Reasoning about Requirements Evolution using Clustered Belief Revision

O. Rodrigues[1], A. d'Avila Garcez[2], and A. Russo[3]

[1] Dept of Computer Science, King's College London, UK `odinaldo@dcs.kcl.ac.uk`
[2] Department of Computing, City University London, UK, `aag@soi.city.ac.uk`
[3] Department of Computing, Imperial College London, UK, `ar3@doc.ic.ac.uk`

Abstract. During the development of system requirements, software system specifications are often inconsistent. Inconsistencies may arise for different reasons, for example, when multiple conflicting viewpoints are embodied in the specification, or when the specification itself is at a transient stage of evolution. We argue that a formal framework for the analysis of evolving specifications should be able to *tolerate* inconsistency by allowing reasoning in the presence of inconsistency without trivialisation, and *circumvent* inconsistency by enabling impact analyses of potential changes to be carried out. This paper shows how *clustered belief revision* can help in this process.

1 Introduction

Conflicting viewpoints inevitably arise in the process of requirements analysis. Conflict resolution, however, may not necessarily happen until later in the development process. This highlights the need for requirements engineering tools that support the management of inconsistencies [12, 17].

Many formal methods of analysis and elicitation rely on classical logic as the underlying formalism. Model checking, for example, typically uses temporal operators on top of classical logic reasoning [10]. This facilitates the use of well-behaved and established proof procedures. On the other hand, it is well known that classical logic theories trivialise in the presence of inconsistency and this is clearly undesirable in the context of requirements engineering, where inconsistency often arises [6].

Paraconsistent logics [3] attempt to ameliorate the problem of theory trivialisation by weakening some of the axioms of classical logic, often at the expense of reasoning power. While appropriate for concise modelling, logics of this kind are too weak to support practical reasoning and the analysis of inconsistent specifications.

Clustered belief revision [15] takes a different view and uses theory prioritisation to obtain plausible (i.e., non trivial) conclusions from an inconsistent theory, yet exploiting the full power of classical logic reasoning. This allows the requirements engineer to analyse the results of different possible prioritisations by reasoning classically, and to evolve specifications that contain conflicting viewpoints in a principled way. The analysis of user-driven cluster prioritisations can also give stakeholders a better understanding of the impact of certain changes in the specification.

In this paper, we investigate how clustered belief revision can support requirements analysis and evolution. In particular, we have developed a tool for clustered revision that translates requirements given in the form of “if then else” rules into the (more efficient) disjunctive normal form (DNF) for classical logic reasoning and cluster prioritisation. We have then used a simplified version of the light control case study [9] to provide a sample validation of the clustered revision framework in requirements engineering.

The rest of the paper is organised as follows. In Section 2, we present the clustered revision framework. In Section 3, we apply the framework to the simplified light control case study and discuss the results. In Section 4, we discuss related work and, in Section 5, we conclude and discuss directions for future work.

2 Clustered Belief Revision

Clustered belief revision [15] is based on the main principles of the well established field of belief revision [1, 7], but has one important feature not present in the original work: the ability to group sentences with a similar *role* into a *cluster*. As in other approaches [11, 8], extra-logical information is used to help in the process of conflict resolution. Within the context of requirements evolution, such extra-logical information is a (partial) ordering relation on sentences, expressing the relative level of preference of the engineer on the requirements being formalised. In other words, less preferred requirements are the ones the engineer is prepared to give up first (as necessary) during the process of conflict resolution.

The formalism uses sentences in DNF in order to make the deduction and resolution mechanisms more efficient. The resolution extends classical deduction by using the extra-logical information to decide how to solve the conflicts. A cluster can be resolved and simplified into a single sentence in DNF. Clusters can be embedded in other clusters and priorities between clusters can be specified in the same way as priorities can be specified within a single cluster. The embedding allows for the representation of complex structures which can be useful in the specification of requirements in software engineering. The behaviour of the selection procedure in the deduction mechanism – that makes the choices in the resolution of conflicts – can be tailored according to the ordering of individual clusters and the intended local interpretation of that ordering.

Our approach has the following main characteristics: *i)* it allows users to specify clusters of sentences associated with some (possibly incomplete) priority information; *ii)* it resolves conflicts within a cluster by taking into account the priorities specified by the user and provides a consistent conclusion whenever possible; *iii)* it allows clusters to be embedded in other clusters so that complex priority structures can be specified; and finally *iv)* it combines the reasoning about the priorities with the deduction mechanism itself in an intuitive way.

In the resolution of a cluster, the main idea is to specify a deduction mechanism that reasons with the priorities and computes a *conclusion* based on these priorities. The priorities themselves are used only when conflicts arise, in which case sentences associated with higher priorities are preferred to those with lower priorities. The *prioritisation principle* (PP) used here is that "*a sentence with priority x cannot block the acceptance of another sentence with priority higher than x*". In the original AGM theory of belief revision, the prioritisation principle exists implicitly but is only applied to the new information to be incorporated.

We also adopt the *principle of minimal change* (PMC) although to a limited extent. In the original AGM theory PMC requires that old beliefs should not be given up unless this is strictly necessary in order to repair the inconsistency caused by the new belief. In our approach, we extend this idea to cope with several levels of priority by stating that "*information should not be lost unless it causes inconsistency with information conveyed by sentences with higher priority*" ($\text{PMC}_{\leq}$). As a result, when a cluster is provided without any relative priority between its sentences, the mechanism behaves in the usual way and computes a sentence whose models are logically equivalent to the models of the (union of) the maximal consistent subsets of the cluster. On the other extreme, if the sentences in the cluster are linearly prioritised, the mechanism behaves in a way similar to Nebel's *linear prioritised belief bases* [11].

Unfortunately, we do not have enough space to present the full formalism of clustered belief revision and its properties here. Further details can be found in [15]. The main idea is to associate labels of set $\mathcal{J}$ to propositional formulae via a function f and define a partial order $\leq$ on $\mathcal{J}$ according to the priorities one wants to express. $\leq$ is then extended to the power set of $\mathcal{J}$ in the following way.[4]

Definition 1. *Let* $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$ *be a cluster of sentences and* $X, Y \in 2^{\mathcal{J}}$. $X \preceq Y$ *iff either* i) $Y = \emptyset$; *or* ii) $\exists x \in X$, $\exists y \in Y$, *s.t.* $x \leq y$ *and* $X - \{x\} \preceq Y - \{y\}$; *or* iii) $\exists x \in X$, $\exists Y' \subseteq Y$, *s.t.* $Y' \neq \emptyset$ *and* $\forall y \in Y'.x < y$ *and* $X - \{x\} \preceq Y - Y'$.

The ordering above is intended to extend the user's original preference relation $\leq$ on the set of requirements to the power set of these requirements. This allows one to compare how subsets of the original requirements relate to each other with respect to the preferences stated by the user on the individual requirements. Other extensions of $\leq$ to $\mathcal{J}$ could be devised according to the needs of specific applications.

A separate mechanism selects some sets in $2^{\mathcal{J}}$ according to some criteria. For our purposes here, this mechanism calculates the sets in $2^{\mathcal{J}}$ that are associated with *consistent* combination of sentences.[5] In order to choose the *best* consistent sets (according to $\leq$), we use the ordering $\preceq$, i.e., we take the minimal elements in $2^{\mathcal{J}}$ that are consistent. Since $\preceq$ forms a lattice on $2^{\mathcal{J}}$, where $\mathcal{J}$ is always the minimum, if the labelled belief base is consistent, the choice of the best

[4] In the full formalism, the function f can map an element x of J to another cluster as well, creating nested revision levels, i.e., when the object mapped to x by f, namely $f(x)$, is not a sentence, $f(x)$ is recursively resolved first.

[5] As suggested about the extension of $\leq$, this selection procedure can be tailored to fit other requirements. One may want for instance to select amongst the subsets of $\mathcal{J}$ those that satisfy a given requirement.

consistent sets will give just $\mathcal{J}$ itself. Otherwise, this choice will identify some subsets of $\mathcal{J}$ according to $\leq$. The search for consistent combinations of sentences and minimal elements of $\preceq$ can be combined and optimised (see [14]).

Example 1. Consider the cluster $\mathcal{C}$ defined by the set $\mathcal{J} = \{x, w, y, z\}$; the partial order $\leq$ on $\mathcal{J}$ given in the middle of Figure 1, where an arrow from a to b indicates priority of a over b; and the following function f: $f(x) = p \wedge q$, $f(w) = \neg p \vee r$, $f(y) = \neg r \vee \neg s$ and $f(z) = \neg q \vee s$.

The sentences above taken conjunctively are inconsistent, so we look for consistent subsets of the base. It can be shown that the maximal consistent subsets of $\{p \wedge q, \neg p \vee r, \neg r \vee \neg s, \neg q \vee s\}$ will be those associated with the labels in the sets $\{x, w, y\}$, $\{x, y, z\}$, $\{x, w, z\}$ and $\{w, y, z\}$. According to the ordering $\preceq$, amongst these $\{x, w, y\}$ and $\{x, y, z\}$ are the ones which best verify PMC$_{\leq}$. The sets $\{w, y, z\}$ and $\{x, w, z\}$ do not verify PP. In fact, $\{w, y, z\}$ has lower priority even than $\{x\}$ since it does not contain the label x associated with the most important sentence in $\mathcal{J}$. $\{x, w, z\}$ on the other hand is strictly worse than $\{x, w, y\}$, since the latter contains y which is strictly better than z according to $\leq$. The resolution of $\mathcal{C}$ would produce a result which accepts the sentences associated with x and y and includes the consequences of the disjunction of the sentences associated with w and z. This signals that whereas it is possible to consistently accept the sentences associated with x and y, it is not possible to consistently include both the sentences associated with w and z. Not enough information is given in $\leq$ in order to make a choice between w and z and hence their disjunction is taken instead.

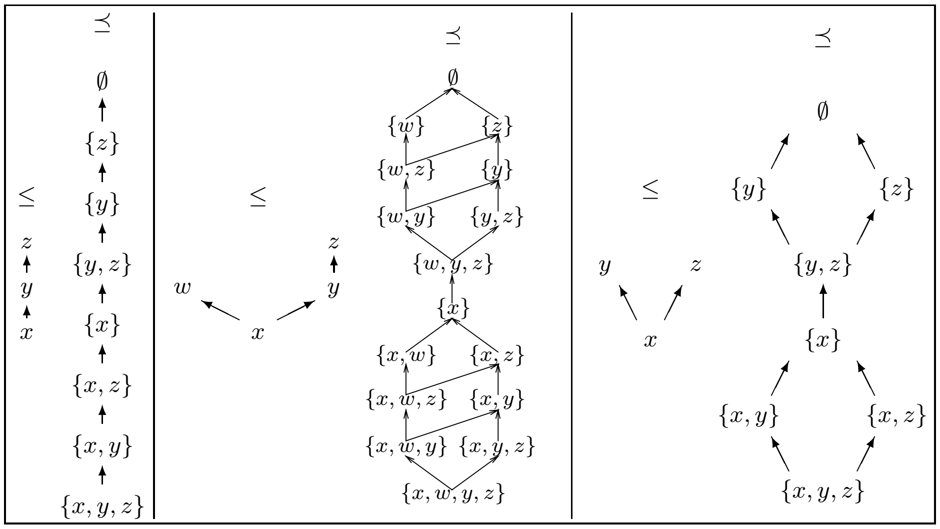

Fig. 1. Examples of orderings $\leq$ and the corresponding final ordering $\preceq$.

3 The Light Control Example

In what follows, we adapt and simplify the Light Control Case Study (LCS) [13] in order to illustrate the relevant aspects of our revision approach. LCS describes the behaviour of light settings in an office building. We consider two possible light scenes: the *default* light scene and the *chosen* light scene. Office lights are set to the default level upon entry of a user, who can then override this setting to a chosen light scene.

If an office is left unoccupied for more than t_1 minutes, the system turns the office's lights off. When an unoccupied office is reoccupied within t_2 minutes, the light scene is re-established according to its immediately previous setting. The value of t_1 is set by the facilities' manager whereas the value of t_2 is set by the office user [9]. For simplicity, our analysis does not take into account how these two times relate.

A dictionary of the symbols used in the LCS case study is given in Table 1. As usual, unprimed literals denote properties of a given state of the system, and primed literals denote properties of the state immediately after (e.g., *occ* denotes that the office is occupied at time t and occ' that the office is occupied at time $t+1$).

prop.	meaning	prop.	meaning
occ	the office is occupied	ui	a user enters an unoccupied office
uo	a user leaves an office unoccupied	e_t_i	t_i minutes have elapsed
t_unocc	unoccupied office for $t < t_2$ mins	$unocc$	unoccupied office for $t \geq t_1$ mins
us_lig	office lights as set by the user	df_lig	office lights in default setting
alm	the alarm is activated	$dark$	office lights are off
$glux_1$	day light level is greater or equal to the light level required by the chosen or default light scene (lux_1)		
$glux_2$	day light level is greater or equal to the maximum luminosity achievable by the office lights (lux_2)		

Table 1. Dictionary of symbols used in the specification.

A partial specification of the LCS is given below:

Behaviour rules

$r_1 : ui \rightarrow occ'$

$r_2 : occ \wedge uo \wedge \neg e\_t_2 \rightarrow t\_unocc'$

$r_3 : t\_unocc \wedge e\_t_1 \rightarrow unocc'$

$r_4 : t\_unocc \wedge ui \rightarrow occ'$

$r_5 : unocc \rightarrow dark'$

$r_6 : t\_unocc \wedge ui \rightarrow us\_lig'$

$r_7 : ui \rightarrow df\_lig'$

Safety rules

$s_1 : alm \wedge \neg e\_t_3 \rightarrow df\_lig'$

$s_2 : alm \wedge e\_t_3 \rightarrow dark'$

$s_3 : df\_lig \leftrightarrow \neg dark$

$s_4 : df\_lig' \leftrightarrow \neg dark'$

Economy rules

$e_1 : glux_1 \wedge (us\_lig \vee df\_lig) \rightarrow dark'$

$e_2 : glux_2 \rightarrow dark'$

We assume that LCS should satisfy two types of properties: *safety* properties and *economy* properties.

The following are safety properties: *i)* the lights are not off in the default light scene; *ii)* if the fire alarm (*alm*) is triggered, the default light scene must be re-established in all offices; and *iii)* t_3 minutes after the alarm is triggered, all lights must be turned off (i.e., only emergency lights must be on). The value of t_3 is set by the facilities manager. The above requirements are represented by rules s_1 to s_4.

The economy properties include the fact that, whenever possible, the system ought to use natural light to achieve the light levels required by the office light scenes. Sensors can check *i)* whether the luminosity coming from the outside is enough to surpass the luminosity required by the current light scene; and *ii)* whether the luminosity coming from the outside is greater than the maximum luminosity achievable by the office lights. The latter is useful because it can be applied independently of the current light scene in an office. Let lux_1 denote the luminosity required by the current light scene, and lux_2 the maximum luminosity achievable by the office lights. *i)* if the natural light is at least lux_1 ($glux_1$) and the office is in the chosen or default light scene, then the lights must be turned off; and *ii)* if the natural light is at least lux_2 ($glux_2$), then the lights must be turned off. This is represented by rules e_1 and e_2.

Now, consider the following scenario. On a bright Summer's day, John is working in his office when suddenly the fire alarm goes off. He leaves the office immediately. Once outside the building, he realises that he left his briefcase behind and decides to go back to fetch it. By the time he enters his office, more than t_3 minutes have elapsed. This situation can be formalised as follows:

i_1: John enters the office (ui), i_2: the alarm is sounding (alm)
i_3: t_3 minutes or more have elapsed since the alarm went off (e_t_3)
i_4: daylight provides luminosity enough to dispense with artificial lighting ($glux_2$)

We get inconsistency in two different ways:

1. Because John walks in the office (i_1), lights go to the default setting (r_7). By s_4, the lights must be on in this setting. This contradicts s_2, which states that lights should be turned off t_3 minutes after the alarm goes off.

   ui (i_1), alm (i_2), e_t_3 (i_3), $df\_lig' \rightarrow \neg dark'$ (s_4), $ui \rightarrow df\_lig'$ (r_7), $alm \wedge e\_t_3 \rightarrow dark'$ (s_2)
2. Similarly, as John walks in the office (i_1), lights go to the default setting (r_7). Therefore lights are turned on (s_4). However, by e_2, this is not necessary, since it is bright outside and the luminosity coming through the window is higher the maximum luminosity achievable by the office lights ($glux_2$).

   ui (i_1), $glux_2$ (i_4), $df\_lig' \rightarrow \neg dark'$ (s_4), $ui \rightarrow df\_lig'$ (r_7), $glux_2 \rightarrow dark'$ (e_2)

This is a situation where inconsistency on the light scenes occur due to violations of safety and economy properties. We need to reason about how to resolve the inconsistency. Using clustered belief revision, we can arrange the components of the specification in different priority settings, by grouping rules in clusters, e.g., a safety cluster, an economy cluster, etc. It is possible to prioritise the clusters *internally* as well, but this is not considered here for reasons of space and simplicity.

The organisation of the information in each cluster can be done independently but the overall prioritisation of the clusters at the highest level requires input from all stakeholders. For example, in the scenario described previously, we might wish to prioritise safety rules over the other rules of the specification and yet not have enough information from stakeholders to decide on the relative strength of economy rules. In this case, we would ensure that the specification satisfies the safety rules but not necessarily the economy ones.

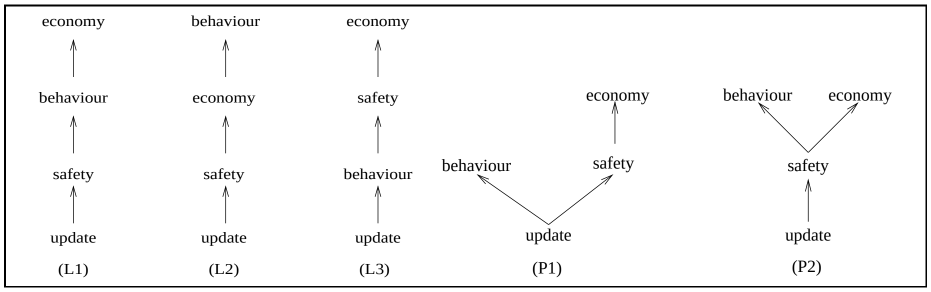

Fig. 2. Linearly (L1, L2 and L3) and partially (P1 and P2) ordered clusters.

Let us assume that sensor and factual information is correct and therefore not subject to revision. We combine this information in a cluster called "update" and give it highest priority. In addition, we assume that safety rules must have priority over economy rules. At this point, no information on the relative priority of behaviour rules is available. With this in mind, it is possible to arrange the clusters with the update, safety, behaviour and economy rules as depicted in Figure 2. Prioritisations L1, L2 and L3 represent all possible linear arrangements of these clusters with the assumptions mentioned above, whereas prioritisations P1 and P2 represent the corresponding partial ones.

The overall result of the clustered revision will be consistent as long as the cluster with the highest priority (factual and sensor information) is not itself inconsistent. When the union of the sentences in all clusters is indeed inconsistent, in order to restore consistency, some rules may have to be withdrawn. For example, take prioritisation L1. The sentences in the safety cluster are consistent with those in the update cluster; together, they conflict with behaviour rule r_7 (see Figure 3). Since r_7 is in a cluster with lower priority in L1, it cannot be consistently kept and it is withdrawn from the intermediate result. The final step is to incorporate what can be consistently accepted from the economy cluster. For example, rule e_1 is consistent with the (partial) result given in Figure 3 and is therefore included in the revised specification, and similarly for rule e_2.

update + safety include (in DNF): $ui \wedge alm \wedge e\_t_3 \wedge glux_2 \wedge dark' \wedge \neg df\_lig'$
behaviour includes (in DNF): $\neg ui \vee df\_lig'$

result 1: $ui \wedge alm \wedge e\_t_3 \wedge glux_2 \wedge dark' \wedge \neg df\_lig'$

Fig. 3. Conflict with behaviour rule r_7.

Notice however, that r_7 might be kept given a different arrangement of the priorities. The refinement process occurs by allowing one to reason about these different arrangements and the impact on the rules in the current specification without trivialising the results. Eventually, one aims to reach a final specification that is consistent regardless of the priorities between the clusters, i.e., consistent in the classical logic sense, although this is not essential in our framework.

Prioritisations L2 and P2 give the same results as L1, i.e., withdrawal of r_7 is recommended. On the other hand, in prioritisation L3, the sentence in the behaviour cluster is consistent with those in the update cluster; together, they conflict with safety rule s_4 (see Figure 4). Since the safety cluster is given lower priority in L3, both sentences s_2 and s_4 cannot be consistently kept. One has to give up either s_2 or s_4. However, if s_4 were to be kept, then e_2 would also have to be withdrawn. Minimal change to the specification forces us to keep s_2 instead, as it allows for the inclusion of e_2.

update + behaviour include (in DNF):	$ui \wedge alm \wedge e\_t_3 \wedge glux_2 \wedge df\_lig'$
safety includes (in DNF):	$((\neg df\_lig' \wedge dark') \vee (\neg df\_lig' \wedge \neg alm) \vee$
	$(\neg dark' \wedge \neg alm) \vee (\neg df\_lig' \wedge \neg e\_t_3) \vee$
	$(\neg dark' \wedge \neg e\_t_3))$

result 2: $ui \wedge alm \wedge e\_t_3 \wedge glux_2 \wedge df\_lig' \wedge dark'$

Fig. 4. Conflict with safety rule s_4.

Finally, prioritisation P1 offers a choice between the sets of clusters {update, safety, economy} and {update, behaviour, economy}. The former corresponds to withdrawing r_7 (reasoning in the same way as for L1, L2 and P2), whereas the latter corresponds to withdrawing s_4 as in the case of L3.

In summary, from the five different cluster prioritisations analysed, a recommendation was made to withdraw a behaviour rule in three of them, to withdraw a safety rule in one of them, and to withdraw either a behaviour or a safety rule in one of them. From these results and the LCS context, the withdrawal of behaviour rule r_7 seems more plausible. In more complicated cases, a decision support system could be used to help the choice of recommendations made by the clustered revision framework.

4 Related Work

A number of logic-based approaches for handling inconsistency and evolving requirements specifications have been proposed in the literature. Zowghi and Offen [18] proposed belief revision for default theories as a formal approach for resolving inconsistencies. Specifications are formalised as default theories where each requirement may be defeasible or non-defeasible, each kind assumed to be consistent within itself. Inconsistencies introduced by an evolutionary change are resolved by performing a revision operation over the entire specification. Defeasible information that is inconsistent with non-defeasible information is not used in the reasoning process and thus does not trigger a revision. Similarly, in our approach, requirements with lower priority that are inconsistent with requirements with higher priority are not considered in the computation of the revised specification. However, in our approach, the use of different levels of priority enables the engineer to fine-tune the specification and reason with different levels of defeasibility.

In [16], requirements are assumed to be defeasible, having an associated *preference ordering relation*. Conflicting defaults are resolved not by changing the specification but by considering only scenarios or models of the inconsistent specification that satisfy as much of the preferrable information as possible. Whereas Ryan's representation of priorities is similar to our own, we use classical logic entailment as opposed to Ryan's *natural entailment* and the priorities in our framework are used only in the solution of conflicts. Moreover, the use of clusters in our approach provides the formalisation of requirements with additional dimensions, enabling a more refined reasoning process about inconsistency.

In [4], a logic-based approach for reasoning about requirements specifications based on the construction of goal tree structures is proposed. Analyses of the consequences of alternative changes are carried out by investigating which goals would be satisfied and which would not, after adding or removing facts from a specification. In a similar fashion, our approach supports the evaluation of consequences of evolutionary changes by checking which requirements are lost and which are not after adding or deleting a requirement.

Moreover, other techniques have been proposed for managing inconsistency in specifications. In [2], priorities are used but only in subsets of a knowledge base which are responsible for inconsistency. Some inference mechanisms are proposed for locally handling inconsistent information using these priorities. Our approach differs from that work in that the priorities are defined independently of the inconsistency and thus facilitating a richer impact analysis on the overall specification. Furthermore, priorities there can only be specified at the same level within the base, whereas we allow for more complex representations (e.g., between and within sub-bases).

Finally, a lot of work has focused on consistency checking, analysis and action based on pre-defined inconsistency handling rules. For example, in [5], consistency checking rules are combined with pre-defined lists of possible actions, but with no policy or heuristics on how to choose among alternative actions. The entire approach relies on taking decisions based on an analysis of the history of the development process (e.g., past inconsistencies and past actions). Differently, our approach provides a formal support for analysing the impact of changes over the specification by allowing the engineer to perform *if* questions on possible changes and to check the effect that these changes would have in terms of requirements that are lost or preserved.

5 Conclusions & Future Work

In this paper, we have shown how clustered belief revision can be used to analyse the results of different prioritisations on requirements reasoning classically, and to evolve specifications that contain conflicting viewpoints in a principled way. A simplified version of the light control case study was used to provide an early validation of the framework. We believe that this approach gives the engineer more freedom to make appropriate choices on the evolution of the requirements, while at the same time offering rigourous means for evaluating the consequences that such choices have on the specification.

Our approach provides not only a technique for revising requirements specifications using priorities, but also a methodology for handling evolving requirements. The emphasis of the work is on the use of priorities for reasoning about potentially inconsistent specifications. The same technique can be used to check the consequences of a given specification and to reason about "what if" questions that arise during evolutionary changes.

A number of heuristics about the behaviour of the ordering $\preceq$ have been investigated in [14]. The use of DNF greatly simplifies the reasoning, but the conversion to DNF sometimes generates complex formulae making the reasoning process computationally more expensive. To improve scalability of the approach, these formulae should be as simple as possible. This simplification could be achieved by using Karnaugh maps to find a "minimal" DNF of a sentence.

References

1. C. A. Alchourrón and D. Makinson. On the logic of theory change: Contraction functions and their associated revision functions. *Theoria*, 48:14–37, 1982.
2. S. Benferhat and L. Garcia, *Handling Locally Stratified Inconsistent Knowledge Bases*, Studia Logica, 70:77–104, 2002.
3. N. C. A. da Costa, On the theory of inconsistent formal systems. Notre Dame Journal of Formal Logic, 15(4):497–510, 1974.
4. D. Duffy et al., A Framework for Requirements Analysis Using Automated Reasoning, CAiSE95, LNCS 932, Springer, 68–81, 1995.
5. S. Easterbrook and B. Nuseibeh, Using ViewPoints for Inconsistency Management. In Software Engineering Journal, 11(1): 31-43, BCS/IEE Press, January 1996.
6. A. Finkelstein et. al, Inconsistency handling in multi-perspective specifications, IEEE Transactions on Software Engineering, 20(8), 569-578, 1994.
7. Peter Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* The MIT Press, Cambridge, Massachusetts, London, England, 1988.
8. P. Gärdenfors and D. Makinson. Revisions of knowledge systems using epistemic entrenchment. TARK II, pages 83–95. Morgan Kaufmann, San Francisco, 1988.
9. C. Heitmeyer and R. Bharadwaj, Applying the SCR Requirements Method to the Light Control Case Study, Journal of Universal Computer Science, Vol.6(7), 2000.
10. M. R. Huth and M. D. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, 2000.
11. B Nebel. Syntax based approaches to belief revision. *Belief Revision*, 52–88, 1992.
12. B. Nuseibeh, J. Kramer and A. Finkelstein, A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, IEEE Transactions on Software Engineering, 20(10): 760-773, October 1994.
13. S. Queins et al., The Light Control Case Study: Problem Description. Journal of Universal Computer Science, Special Issue on Requirements Engineering: the Light Control Case Study, Vol.6(7), 2000.
14. Odinaldo Rodrigues. *A methodology for iterated information change.* PhD thesis, Department of Computing, Imperial College, January, 1998.
15. O. Rodrigues, Structured Clusters: A Framework to Reason with Contradictory Interests, Journal of Logic and Computation, 13(1):69–97, 2003.
16. M. D. Ryan. Default in Specification, IEEE International Symposium on Requirements Engineering (RE93), 266–272, San Diego, California, January 1993.
17. G. Spanoudakis and A. Zisman. Inconsistency Management in Software Engineering: Survey and Open Research Issues, Handbook of Softawre Engineering and Knowledge Engineering, (ed.) S.K. Chang, pp. 329-380, 2001.
18. D. Zowghi and R. Offen, A Logical Framework for Modeling and Reasoning about the Evolution of Requirements, Proc. 3rd IEEE International Symposium on Requirements Engineering RE'97, Annapolis, USA, January 1997.