 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 [3].

While this lightweight approach to Requirements Engineering (RE) has become popular in many segments of industry, the IEEE standard for software requirements [4] 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. The past decade has revealed that in most cases these criteria are rarely, if ever, possible to achieve. In this paper we argue that the above shift demands a much more flexible approach to requirements modeling and analysis.

To illustrate the usefulness of deferring conflict resolution, consider the requirements fragment represented in Fig. 1. The figure represents the requirements (shown as ovals) for the business ("optimize sales") and imposed requirements from an applicable security standard (PCI-DSS, which we discuss fully in Section 4.1). The red, X-headed relation between goals "..WEP" and "4.1.1..." represents a conflict. In this case, the conflict is between the business use of the Wireless Encryption Protocol (WEP) and the security problems with WEP. Existing approaches to requirements analysis either (i) insist that the conflict is resolved before proceeding with further reasoning (e.g., KAOS [5]), or (ii) represent the conflict as tradeoffs for higher-level goals (such as those in [6]). By contrast, our approach supports two possible courses of action. Because our

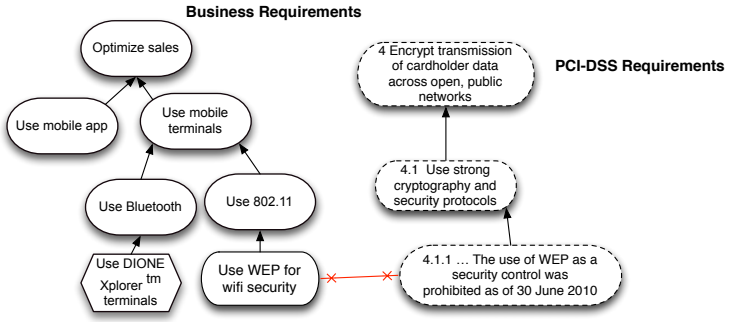

Fig. 1. Fragment of the requirements model from the payment card case. (Section 4.1)

framework is paraconsistent, it can (1) isolate and ignore the conflict for the time

being. This is useful if, for example, the need for compliance is not immediate, and time and money can be better spent elsewhere. The other course of action is to (2) proceed with one of the conflicting choices, and insist that the model be later revised to support that choice. In this case, we would ask for alternative solutions for optimizing sales. In the example, we might abandon mobile payment terminals in favour of mobile apps, or use Bluetooth rather than WEP for wireless transmission. In this way the stakeholders are presented with a range of options for proceeding.

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[1], thus helping to accommodate 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.

This paper makes the following contributions:

- identifies 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 solutions to possibly inconsistent requirements problems in a goal-oriented framework;
- explicitly introduces paraconsistent reasoning into a prototype tool;
- evaluates the framework with an industrial case study.

In [7], we introduced the notion of a Requirements Engineering Knowledge Base REKB for maintaining a requirements model. In this paper, we build on the notion of an REKB to handle the case where problems are changing and possibly inconsistent (i.e., contain contradictory assertions).

2 Background

2.1 The Requirements Problem

We start from Zave & Jackson's [8] definition of the *requirements problem*: 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.

In our work on the Techne language [9, 7], requirements problems are structured, representing notions ranging from high-level requirements ("sell more products") to low-level tasks ("use Moneris payment terminals"). A key part of solving requirements problems is therefore to find ways to refine requirements so they are eventually reduced into tasks, and to record conflicts between requirements. We therefore re-state the requirements problem as the search for tasks T and refinements/realizations/constraints R that can be added to the

[1] In other words, local inconsistencies are not propagated globally, "polluting" all inferences, as in standard logic.

world knowledge/domain assumptions D, such that requirements, captured as goals G, are satisfied, i.e.,

$$D \cup T \cup R \vdash G \tag{1}$$

Techne proposes to identify all *candidate solutions* to the problem — all sets T which achieve the goals.[2]

Techne's REKB accepts the following as well-formed formulae:

$$formula ::= atom \mid (\bigwedge_{i=1}^{n} atom_i) \rightarrow atom \mid (\bigwedge_{i=1}^{n} atom_i) \rightarrow \bot \tag{2}$$

where atoms are propositions (un-analyzed natural language sentences). Implications of the form $\beta \rightarrow \bot$ represent conflicts between the atoms in β, while those of the form $\beta \rightarrow b$ encode refinements/realizations (where b is an atom).

2.2 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 classic, inconsistent REKB useless for solving requirements problems, and is the reason why the premise of the sequent $\vdash$ in (1) must be consistent.

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

In the requirements engineering research community, the term "conflict" has typically been used to denote social disagreement over the nature of the system requirements. Robinson et al. [10] 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" [11]. Zowghi and Gervasi [12] show that "consistency" is causally related to requirements "completeness": a more complete requirements document is often less consistent (since more competing requirements are introduced). In Techne, 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., model merging). In this paper, conflict is the presence of $\bot$, while inconsistency is when false ($\bot$) is derived. Ultimately, we not only want to permit conflict (and possibly inconsistency) to be represented; we also want to specify what we ought to do when inconsistency is detected.

[2] The second step, of selecting among the candidates a solution using some techniques to rank alternatives, is not discussed in this paper.

Classical languages, such as propositional logic and first-order logic, cannot tolerate inconsistency, in the sense that no useful reasoning can be done in its presence, and yet, in the requirements engineering domain, tolerating inconsistency is important. Nuseibeh *et al* [13] give a few important reasons:

- to facilitate distributed collaborative working,
- to prevent premature commitment to design decisions,
- to ensure all stakeholder views are taken into account,
- 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 [14], 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 REKB involves *paraconsistent reasoning*. A paraconsistent logic, broadly, is one which does not trivialize in the presence of inconsistency. Section 3.2 will show how we define operators on the knowledge base that continue to give meaningful answers even when inconsistency is present.

3 RE-KOMBINE

We now define a framework for managing the inconsistency in requirements problems that arise due to variability and evolution. We will do so by defining below two fundamental operators that can be applied to an REKB in order to find solutions to the requirements problem, even in the presence of inconsistency. But before doing so, we need some formal machinery and discussion for its motivation.

3.1 Paraconsistent Reasoning on Requirements Problems

We first present a general approach to defining what it means to draw conclusions from a possibly inconsistent set of assumptions, denoted by the $\mid\!\sim$ symbol. To define $\mid\!\sim$, we go back to one of the early attempts to deal with paraconsistent reasoning, that of Rescher and Manor [15]. Given a theory Δ, define $\mathsf{MC}(\Delta)$ as the set of maximal consistent subsets of Δ, and then consider "weak" (a.k.a. "credulous") consequences those that follow from one element of $\mathsf{MC}(\Delta)$, while "inevitable" (a.k.a. "cautious, skeptical") ones hold in all such maximal subsets.

We propose the following definition

Definition 1. *$\Delta \mid\!\sim S$ iff there exists $\Pi \subseteq \Delta$ such that*

1. *$\Pi \in \mathsf{MC}(\Delta)$,*
2. *Π contains all implications in Δ, (written Implications(Δ)),*
3. *$\Pi \vdash S$*

Given domain theory D and a specific set of (high-level) goals G_0 which we are trying to achieve, a solution to the requirements problem will then be said to consist of a set of refinements/conflicts R_0 and a set of task atoms T_0 such that

$$D \cup R_0 \cup T_0 \ \mid\!\sim \ G_0 \tag{3}$$

which replaces the original equation (1).

The motivation for condition *2.* in Definition 1, which is the one addition to the original "weak" entailment in [15], is specific to Requirements Engineering, particularly the Techne family of languages and their methodology: 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. Moreover, a set of Horn clauses is always satisfiable/consistent, so requiring condition *2.* does not affect the existence of maximal consistent subsets.

The above definition is "credulous" since it depends only on the existence of *some* set Π, from which S can be derived. In contrast, much of non-monotonic reasoning and database reasoning with inconsistent data deals with the "skeptical" mode: S must be derivable from *all* maximally consistent Π of the above form. This distinction is less significant in requirements problems: since we are considering possible future states of the world, we are making assumptions about which tasks to implement. In the paraconsistent case, we are identifying individual sets of tasks which solve the requirements problem. Thus, the skeptical approach is overly constraining, since the presence of a single solution is all we need for the implementation phase. Implicitly, what the sceptical semantics for paraconsistency in requirements engineering do is restrict the nature of the eventual system we build. In RE-KOMBINE, the only constraint imposed is that implications may not be discarded.

It remains to consider whether we want to be able to obtain solutions under all conditions, or whether there are additional criteria for solutions to make sense. Consider the following criteria:

$$D \cup R_0 \nvdash \bot \tag{4}$$

$$D \cup R_0 \cup G_0 \nvdash \bot \tag{5}$$

$$D \cup R_0 \cup T_0 \nvdash \bot \tag{6}$$

Violating criterion (4) indicates the presence of something we call 'blockers': since domain assumptions hold, then no "reasonable" solution can ever exist if criterion (4) does not hold. We also want to insist that the goals we are trying to achieve (i.e., $\bigwedge G_0$) are mutually consistent in view of the background theory, criterion (5). Finally, we expect that not only can we achieve those goals, but that there are consistent sets of tasks which will do so, criterion (6).

In order to achieve the above, in RE-KOMBINE we restrict the asserted members of the knowledge base theory to be elements of D and R. Additional formulas that are implications can be added to R and D. We must include domain assumptions because, at least for now, such atoms are universally true, and thus always relevant. We include refinements and conflicts because they form the set

R, mentioned above, and presumably reflect some domain knowledge about how requirements interact.[3]

In some requirements problems we may wish to speculate about certain low-level goals being true or tasks having to be carried out; e.g., "suppose goal g_1 were achieved; what else is necessary to achieve top-level goal g_0?". In that case we may *"hypothetically"* assert these atoms by including them in R ("r: goal g_1 is achieved"), which makes it appear that we *assume* that the corresponding goal/task has been achieved.

The operator specifications below follow the above discussion, by alerting the user if condition 5 (and hence 4) is violated. Condition 6 will not be violated by virtue of Definition 1, and equation (3).

3.2 Operators for Paraconsistent Requirements Problems

RE-KOMBINE is defined in a functional style, specifying update and query operators on the requirements knowledge base (REKB). In [7] we introduced operators for consistent requirements problems, but the need for consistency ruled out the flexible approach we describe in this work. In particular, the crucial operators which help users select and decide on solutions to their requirements problems are specified using paraconsistent consequence ($\mid\!\sim$) introduced above. If the arguments to the operation are themselves (internally) inconsistent, the reasoner will generate an exception. We describe the operators in the style of Javadoc by naming the parameters and their types, etc. We use $\wp(S)$ to represent the set of all subsets of S (powerset). Examples from a case study are shown in Section 4.

Operation 1 — PARACONSIST-MIN-GOAL-ACHIEVEMENT

@param wantedG : $\wp$(GOALS)
@return TaskSets : $\wp(\wp(\text{TASKS}))$ consisting of all sets S of tasks such that:
@effect REKB $\cup$ $S \mid\!\sim \bigwedge$ wantedG, and no subset of S has this property.
@throws exception if wantedG $\cup$ Implications(REKB) $\vdash \bot$.

The PARACONSIST-MIN-GOAL-ACHIEVEMENT operation supports what has been called "backward reasoning" in the RE literature (e.g., [6]). Backward reasoning sets some high-level goals as desiderata, and determines which tasks can accomplish those goals. PARACONSIST-MIN-GOAL-ACHIEVEMENT is an abductive search. Abduction only works from consistent theories, so the classical version of this operation generates an exception if the theory REKB is inconsistent. Since Implications(REKB) is always consistent, the above modified version excludes aspects of REKB that may be inconsistent *on their own* – hence the paraconsistent behaviour of our operator. Note that the minimality of S in the above specification also prevents $\mid\!\sim$ from choosing S that have conflicting tasks, and gives preference to tasks specified as obligatorily occurring in REKB.

[3] Although arguably such refinements are not domain knowledge, but also part of the desired state of affairs. However, for the purposes of this paper we consider them assumptions.

Example. Consider the requirements problem which can be represented as $\mathbf{R} = \{D \wedge E \rightarrow B, F \wedge H \rightarrow C, C \rightarrow A, B \rightarrow A, E \wedge F \rightarrow \bot\}, \mathbf{D} = \{E, F\}$. If we then define $\mathbf{G} = \mathsf{wantedG} = \{A\}$, the problem is *classically* inconsistent, since there is a conflict when we identify potential solution sets which must contain the domain assumptions E and F. Paraconsistently, however, we can identify two separate answers to the operation: $\mathbf{S_1} = \{D, E\}, \mathbf{S_2} = \{F, H\}$. This supports our desire to continuing to reason despite a conflict.

In the requirements problem, we are interested in optimality with respect to the people communicating the requirements for the new system (stakeholders). In that context, the stakeholder may not be content with a subset-minimal implementation that satisfies the mandatory requirements (as returned by PARACONSIST-MIN-GOAL-ACHIEVEMENT). Rather, he or she is interested in implementations which also satisfy other, non-mandatory 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 2 — PARACONSIST-GET-CANDIDATE-SOLUTIONS

@param mandG : $\wp$(GOALS)

@param wishG : $\wp$(GOALS)

@return set of pairs $\langle \mathsf{solnT}, \mathsf{satG} \rangle$, where solnT is a set of tasks, and satG is a set of goals such that

@effect 1) REKB $\cup$ solnT $\mid\sim$ satG; 2) satG = mandG (the mandatory goals) $\cup$ wishG_0 (a subset of the optional goals wishG), such that satG $\cup$ Implications(REKB) 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 mandG is inconsistent with Implications(REKB).

Operation PARACONSIST-GET-CANDIDATE-SOLUTIONS requires that the set solnT paraconsistently derive satG, i.e., there is a consistent subset of REKB, which includes implications, from which one can classically prove satG, using solnT.

The above operators illustrate how paraconsistency is a fairly natural concept in solving the requirements problem. Our focus remains on the appropriate sets to return, but we adopt a credulous approach in which a single consistent subset of the larger requirements problem can be used to derive our mandatory requirements. This model of operation also maps nicely to our choice of the ATMS for implementation, as we discuss in the following subsection.

Note that there are more operators than the ones we described here, although we believe these are the two most relevant to variability in requirements. In particular, operators to calculate so-called "forward-reasoning" [6], using input tasks and a set of high-level goals, are useful (and more tractable).

3.3 Tool-supported RE-KOMBINE

We have implemented the RE-KOMBINE framework using an Assumption-based Truth Maintenance System (ATMS) [16]. An ATMS naturally supports our simple definition of well-formed formulae, and support paraconsistent reasoning. However, other choices are possible, including the use of weighted SATisfiability solvers, as used in Sebastiani et al.[6].

De Kleer [16] proposed the ATMS. In an ATMS each node has associated a set of possible *explanations*, in which that node is `:IN` (interpreted as true). Explanations 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 a given node N will typically be labelled with explanations: all sets of assumptions for which it can be derived :IN.

Most importantly, these sets are minimal – no nodes can be removed from such an explanation without losing the full justifications, and the sets are consistent in the sense that no contradictions ($\bot$) can be derived from them. We encode atoms in RE-KOMBINE as ATMS nodes; atoms with sort TASK become assumptions, and implications or contradictions become justifications and contradictions, respectively, with a special CONTRADICTION node added as necessary. Cycles are supported in the ATMS because of short-circuit evaluation. If a node has a label which is the same as a possible new label, evaluation terminates.

Listing 1 gives an example of the domain-specific language (DSL) for capturing requirements problems in Techne. The operation declare-atomic introduces (but does not assert) goals in the model, while assert-formula defines inference or conflict relations, in this case between g0 and its antecedents. The tool supports a graphical front-end (as seen in the images below) with a translation to the DSL, and 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). RE-KOMBINE, while clearly more onerous in terms of ease of use, makes the tradeoff that this flexibility nonetheless needs longer-term support, particularly in more complex domains.

We are currently integrating RE-KOMBINE with a commercial requirements tool, 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 analyst 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.

```
(defvar
g0 (declare-atomic nil "Comply with PCI DSS" :GOAL *rekb*)
; ...
(assert-formula gc1.2.1.2.1 (list g0) :DA *rekb*)
```

Listing 1: DSL for introducing Techne formulae into RE-KOMBINE.

Another approach is using Qualitative Goal Models, introduced in [17]. They support qualitative (strong, weak) evidence both in favor and against propositional goals. The solution of Giorgini et al. [18] is to formalize this by replacing each proposition g, standing for a goal, by four propositions (FS_g, PS_g, PD_g, FD_g) representing that there is full (resp. partial) evidence for the satisfaction (resp. denial) of g. A traditional implication such as $p \wedge q \rightarrow r$ is then translated into a series of implications connecting these new symbols, including $FS_p \wedge FS_q \rightarrow FS_r$, $PS_p \wedge PS_q \rightarrow PS_r$, as well as $FD_p \rightarrow FD_r$, $FD_q \rightarrow FD_r$, etc. The important point is that, first, the result is a classical propositional theory, but one where there is *never any inconsistency* that would cause everything to be inferred, since conflicts between a and b are recorded by axioms of the form $FS_a \rightarrow FD_b$, and it is quite consistent to have both FS_x and FD_x be true at the same time (there is strong evidence both for and against x). In fact, the above can be viewed as a many-valued logic, where symbol g can be assigned a *subset* of the truth values $\{FS, PS, PD, FD\}$.

Giorgini et al.'s approach is extended by Sebastiani et al. [6], including axioms for avoiding conflicts of the form $FS_a \wedge FD_a$. If we represent (for example) the Techne conflict relation $A \wedge B \rightarrow \perp$ as their $(FS_A \rightarrow FD_B) \wedge (FS_B \rightarrow FD_A)$, this supports the detection of conflicts of the form "the solution may not contain both goal A and goal B together". By using a MinWeight SAT solver which can minimize the number of asserted atoms involved in the solution, and a model which avoids the use of partial contributions, one can simulate portions of the RE-KOMBINE framework. We use this tool to compare with the ATMS implementation in Section 4.2.

4 Evaluating RE-KOMBINE

We begin with a description of our case study of variability and evolution in requirements found in the Data Security Standard (henceforth PCI-DSS) [19], an industry standard which regulates security of credit card transactions. We use this case study to first illustrate how paraconsistency is essential for solving the requirements problem in a specific example. We then discuss how our framework scales to industrially relevant sizes.

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.

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. 2. We showed that the mere presence of the conflict relation 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 $D \cup R \cup T \mid\sim G$. The state of the REKB, that is, the nature of the requirements problem of Fig. 2, is as follows.

Set R contains all implications and conflicts, among others the refinement of Optimize Sales by Use Mobile App and the conflict between Use WEP and WEP Prohibited. Set G contains the twin goals of Optimize Sales and Encrypt Transmission, since in this case study the business would like to achieve both compliance and business success. There is only a single task identified in T, the use of Dione XPlorer terminals. For this fragment there are no domain assumptions in D. Finally, we add to R the assumptions that goal Use 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., that goal WEP Prohibited is also satisfied. 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-ACHIEVEMENT with the argument wantedG = {Optimize Sales, Encrypt Transmission}. RE-KOMBINE performs an abductive search to find minimal sets of tasks or assumptions which will satisfy the conjunction of these goals. In this case, possible answers, returned as TaskSets, include {Use Mobile App, WEP Prohibited}, {WEP Prohib-

ited,DioneXplorer}. Solutions which include both 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 one of these strategies instead. Again, paraconsistency in RE-KOMBINE allows us to maintain the complete model, adding or retracting tasks as required.

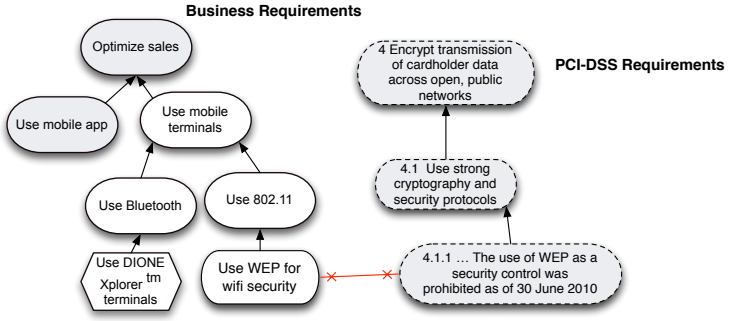

Fig. 2. WEP fragment of the model from the payment card case showing a possible configuration. Grey indicates satisfied goals.

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). 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. 3 captures this fragment of the requirements problem. Assume there is a call to PARACONSIST-GET-CANDIDATE-SOLUTIONS with wishG = {Use existing hardware} and mandG = {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 R 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.

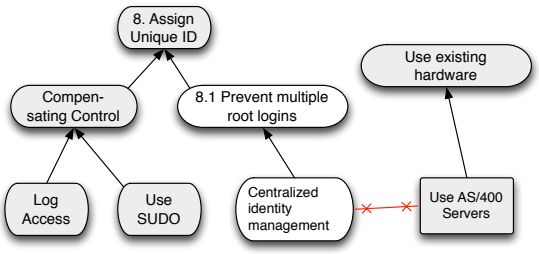

Fig. 3. The graphical representation of the compensating controls example.

4.2 Demonstrating Scalability

In [7] we showed that the ATMS reasoner was scalable. It can return decisions in less than 100 seconds on random models that were as large as six hundred requirements and two hundred relations. 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.[4] We then ran our reasoner on the PCI-DSS case study model. We also evaluated the case study using the tool from [6], as described in Section 3.3. 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 (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 [20]. 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 [21] listed KAOS model sizes that were, on average, 540 goals and requirements). We have found that the

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

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 [6], 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.

5 Related Work

We described the work of Giorgini et al. [18] and Sebastiani et al. [6] in Section 3.3. 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 satifaction, 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.

Zowghi and Offen [22] and Ghose [23] both use a default logic approach to requirements modeling. 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 a trajectory. 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. We do not insist on anticipating future change in this fashion.

Hunter and Nuseibeh [24] described one of the early approaches to inconsistent requirements specifications. They focused primarily on up-front requirements specification, in particular the possibility that different stakeholders may model the problem differently. They used labeled Quasi-Classical logic to permit the resolution of inconsistency during specification development. This work prefigures ours in using a paraconsistent language for continuing to reason in the presence of inconsistency, although different inferences are drawn. 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 [25] discuss the notions of *refinement* and *retrenchment*, which are used to model the transformation of a program as the specification changes. 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 Conclusion

This paper has operated from a premise that establishing the entirety of a project's requirements up-front is unrealistic and even undesirable. It proposed a systematic approach to agile requirements evolution where it is easy to change requirements and automatically evaluate the consequences of these changes. We showed that this reconciliation also makes it possible to delay decisions about conflicting requirements until more information becomes available. In order to support these trends, we described a framework, RE-KOMBINE, for expressing requirements formally yet sufficiently flexibly as to enable deferred commitment. The paper introduced a new definition of paraconsistency in requirements specifications using Techne as the underlying propositional language. It then described properties for defining a paraconsistent consequence operator, $\mid\!\sim$. Using that operator, we introduced 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 introduced an industrial case study of payment card requirements, and showed that the operations can be scaled to typical industrial design problems. In future, we intend to continue investigating how the approach can be used in even more complex, real-world problems.

References

1. Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J., Robinson, W.N.: The Brave New World of Design Requirements: Four Key Principles. In: International Conference Advanced Informations Systems Engineering, Hammaret, Tunisia (May 2010) 470–482
2. Mcgee, S., Greer, D.: Software Requirements Change Taxonomy : Evaluation by Case Study. In: Int. Conf. on Req. Engineering, Trento, Italy (September 2011)
3. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal **20**(5) (November 2010) 449–480
4. IEEE Software Engineering Standards Committee: IEEE Recommended Practice for Software Requirements Specifications. Technical report, IEEE (1998)

5. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20**(1-2) (April 1993) 3–50
6. Sebastiani, R., Giorgini, P., Mylopoulos, J.: Simple and Minimum-Cost Satisfiability for Goal Models. In: International Conference Advanced Informations Systems Engineering, Riga, Latvia (June 2004) 20–35
7. Ernst, N.A., Borgida, A., Jureta, I.: Finding Incremental Solutions for Evolving Requirements. In: Int. Conf. on Req. Engineering, Trento, Italy (September 2011)
8. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology **6** (1997) 1–30
9. Jureta, I.J., Borgida, A., Ernst, N.A., Mylopoulos, J.: Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling. In: Int. Conf. on Req. Engineering, Sydney, Australia (September 2010)
10. Robinson, W.N., Pawlowski, S.D., Volkov, V.: Requirements interaction management. ACM Computing Surveys **35**(2) (2003) 132
11. Easterbrook, S.M., Nuseibeh, B.: Managing inconsistencies in an evolving specification. In: Int. Conf. on Req. Engineering, York, England (1995) 48–55
12. Zowghi, D., Gervasi, V.: On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software Technology **45**(14) (November 2003) 993–1009
13. Nuseibeh, B., Easterbrook, S.M., Russo, A.: Making inconsistency respectable in software development. Journal of Systems and Software **58**(2) (2001) 171–180
14. Thimbleby, H.: Delaying commitment. IEEE Software **5**(3) (1988) 78–86
15. Rescher, N., Manor, R.: On inference from inconsistent premisses. Theory and Decision **1**(2) (December 1970) 179–217
16. de Kleer, J.: An assumption-based TMS. Artificial Intelligence **28**(2) (March 1986) 127–162
17. Chung, L., Mylopoulos, J., Nixon, B.A.: Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. Trans. Soft. Eng. **18** (1992) 483–497
18. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. Journal on Data Semantics **2800** (2003) 1 – 20
19. PCI Security Standards Council: PCI DSS Requirements and Security Assessment Procedures, Version 2.0. Technical report, PCI, Boston (October 2010)
20. O'Callaghan, R.: Fixing the payment system at Alvalade XXI: a case on IT project risk management. Journal of Information Technology **22**(4) (December 2007) 399–409
21. van Lamsweerde, A.: Goal-oriented requirements enginering: a roundtrip from research to practice. In: Int. Conf. on Req. Engineering. (2004) 4–7
22. Zowghi, D., Offen, R.: A logical framework for modeling and reasoning about the evolution of requirements. In: Int. Conf. on Req. Engineering. (1997) 247–257
23. Ghose, A.: Formal tools for managing inconsistency and change in RE. In: International Workshop on Software Specification and Design. (2000) 171–181
24. Hunter, A., Nuseibeh, B.: Managing inconsistent specifications: reasoning, analysis, and action. ACM Transactions on Software Engineering and Methodology **7**(4) (1998)
25. Poppleton, M., Groves, L.: Software evolution with refinement and retrenchment. In: International Workshop on Refinement of Critical Systems: Methods, Tools and Developments, Turku, Finland (2003)