Reasoning about Requirements Evolution using Clustered Belief Revision

O. Rodrigues[ω], A. d'Avila Garcez[δ], A. Russo[ρ]

[ω]Department of Computer Science, King's College London, UK, odinaldo@dcs.kcl.ac.uk

[δ]Department of Computing, City University London, UK, aag@soi.city.ac.uk

[ρ]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. These inconsistencies cannot always be resolved immediately. As a result, 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. Clustered belief revision allows for the grouping of requirements with similar functionality into clusters and the assignment of priorities between them. By analysing the result of a cluster, an engineer can either choose to rectify problems in the specification or to postpone the changes until more information becomes available.

1 Introduction

Conflicting viewpoints inevitably arise in the process of requirements elicitation. 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 [13, 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 [9]. This facilitates the use of well-behaved and established theorems and proof procedures. On the other hand, Classical Logic does not accept inconsistency, in the sense that one can derive anything from an inconsistent theory. For example, one can derive any proposition B from propositions A and $\neg A$. This is known as *theory trivialisation*, and is clearly undesirable in the context of requirements engineering, where inconsistency often arises [5, 8].

Paraconsistent Logics [2] attempt to ameliorate the problem of theory trivialisation by weakening some of the axioms of classical logic, often at the expense of reasoning power. For instance, Belnap's four valued logic [1] allows for non trivial logical representations where propositions can be both true and false, but does not verify basic inference rules such as Modus Ponens. 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. not 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 elicitation 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 [7] 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 two main ideas: the use of extra-logical information to help in the process of conflict resolution and the ability of group sentences with similar *role* into a *cluster*. In order to make the deduction mechanism more efficient, sentences are expressed in DNF. A cluster can be resolved and simplified into a single sentence in DNF. The resolution extends classical deduction by using the extra-logical information to decide how to solve the conflicts. Clusters can be embedded in other clusters and priorities between clusters can be specified in the same way that 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 its intended local interpretation. We provide one such ordering based on the confidence/priority that the user has/wants to assign to each cluster.

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 override those associated with lower priorities. The *prioritisation principle* used here is that "a sentence with priority x cannot block the acceptance of another sentence with priority higher than x". The other principle used is that of *minimal change*: information should not be lost unless it causes inconsistency with information conveyed by sentences with higher priority. As a result, when a cluster is provided with no relative priority between sentences, the mechanism 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 sentences in the base are linearly prioritised, the mechanism behaves in a way similar to Nebel's *linear prioritised belief bases*[1] [12].

Definition 1 *A* labelled belief base *(LBB) is a tuple* $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$*, where* $\mathcal{J}$ *is a set of labels,* $\leq$ *is a (partial) pre-order on* $\mathcal{J}$ *and* f *assigns elements of* $\mathcal{J}$ *to sentences of the language.*

Definition 2 *A* structured cluster *is a tuple* $\Xi = \langle \mathcal{C}, \sqsubseteq, g \rangle$ *where* $\mathcal{C}$ *is a set of labels,* $\sqsubseteq$ *is a (partial) pre-order on* $\mathcal{C}$ *and* g *is a function assigning elements of* $\mathcal{C}$ *to either a sentence; a LBB or another cluster.*

Definition 3 *The level of a propositional logic formula is* 0*. Let* $\Xi = \langle \mathcal{C}, \sqsubseteq, g \rangle$ *be a cluster. The level of* Ξ*, in symbols* $\text{level}(\Xi)$ *is defined recursively as* $\text{level}(\Xi) = max_{i \in \mathcal{C}}\{\text{level}(g(i))\} + 1$.

Thus, a cluster of level 1 is just a labelled belief base as defined previously.

Definition 4 *Let* $K = \{\varphi_1, \ldots, \varphi_k\}$ *be a finite set of sentences. A* matrix representation of K *is obtained by associating rows of the matrix with logically equivalent formulas in DNF of each sentence where the columns are the disjuncts in those sentences.* $\mathcal{M}_K$ *will denote a chosen matrix representation of* K*.*

Definition 5 *Let* $\mathcal{M}_K$ *be a matrix representation of a set* K*. A* path in $\mathcal{M}_K$ *is a set* $\wp$ *of disjuncts, each and only one taken from each row in* $\mathcal{M}_K$*. We denote the set of all paths in* $\mathcal{M}_K$ *by* $\text{paths}(\mathcal{M}_K)$*.*

Note that a given path contains exactly one *representative* disjunct of each sentence in the belief base.

Definition 6 *The conjunction of all disjuncts visited in a path* $\wp$ *will be denoted by* $\sigma(\wp)$*. If* $\wp$ *is empty, we define* $\sigma(\wp)$ *to be* $\top$ *(we assume the language has a symbol for truth).*[2]

Ultimately what we want is to compare the best combinations of sentences in a belief base verifying prioritisation and consistency. If we can keep them all consistently so much the better, but this is not always possible. In order to compare subsets of the belief base, we define an ordering $\preceq$ on $2^{\mathcal{J}}$. We use $X \preceq Y$ to denote that the satisfiability X's sentences is at least as plausible as Y's. $X \prec Y$ means $X \preceq Y$ and $Y \not\preceq X$.

Definition 7 *Let* $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$ *be a labelled belief base 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* $X - \{x\} \preceq Y - Y'$*.*

[1] In our case, a belief base is a partial specification.

[2] Notice that $\sigma(\wp)$ is simply a conjunction of literals and the basis for rebuilding formulas in DNF. In model theoretical terms, $\sigma(\wp)$ represent one possible way of satisfying a belief base.

Figure 1 gives examples of some orderings $\leq$ on $\mathcal{J}$ and the derived orderings $\preceq$ on $2^{\mathcal{J}}$. A connecting arrow from a to b indicates that $a < b$ or $a \prec b$ (i.e. that a is preferred to b).

Given a path in a matrix, we are interested in combinations of disjuncts in the path that are not contradictory (we will be especially interested in *maximal such combinations*):

Definition 8 *Let $\mathcal{M}_K$ be a matrix representation of a set K and* $\text{paths}(\mathcal{M}_K)$ *the set of all paths in $\mathcal{M}_K$. The set of* consistent subpaths of $\mathcal{M}_K$ *is defined as* $\text{paths}^{\top}(\mathcal{M}_K) = \{\xi \,|\, \exists\wp \in \text{paths}(\mathcal{M}_K) \textit{ s.t. } \xi \subseteq \wp \textit{ and } \sigma(\xi) \textit{ is not contradictory}\}$.

Definition 9 *The* label set of a path $\wp$ *is the set* $\text{ls}(\wp) = \{\alpha \mid \alpha : P \in \wp\}$. *The* label abstraction of a set of paths Λ *is the set* $\text{La}(\Lambda) = \{\text{ls}(\wp) \mid \wp \in \Lambda\}$.

Definition 10 *The* maximal plausible subpaths of a matrix representation of a set K *are the elements of the set* $\text{mps}(\mathcal{M}_K) = \{\wp \in \text{paths}^{\top}(\mathcal{M}_K) \mid \text{ls}(\wp) \textit{ is } \preceq\textit{-minimal in } \text{La}(\text{paths}^{\top}(\mathcal{M}_K))\}$.

Definition 11 *Let $\mathsf{B} = \langle \mathcal{J}, \leq, f\rangle$ be a labelled belief base and $\mathcal{M}_{\mathsf{B}}$ a matrix representation of the sentences mapped by f in B. The result of flattening out B, in symbols* $\text{FLATTEN\_BASE}(\mathsf{B})$*, is the sentence* $\bigvee_{\xi\in\text{mps}(\mathcal{M}_{\mathsf{B}})} \sigma(\xi)$.

It is also possible to flatten out a cluster of higher order by recursively flatenning out all embedded subclusters as follows (this is simply an extension to Definition 11).

Definition 12 *Let $\Xi = \langle \mathcal{C}, \sqsubseteq, g\rangle$ be a structured cluster. The result of flattening out Ξ, in symbols* $\text{FLATTEN\_CLUSTER}(\Xi)$*, is the sentence in DNF obtained in the following way:*

$$\text{FLATTEN\_CLUSTER}(\Xi) \quad = \quad \begin{cases} \text{FLATTEN\_BASE}(\Xi) & \Rightarrow \; \textit{if } \text{level}(\Xi) = 1 \\ \text{FLATTEN\_CLUSTER}(\Xi') & \Rightarrow \; \textit{otherwise} \end{cases}$$

where Ξ' is the cluster obtained from Ξ by replacing the function g by the function g', such that $g'(i) =$ $\text{FLATTEN\_CLUSTER}(g(i))$*, for all $i \in \mathcal{C}$.*

Example 1 *Consider the ordering $\leq$ in the middle of Figure 1 and assume that $f(x) = user\_in \wedge gte\_lux_2$, $f(w) = \neg user\_in \vee default\_lights'$, $f(y) = \neg default\_lights' \vee \neg no\_lights'$ and $f(z) = \neg gte\_lux_2 \vee no\_lights'$. These sentences correspond to the DNF of the sentences in the second inconsistency example discussed in the next section and the ordering $\leq$ is actually the same in prioritisation P1 there.*

The sentences taken conjunctively are inconsistent, so we would have to look for consistent subpaths in the matrix of this set. It can be shown that the consistent subpaths with highest number of elements will be associated with the labels in the sets $\{x, w, y\}$, $\{x, y, z\}$, $\{x, w, z\}$ and $\{w, y, z\}$. According to the ordering $\preceq$, the most plausible ones amongst those are $\{x, w, y\}$ and $\{x, y, z\}$. $\{w, y, z\}$ cannot be chosen as it does not contain a label of the most important sentence, namely x. $\{x, w, z\}$ is not chosen because it is strictly worse than $\{x, w, y\}$, since the latter contains y which is strictly better than z.

As a result, this ordering would produce a result which accepts sentences associated with x and y and includes the consequences of the disjunction of sentences w and z. This signals that whereas it is possible to consistently accept x and y, it is not possible to consistenly include both w and z. Given the assigned priorities, a choice between them cannot be made and their disjunction is taken instead.

3 The Light Control Example

In what follows, we adapt and simplify the Light Control Case Study [14] in order to illustrate the relevant aspects of our revision approach. The Light Control System (LCS) describes the behaviour of light settings in an office building. We consider two possible light scenes: a *default* light scene and a *chosen* light scene. Office lights are set to a 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 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 [7]. A partial specification of the LCS is as follows:

Behaviour rules

$r_1 : user\_in \rightarrow occupied'$

$r_2 : occupied \wedge user\_out \wedge \neg elapsed\_T_2 \rightarrow temp\_unocc'$

$r_3 : temp\_unocc \wedge elapsed\_T_1 \rightarrow unoccupied'$

$r_4 : temp\_unocc \wedge user\_in \rightarrow occupied'$

$r_5 : unoccupied \rightarrow no\_lights'$

$r_6 : temp\_unocc \wedge user\_in \rightarrow chosen\_lights'$

$r_7 : user\_in \rightarrow default\_lights'$

In rules r_1 to r_4, *occupied* states that a user is in the office; *user_in* indicates that a user has entered an unoccupied office; *user_out* indicates that a user has left an office unoccupied; *temp_unocc* indicates that

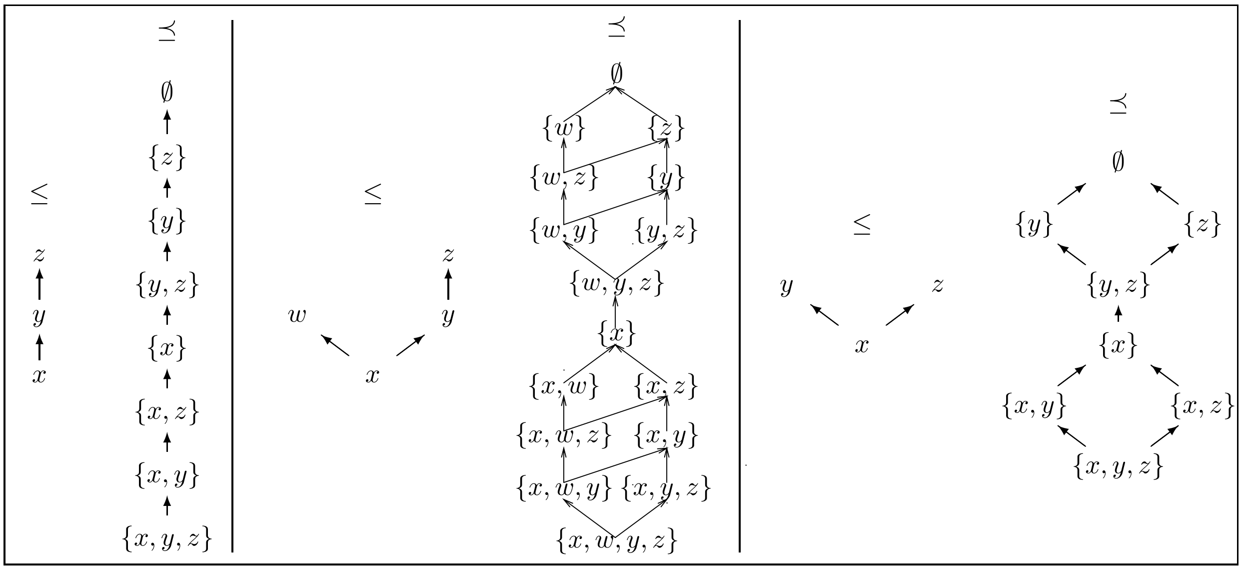

Figure 1: Examples of orderings $\leq$ on the clusters and the corresponding final ordering $\preceq$.

the office has been left unoccupied for less than T_2 minutes; *unoccupied* indicates that the office has been unoccupied for more than T_1 minutes; and $elapsed\_T_i$ ($i \in \{1, 2, 3\}$) indicates that T_i minutes have elapsed. As usual, unprimed literals denote properties of the current state of the system, and primed literals denote properties of the next state (e.g., *occupied* denotes that a user is in the office at time t, and $occupied'$ denotes that a user is in the office at time $t+1$). Rules r_5 to r_7 specify the intended behaviour of the office lights: *no_lights* indicates that the office lights are off; *chosen_lights* indicates that the office lights are as set by the user; and $default\_lights$ indicates that the office lights are in the default setting. We assume that the initial chosen light scene is set to the default one.

In our study, we consider that the light control system 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 (*alarm*) is triggered, the default light scene must be established in all offices; and *iii)* T_3 minutes after the alarm is triggered, the lights must all 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:

Safety rules

$s_1 : alarm \wedge \neg elapsed\_T_3 \rightarrow default\_lights'$ $\quad$ $s_3 : default\_lights \leftrightarrow \neg no\_lights$

$s_2 : alarm \wedge elapsed\_T_3 \rightarrow no\_lights'$ $\quad$ $s_4 :: default\_lights' \leftrightarrow \neg no\_lights'$

Economy properties include the fact that, to the extent feasible, 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 outside is enough to surpass the luminosity required by the current light scene; and *ii)* whether the luminosity coming from 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. The above can be summarised as follows: *i)* if the natural light is at least lux_1 (gte_lux_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 (gte_lux_2), then the lights must be turned off. The above properties are represented as follows:

Economy rules

$e_1 : gte\_lux_1 \wedge (chosen\_lights \vee default\_lights) \rightarrow no\_lights'$ $\quad$ $e_2 : gte\_lux_2 \rightarrow no\_lights'$

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, the alarm has been going off for more than T_3 minutes. This situation can be formalised as follows:

i_1: John enters the office (*user_in*)

i_2: The alarm is sounding (*alarm*)

i_3: T_3 minutes or more have elapsed since the alarm went off ($elapsed\_T_3$)

i_4: Day light provides luminosity enough to dispense with artificial lighting (gte_lux_2)

We get inconsistency in two different ways:

1. Because John walks in the office (i_1), the default light setting is chosen (r_7). By s_4, the lights must be on in this setting. This is a contradiction with safety rule s_2, which states that lights should be turned off T_3 minutes after the alarm goes off.
2. Similarly, when John walks in the office (i_1), the default light scene is set (r_7). This effectively forces the lights to be turned on (s_4). However, by e_2, this is not necessary since the amount of luminosity coming from outside is higher the the maximum luminosity achievable by the office lights (gte_lux_2).

We are, therefore, in a situation where inconsistency on the light scenes occur due to a safety property violation and due to an economy property violation. We need to reason about the courses of action to deal with this problem. Using clustered belief revision, we can arrange the several components of the specification in different priority settings, by grouping rules in clusters, e.g., safety cluster, economy cluster, etc. 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. Since the specification is being refined, the framework must cope with potential inconsistencies without trivialising the results. The formalism allows for arbitrary orderings *inside* the clusters as well, but this is not considered here for reasons of space and simplicity.

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 or ones.

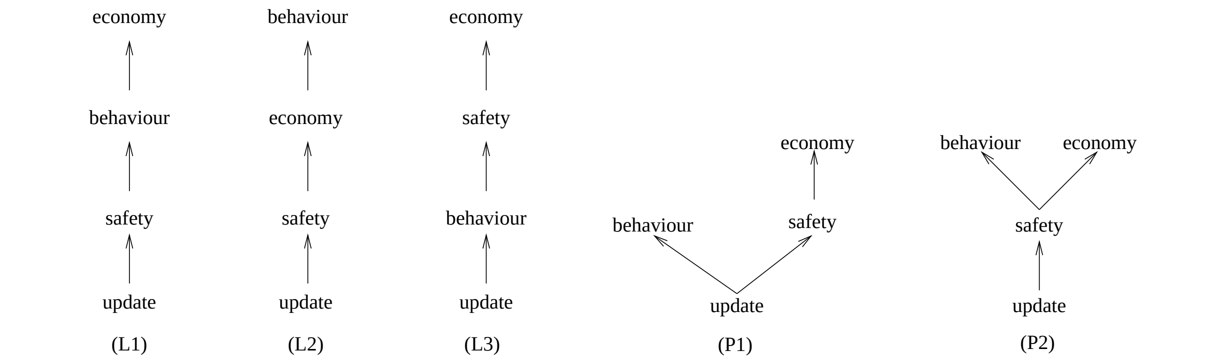

Figure 2: Linearly (L1, L2 and L3) and partially (P1 and P2) ordered clusters.

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.[3] 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. As we mentioned, each of the components economy, behaviour, safety and update could be associated with its own partial priority order as well, allowing for the expression of more complex relationships between individual properties.

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 the clusters is indeed inconsistent, in order to restore consistency, some rules may have to be withdrawn. The result will be such that rules will be kept as long as their inclusion does not cause inconsistency with other rules in a cluster with higher priority. Note that, to check whether the revised specification satisfies a rule, one needs to check for derivability of that rule from the final result.

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:

update + safety include (in DNF): $user\_in \wedge alarm \wedge elapsed\_T_3 \wedge gte\_lux_2 \wedge no\_lights' \wedge \neg default\_lights'$
behaviour includes (in DNF): $\neg user\_in \vee default\_lights'$

result 1: $user\_in \wedge alarm \wedge elapsed\_T_3 \wedge gte\_lux_2 \wedge no\_lights' \wedge \neg default\_lights'$

Since r_7 is given 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 e_2.[4]

[3] Recall that a connecting arrow between clusters indicates priority of the source cluster over the target one.
[4] e_1 is also implicitly incorporated since we can neither prove the antecendent nor the negation of the consequent.

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 of 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. 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:

$$\begin{array}{ll}
\text{update + behaviour include (in DNF):} & user_in \wedge alarm \wedge elapsed_T_3 \wedge gte_lux_2 \wedge default_lights' \\
\text{safety includes (in DNF):} & ((\neg default_lights' \wedge no_lights') \vee (\neg default_lights' \wedge \neg alarm) \vee \\
& (\neg no_lights' \wedge \neg alarm) \vee (\neg default_lights' \wedge \neg elapsed_T_3) \vee \\
& (\neg no_lights' \wedge \neg elapsed_T_3)) \\
\hline
\multicolumn{2}{c}{\text{result 2: } user_in \wedge alarm \wedge elapsed_T_3 \wedge gte_lux_2 \wedge default_lights' \wedge no_lights'}
\end{array}$$

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 be required to be withdrawn. The only way to cause minimal change to the specification is therefore to keep s_2 instead, since it allows the inclusion of e_2.

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, while the latter corresponds to withdrawing s_4 as in the case of L3. It is not possible to make a choice based on the available priority information and hence the disjunction of results 1 and 2 above is taken.

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 type is assumed to be consistent. Inconsistencies introduced by an evolutionary change are resolved by performing a revision operation over the entire specification. Change actions for handling inconsistency are implicitly given by the definition of such a belief revision operator, which changes the status of information from defeasible to non-defeasible and vice-versa to remove the inconsistent. Non-defeasible information that is inconsistent with defeasible information is not taken into consideration during the reasoning process (thus avoiding trivialisation). 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 preferable information as possible. Whereas Ryan's preference relation is similar to our priority relation, the use of clusters in our approach provides the formalisation of the requirements with addtional dinemsions, which enables a more refined reasoning process about the inconsistencies.

In [3], 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. Priority plays an important role in this process as the analysis could be focused on those requirements that have the highest priority only.

Finally, many other techniques have been proposed in the literature on managing inconsistency, but much of this work has focused on consistency checking, analysis and action based on pre- defined inconsistency handling rules. For example, in [4], 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 and Future Work

In this paper, we have shown how clustered belief revision can be used to analyse the results of different specification prioritisations reasoning classically, and to evolve specifications that contain conflicting viewpoints in a principled way. We developed a tool for clustered revision and used a simplified version of the light control case study to provide an early validation of the tool. We believe that this approach provides the engineer with 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.

As future work, we intend to apply different Machine Learning [11] techniques to revise requirements specifications [6, 10]. Consider, for instance, the Light Control example of Section 3. Assume that a person in a particular office needs to have a light scene that violates the economy properties of the specification. This is a *scenario* which, in terms of Machine Learning, can be seen as an example to be learned. This example, when trained, may evolve the specification into a consistent new specification. In fact, Machine Learning techniques may add new concepts to the specification, according to the scenarios available for training. Differently from Belief Revision, though, Machine Learning techniques do not normally guarantee consistency of the new specification, nor that the principle of Minimal Change is satisfied. A comparative analysis of these two methods of theory revision in the context of requirements evolution would be highly desirable.

References

[1] N. D. Belnap, A useful four-valued logic. Modern Uses of Multiple-Valued Logic, eds. G. Epstein and J. M. Dunn, Reidel Publishing Company, pp. 7-37, 1977.

[2] N. C. A. da Costa, On the theory of inconsistent formal systems. Notre Dame Journal of Formal Logic, 15(4):497–510, 1974.

[3] D. Duffy, C. MacNish, J. McDermid and P. Morris, A Framework for Requirements Analysis Using Automated Reasoning, Proceedings of CAiSE95, LNCS 932, Springer, 68-81, 1995.

[4] S. Easterbrook and B. Nuseibeh, Using ViewPoints for Inconsistency Management. In Software Engineering Journal, 11(1): 31-43, BCS/IEE Press, January 1996.

[5] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh, Inconsistency handling in multi-perspective specifications, IEEE Transactions on Software Engineering, 20(8), 569-578, 1994.

[6] A. S. d'Avila Garcez, A. Russo, B. Nuseibeh and J. Kramer. Combining Abductive Reasoning and Inductive Learning to Evolve Requirements Specifications. IEE Proceedings - Software 150(1):25-38, 2003.

[7] C. Heitmeyer and R. Bharadwaj, Applying the SCR Requirements Method to the Light Control Case Study, Journal of Universal Computer Science, Special Issue on Requirements Engineering: the Light Control Case Study, Vol.6(7), 2000.

[8] D. Gabbay and A. Hunter, Making inconsistency respectable 1: A logical framework for inconsistency in reasoning, Foundations of Artificial Intelligence Research, eds. Ph. Jorrand and J. Kelemen, LNCS 535, pp. 19-32, Springer, 1991.

[9] M. R. Huth and M. D. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. 387 pages. Cambridge University Press, 2000.

[10] A. van Lamsweerde and L. Willemet, Inferring Declarative Requirements Specifications from Operational Scenarios, IEEE Transactions on Software Engineering, Special Issue on Scenario Management, 1998.

[11] T. Mitchell, Machine Learning, McGraw Hill, 1997.

[12] B. Nebel. Syntax based approaches to belief revision. *Belief Revision*, pages 52–88, 1992.

[13] 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.

[14] 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.

[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 Proceedings of 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.