(to appear in) Proc. of 3rd Int. Symposium on Requirements Engineering (RE97), Annapolis, USA, Jan 97, IEEE CS Press.

Analysing Inconsistent Specifications

Anthony Hunter Bashar Nuseibeh

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
Email: (abh, ban}@doc.ic.ac.uk

Abstract

In previous work we advocated continued development of specifications in the presence of inconsistency. To support this we presented quasi-classical (QC) logic for reasoning with inconsistent specifications. The logic allows the derivation of non-trivial classical inferences from inconsistent information. In this paper we present a development called labelled QC logic, and some associated analysis tools, that allows the tracking and diagnosis of inconsistent information. The results of analysis are then used to guide further development in the presence of inconsistency. We illustrate the logic and our tools by specifying and analysing parts of the London Ambulance Service. We argue that the scalability of our approach is made possible by deploying the ViewPoints framework for multi-perspective development, such that our analysis tools are only used on partial specifications of a manageable size.

1. Motivation and Background

Inconsistent specifications are an inevitable intermediate product of a requirements engineering process. Inconsistencies may arise because of the preliminary nature of elicited requirements, or indeed because of inherently conflicting customer needs (for example, because of multiple, conflicting views that these customers hold on a problem or solution domain).

A desirable product of a requirements engineering process is a formal specification which captures customer requirements. The formality of such a specification is desirable because it is amenable to formal reasoning and analysis which, in turn, also facilitate the validation of customer requirements. The process of translating informal (often vague and inconsistent) requirements statements into a precise formal (consistent) specification is a difficult one. We believe that tools which enable reasoning and analysis of inconsistent, but formal, specifications can help improve such a translation process.

In this paper, we present some formal, light-weight, logic-based tools that can provide a requirements engineer with a handle on inconsistencies in specifications. The aim of these tools is to provide additional "non-intrusive" reasoning in the presence of inconsistency and simple analysis of inconsistent information. Such reasoning and analysis can in turn provide *guidance* to the requirements engineer on what course of *action* to take in the presence of certain inconsistencies (we still believe, however, that such action is ultimately a human-driven process). The "non-intrusive" operation of such tools is necessary because in many instances, the logic-based tools are computationally intensive and would otherwise render automated tool support unusable.

The work presented here complements our previous work on eliciting requirements from multiple perspectives using "ViewPoints" [11, 12, 25], and develops our approach of inconsistency handling in this setting [16]. Our motivation is that inconsistencies are inevitable in software development (and requirements engineering) processes and products. They provide a focus for further development (e.g., requirements elicitation), and can be regarded as "desirable" in that they highlight issues that need further attention. As such, they should be tolerated, analysed and acted upon - in other words, systematically *managed* [23].

The focus of this paper is a logic-based approach to managing inconsistent specifications. In particular, we focus on an adaptation of classical logic (termed quasi-classical, or QC, logic) that allows limited reasoning in the presence of inconsistency (§3), and extend it in simple ways that facilitate the analysis of inconsistent specifications (§4). We then develop an example to illustrate our logical tools (§5) based on the IWSSD-8 case study of the "London Ambulance Service" [15], and discuss the impact of the kind of analysis we advocate on our goal of inconsistency handling and management. We conclude with a short discussion on the role of automated tool support, and related and future work (§6 and §7). A more detailed discussion is available in [20].

2. Requirements: From fuzzy to formal

The requirements of many large software systems are characterised by imprecision. Customers often under- or over-specify their requirements, and requirements statements are often contradictory. However, in order to elicit customer requirements effectively it is essential that the needs of *all* stakeholders are captured. To this end we have used the ViewPoints framework for multi-perspective development to explicitly represent different stakeholder requirements [25].

In moving towards a precise specification that we can validate and then satisfy, there also is the need for some formal reasoning and analysis. We have found classical logic to be an appealing form of formal representation because it allows the capture of a wide range of development information, and has an existing body of tools and technology for analysis and reasoning; e.g., [1], [6]. Unfortunately, a large body of requirements information, elicited during the early part of the requirements engineering cycle, is inconsistent, and therefore there is a need to reason with inconsistent information. Classical logic, however, is *trivialised* in the presence of inconsistency; that is, by the definition of the logic, any inference follows from inconsistent information (*ex falso quodlibet*). Formally,

$$\{ \alpha, \neg\alpha \} \vdash \beta$$

To address this problem, we developed a quasi-classical (QC) logic that allows non-trivial reasoning in the presence of inconsistency.

3. QC Logic: Reasoning in the presence of inconsistency

The full formal definition of QC Logic may be found in [4]. Here we provide an informal presentation of the logic and illustrate its reasoning capabilities with a simple example[1].

The proof theory of QC logic is based on reasoning with formulae that are in conjunctive normal form (CNF). These are formulae of the following form:

$$\alpha_1 \wedge \ldots \wedge \alpha_n$$

where each α_i is of the form:

$$\beta_1 \vee \ldots \vee \beta_m$$

and each β_j is a literal.

The proof theory of QC logic (see appendix) provides the power to derive a CNF of any formula, together with the power of resolution:

$$\frac{\neg\alpha \vee \beta \qquad \alpha \vee \gamma}{\beta \vee \gamma}$$

Only as a last step in any derivation is *disjunction introduction* allowed. This means that any resolvant of a set of formulae can be derived, but no trivial formulae can be derived. The proof theory is presented as a set of natural deduction rules such as:

$$\frac{\alpha}{\alpha \vee \gamma} \quad \textit{[disjunction introduction]}$$

All the QC natural deduction rules hold in classical logic, but the logic is *weaker* than classical logic in the way it is *used*. QC logic is used by providing any set of classical formulae as assumptions, and any classical formula as a query. The query follows from the assumptions if and only if there is a derivation of a CNF of the query from the assumptions using the QC natural deduction rules.

To illustrate the use of QC in the context of the ViewPoints framework, consider the following simple example. Suppose we have two partial specifications VP1 and VP2 (representing two stakeholder ViewPoints). In VP1, there is the association:

has-exactly-one(Ambulance, Operator)

and in VP2 there is the association:

has-exactly-two(Ambulance, Operator)

If we also have the constraint:

$\forall$X,Y, has-exactly-one(X,Y) $\leftrightarrow$ $\neg$has-exactly-two(X,Y)

then this constraint together with VP1 and VP2 are inconsistent. However, if we also use the following additional "domain knowledge", discovered perhaps during development, or the result of agreements between developers:

$\forall$X, Y, has-exactly-one(X,Y) $\rightarrow$ has-one-or-more(X,Y)

$\forall$X, Y, has-exactly-two(X,Y) $\rightarrow$ has-one-or-more(X,Y)

then for both VP1 and VP2, and despite the inconsistency between them, we can still derive the potentially useful non-trivial inference:

has-one-or-more(Ambulance, Operator).

Reasoning such as in the above example is potentially useful for a range of activities in the management of inconsistencies. It allows us to go beyond *having* to remove the inconsistencies from our specifications, and then it allows us to perform the kind of analysis of inconsistent information described in the next section.

4. Logical Analysis of Inconsistent Specifications

Our analysis aims at facilitating the *tracking* and *diagnosis* of inconsistencies in order to handle the consequences of tolerating (and possibly propagating) such inconsistencies in our specifications. We present two kinds of analysis both of which are independent of the QC language itself, but which obviously make more sense in the context of inconsistent specifications. *Qualification* of inferences provides us with some intuition about our confidence in particular pieces of information in our specification, while *labelling* of specifications provides us with the infrastructure for tracking inconsistencies and identifying their likely sources.

4.1. Identifying likely sources of inconsistency

After identifying an inconsistency in a specification, our analysis attempts to indicate the likely source of that inconsistency before we decide on a further course of action. Using *labelled* QC reasoning, we obtain the labels

1 At this point in our work we assume a first order language without function symbols and existential quantifiers. This gives us certain computational advantages such as rendering consistency checking decidable (because, effectively, we are working with a propositional language).

of the assumptions used to derive an inconsistency. We use the term 'source' to denote the subset of the assumptions that we believe to be incorrect.

We use a labelled language to allow us to uniquely identify each item of our specification. We propagate the labels by labelling consequences with the union of the labels of the premises. This means we can identify the ramifications of each item in the reasoning, since each inference will be labelled. Labels can be used to differentiate different types of development information (e.g., constraints, domain knowledge, etc.) and in particular they can indicate the sources of information. For this paper, we adopt the following labelling strategy[2].

Definition 1. Let S be some set of atomic symbols such as an alphabet, and L a logic such as QC. If $i \subseteq S$ and $\alpha \in L$, then i: α is a labelled formula.

So for example, the labelled form of the resolution proof rule is:

$$\frac{\text{i: } \neg\alpha \vee \beta \qquad \text{j: } \alpha \vee \gamma}{\text{i} \cup \text{j: } \beta \vee \gamma}$$

To illustrate the use of labels, suppose we have the specification:

{a}: patient-waiting(London-Road)

{b}: ¬ patient-waiting(London-Road) ∨ ambulance-available(Hospital)

{c}: ambulance-available(Hospital)

and suppose {a}: patient-waiting(London-Road) and {b}: ¬patient-waiting(London-Road) ∨ ambulance-available(Hospital) have been a stable and well-accepted part of the specification for some time, and by contrast {c}: ambulance-available(Hospital) is just a new and tentative piece of specification. Then for the inconsistency {a, b, c}: ⊥, we could regard {c}: ambulance-available(Hospital) as the 'source' of the inconsistency.[3]

Informally then, a 'possible source' of an inconsistency is a subset of the assumptions used to obtain the inconsistency, and the remainder of the assumptions are consistent. Such a subset may be obtained by formally defining possible sources of inconsistency as follows.

Definition 2. Let Δ be a set of labelled formulae representing specification information, and let i be some label of some inference from Δ. The set of assumptions from Δ corresponding to the label is defined as follows:

$$\text{Formulae}(\Delta, \text{i}) = \{ \text{j: } \alpha \in \Delta \mid \text{j} \subseteq \text{i} \}$$

For an inconsistency i: ⊥, Formulae(Δ, i) is a possible source of the inconsistency if j ⊆ i and Formulae(Δ, i - j) ∈ $CON(\Delta)$, where $CON(\Delta)$ is the set of consistent subsets of Δ (formally defined in §4.2).

There maybe a large number of possible sources of a single inconsistency, and a number of options for addressing it, such as working only with the smallest sources, or working with only the sources that have the least effect on the number of inferences from the specification. However, if we are to act on inconsistency effectively, then we really need to identify the 'likely' sources of inconsistency. To do this, we assume that for any development information, there is some ordering over that information, where the ordering captures the likelihood of the information being erroneous. So, if i is higher in the ordering than j, then i: α is less likely to be erroneous than j: β. We assume this ordering is transitive, though not necessarily linear.

Moreover, if we assume there is an ordering over assumptions, then a more likely source is the smallest possible source that contains less preferred assumptions. Assuming such an ordering over development information is reasonable in software engineering. First, different kinds of information have different likelihoods of being incorrect. For example, "method rules" are unlikely to be incorrect, whereas some tentative specification information is quite possibly incorrect. Second, if a specification method is used interactively, a user can be asked to order pieces of specification according to likelihood of correctness.

There are a number of ways that this approach can be developed. First, there are further intuitive ways of deriving orderings over formulae and sets of formulae. These include ordering sets of formulae according to their relative degree of contradiction [17]. Second, there are a number of analyses of ways of handling ordered formulae and sets of ordered formulae. These include the use of specificity [26], ordered theory presentations [27], and prioritised syntax-based entailment [3].

4.2. Qualifying inferences from inconsistent information

When considering inconsistent information, we have more confidence in some inferences over others. For example, we may have more confidence in an inference α from a consistent subset of the specification if we cannot also derive ¬ α from another consistent subset of the specification. Therefore, we now provide formal definitions of some tools for qualifying inconsistent information. We follow these with an informal summary and an example. We begin with the definitions of some useful subsets of our specification.

Definition 3. Let Δ be a set of labelled formulae representing specification information. We form the following sets of sets of formulae:

2 There are many strategies that we could adopt for labelling software development information. Options include combinations of the source of the item, and time the item was inserted. For this, some mapping from labels to their associated meaning needs to be recorded. For instance, different developers could use different disjoint subsets of the labels.

3 ⊥ denotes an inconsistency.

$$CON(\Delta) = \{ \Gamma \subseteq \Delta \mid \Gamma \nvdash_Q i : \bot \}$$

$$INC(\Delta) = \{ \Gamma \subseteq \Delta \mid \Gamma \vdash_Q i : \bot \}$$

Essentially, $CON(\Delta)$ is the set of consistent subsets of Δ, and $INC(\Delta)$ is the set of inconsistent subsets of Δ.[4]

We can now define $MI(\Delta)$ as a set of sets of labels, where each set of labels corresponds to a set of minimally inconsistent formulae. A set of formulae is *minimally inconsistent* if every proper subset is consistent. Similarly, we define $MC(\Delta)$ as a set of sets of labels, where each set of labels corresponds to a set of maximally consistent formulae. A set of formulae is *maximally consistent*, if the set is consistent and adding any further formulae to the set from Δ causes the set to be inconsistent.[5]

We can consider a maximally consistent subset of a specification as capturing a "plausible" or "coherent" view on the specification. Furthermore, we consider $FREE(\Delta)$, which is equal to $\bigcap MC(\Delta)$, as capturing all the "uncontroversial" information in Δ. In contrast, we consider the set $MI(\Delta)$ as capturing all the "problematic" data Δ. Note that $MC(\Delta)$ is equal to $Labels(\Delta)$ - $MI(\Delta)$. Thus, reasoning with $FREE(\Delta)$ is equivalent to revising the specification by removing all the "problematic" data. This means we have a choice. We can either reason with the data directly using $FREE(\Delta)$ or we can revise the data by removing the formulae corresponding to $MI(\Delta)$.

We can now use these concepts to define three qualifications for an inference from inconsistent information[6].

Definition 4. Let Δ be a set of labelled formulae representing specification information, and let $\Delta \vdash_Q i : \alpha$ hold. We form the following qualifications for inferences:

α is an **existential inference** if $\exists k \in MC(\Delta)$ such that $i \subseteq k$.

α is a **universal inference** if $\forall k \in MC(\Delta)\ \exists j$ such that $j \subseteq k$ and $\Delta \vdash_Q j : \alpha$ holds.

α is a **free inference** if $i \subseteq FREE(\Delta)$.

Informally, a formula is an existential inference if it is an inference from a consistent subset of the specification. A formula is a universal inference if it is an inference from each maximally consistent subset of the specification. Finally, a formula is a free inference if it is an inference from the intersection of the maximally consistent subsets of the specification.

If α is a free inference, it is also a universal inference. Similarly, if α is a universal inference, it is also an existential inference. Clearly, if α is only an existential inference, then we are far less confident in it than if it was a universal inference. If it is a free inference, then it is not associated with any inconsistent information.

Example. Consider the following assumptions:

{a}: accident-occurred(London-Road) $\wedge$ accident-reported(London-Road)

{b}: accident-occurred(London-Road) $\wedge$ $\neg$accident-reported(London-Road)

{c}: ambulance-available(London-Road)

This gives two maximally consistent subsets:

Set 1

{a}: accident-occurred(London-Road) $\wedge$ accident-reported(London-Road)

{c}: ambulance-available(London-Road)

Set 2

{b}: accident-occurred(London-Road) $\wedge$ $\neg$accident-reported(London-Road)

{c}: ambulance-available(London-Road)

From this, accident-reported(London-Road) and $\neg$accident-reported(London-Road) are only existential inferences, whereas ambulance-available(London-Road) is a free inference, and accident-occurred(London-Road) is a universal inference.

These kinds of qualification are useful when reasoning with inconsistent information because they provide a clear and unambiguous relationship between the inferences and problematic data. This could be useful in facilitating further development in the presence of inconsistency, since we would feel happier about relying on the less qualified inferences. Furthermore, they provide a useful vocabulary for participants in the development process to discuss the inconsistent information.

Whilst there is an overlap for existential inferencing with the approach of truth maintenance systems (for example [10], [8]), we go beyond this by adopting universal and free inferencing. Furthermore, by adopting labelling, we integrate our inconsistency management with QC reasoning and with identifying likely sources of inconsistency.

4.3. Remarks on utility of analyses

Recall that a primary objective of our analysis of inconsistent information is to provide the developer with guidance on how to act in the presence of inconsistency. Reasoning in the presence of inconsistency that QC logic facilitates was the first step in achieving this objective. The simple analyses we have described above provide the

4 $\vdash_Q$ is the consequence relation for QC logic.

5 Let Δ be a set of labelled formulae representing specification information. If we define the function *Labels* as follows:

$Labels(\Delta) = \{ i \mid i : \alpha \in \Delta \}$

then the following set of labels can be formed:

$MI(\Delta) = \{\text{Labels}(\Gamma) \mid \Gamma \in INC(\Delta) \text{ and } \forall \Phi \in INC(\Delta)\ \ \Phi \not\subset \Gamma\}$

$MC(\Delta) = \{\text{Labels}(\Gamma) \mid \Gamma \in CON(\Delta) \text{ and } \forall \Phi \in CON(\Delta)\ \ \Gamma \not\subset \Phi\}$

$FREE(\Delta) = \bigcap MC(\Delta)$

6 The approach is a derivative of argumentative logics [13].

next step. They point to likely sources of inconsistency - to which more attention could be devoted, and they give us some indication of the "quality" of our information - which again guides our actions.

The simplicity of the tools we have presented is essential for usability and construction of tool support. Nevertheless, there remains an issue of scalability which we regard as a major concern. While we are exploring the *limits* on scalability by engaging in a number of large case studies, we are not necessarily trying to prove that our techniques scale up to industrial size specifications. In our ViewPoints framework, we already have an approach for partitioning specifications into more "manageable units", which are more amenable to the kinds of analysis we propose. ViewPoints encapsulate partial specifications that can be deliberately chosen to be of a size that can be handled by our techniques. Moreover, the ViewPoints framework itself and its support tools have been constructed with the intention of tolerating inconsistency [12, 24]. Thus, the tools we have presented can be added to our framework without hindering the ViewPoint-oriented development process. So for example, while a requirements engineer is developing his/her ViewPoint specification, our tools can be happily churning out (potentially) useful inferences and analysis results.

5. Application Example

To validate and illustrate the work presented in this paper, we present excerpts of an example application: eliciting and specifying the requirements of the London Ambulance Service (LAS). This case study was the focus of, and common example used by, delegates at the Eighth International Workshop on Software Specification and Design (IWSSD-8) [15]. As mentioned earlier (§4.3), by examining this case study we are not attempting to demonstrate the scalability of our approach, but rather to demonstrate its potential usefulness when used in conjunction with a host of other tools from the requirements engineer's toolbox. Thus, we do not expect the wholesale translation of large, monolithic specifications into QC logic on which we can then perform the kind of reasoning we have described. Rather, our aim is to reduce the complexity of our reasoning and analysis by restricting them to smaller partial specifications (ViewPoints) to which we then apply our tools. The exposition below is somewhat artificial in order to illustrate the issues and contributions presented in the paper.

5.1. Requirements document

Consider the requirements for a computer-aided ambulance despatching system. A reasonable requirements engineering method involves interviews with the staff involved in order to elicit the system's requirements. In our example, stakeholder analysis yielded, among others, the following "client authorities" who laid down the procedures deployed by LAS. The requirements document for the LAS that contained information such as the following.

Stakeholder 1: LAS Incident Room Controller

- A medical emergency is either the result of an illness or accident.
- On receipt of a phone call reporting a medical emergency, an ambulance should be despatched to the scene.
- On receipt of a phone call, if the incident is judged not to be a medical emergency, then the call should be transferred to another emergency service (e.g., police or fire brigade).

Stakeholder 2: Operations Manager

- On receipt of a phone call reporting an incident, if an ambulance is available then it should be despatched to the scene.
- On receipt of a phone call reporting an incident, if an ambulance is not available then it should not be despatched to the scene.

Stakeholder 3: Logistics Manager

- If no ambulance operators (drivers/medics) are available, then no ambulance is available.
- If no ambulances are available, then initiate a search for a free ambulance.
- If one year has passed since the maintenance work was last done on an ambulance, then perform a safety check on that ambulance.

5.2. Preliminary specification

From the above requirements document, one could generate, for example, agent hierarchies, data flow diagrams, action tables, object diagrams, and so on. Below we use QC logic (as presented in this paper), directly, to represent this specification information.

Stakeholder 1: LAS Incident Room Controller

{a}: $\forall$X,Y, accident(X, Y) $\vee$ illness(X, Y) $\leftrightarrow$ medical-emergency(X, Y)

{b}: $\forall$X, Y, call(X, Y) $\wedge$ medical-emergency(X, Y) $\rightarrow$ despatch-ambulance(X, Y)

{c}: $\forall$X, Y, call(X, Y) $\wedge$ $\neg$ medical-emergency(X, Y) $\rightarrow$ transfer-service(X, Y)

Stakeholder 2: Operations Manager

{d}: $\forall$X, Y, call(X, Y) $\wedge$ ambulance-available(X) $\rightarrow$ despatch-ambulance(X, Y)

{e}: $\forall$X, Y, call(X, Y) $\wedge$ $\neg$ ambulance-available(X) $\rightarrow$ $\neg$despatch-ambulance(X, Y)

Stakeholder 3: Logistics Manager

{f}: ∀X, Y, ¬has-one-or-more(X, Y) → ¬ambulance-available(X)

{g}: ∀X, Y, ¬ambulance-available(X) → initiate-search-for-free-ambulance

{m}: ∀X, over-one-year-since-last-maintenance(X) → initiate-ambulance-safety-check(X)

5.3. Inconsistency handling

From the above preliminary formal specification, we can now demonstrate the reasoning and analysis described in sections 3 and 4.

5.3.1. Reasoning

To check certain scenarios with respect to the preliminary specification, we must add further relevant facts (e.g., domain knowledge) to model each scenario. For example, consider the following facts:

{h}: accident(Anthony, London-Road)

{i}: call(Anthony, London-Road)

{j}: ¬has-one-or-more(Ambulance1, Operator)

{k}: ¬illness(Anthony, London-Road)

{n}: over-one-year-since-last-maintenance(Ambulance2)

From this scenario, we can generate the following (inconsistent) inferences:

{a, b, h, i}:
despatch-ambulance(Ambulance1, London-Road)

{e, f, i, j}:
¬despatch-ambulance(Ambulance1, London-Road)

Using QC logic, we can still continue reasoning with the above facts, together with the preliminary specification, to generate additional inferences such as the following:

{f, g, j}: initiate-search-for-free-ambulance

{m, n}: initiate-ambulance-safety-check(Ambulance2)

So even though the assumptions are inconsistent, we can generate a useful inference. It is possible then, for example, to develop a definition for the initiate-search-for-free-ambulance procedure, without necessarily having to resolve the inconsistencies in the preliminary specification (although one may want to perform the analysis below before developing a definition for a procedure that may later have to be retracted).

5.3.2. Analysis: qualifying inferences

The following inferences are only existential inferences, and hence need to be treated with caution when the specification is revised or analysed further.

{a, b, h, i}:
despatch-ambulance(Ambulance1, London-Road)

{e, f, i, j}:
¬despatch-ambulance(Ambulance1, London-Road)

In contrast, the following inference is a universal inference, and hence is less likely to be retracted when the specification is revised.

{f, g, j}: initiate-search-for-free-ambulance

The following is a free inference from the specification, which is reassuring given the preliminary nature of the specification.

{m, n}: initiate-ambulance-safety-check(Ambulance2)

5.3.3. Analysis: identifying sources of inconsistency

For the inconsistency identified above, there are two sets of labels, in particular, that refer to problematical data. These are the labels attached to the conflicting inferences generated above, {a, b, h, i} and {e, f, i, j}. There are many possible sources of the inconsistency. However, if we assume the facts we added for the scenario, labelled from the set {h, i, j, k, n} are not causing the problem, we order these *above* the set of labels, {a, b, c, d, e, f, m}, referring to the preliminary specification. Using this ordering, we obtain a smaller subset containing the likely sources of the inconsistency, namely {a, b, e, f}. These pieces of procedural information were elicited from all three stakeholders, who need to be consulted again in order to rectify this problem (although we may also have some ordering of information according to the particular participant from which it was elicited; e.g. "the boss is always right"!).

6. Related Work

The overwhelming majority of work on consistency management has dealt with tools and techniques for maintaining consistency and avoiding inconsistency. Increasingly however, researchers have begun to study the notion of consistency in software systems, have recognised the need to formalise this notion, and have proposed techniques for tolerating or even living with inconsistencies; e.g., [2, 5, 14, 18, 22, 28, 29]. A review of this work can be found in [19, 20, 23].

Other related approaches address inconsistencies that arise in software development processes themselves. For example, an inconsistency may occur between a software development process definition and the actual (enacted) process instance [9]. Such an inconsistency between "enactment state" and "performance state" is often avoided by blocking further development activities until some precondition is made to hold. Since this policy is overly restrictive, many developers attempt to fake conformance to the process definition (for example, by fooling a tool into thinking that a certain task has been performed in order to continue development). Cugola et al. [7] have addressed exactly this problem in their temporal logic-based approach which is used to capture and tolerate some deviations from a process description during execution. Deviations are tolerated as long as they do not affect the correctness of the system (if they do, the incorrect data must be fixed, or the process model - or its active instance - must be changed). Otherwise, deviations are tolerated, recorded and propagated - and "pollution analysis" (based on logical reasoning) is performed to identify possible sources of inconsistency.

From the AI and logics communities there have been a number of other related contributions that are relevant, including fuzzy sets and non-monotonic logics (for a review, see [19, 21]). Whilst they constitute important developments that could be incorporated in our framework, they are not directly oriented to the inconsistency management issues that we consider with in this paper. In the main they are focused on resolving inconsistency by finding the best possible inferences for any given set of information, whereas we really need to be able to analyse inconsistent information, consider options, and track information to find likely sources of inconsistency.

7. Discussion, Conclusions and Future Work

Our earlier work began by providing a framework for multi-perspective software development in which multiple development participants, and the partial specifications they maintained, were represented by ViewPoints. The inconsistencies that inevitably arose between multiple overlapping ViewPoints led us to adopt an inconsistency handling approach that was tolerant of such inconsistencies. This approach relied on identifying inconsistencies, the context in which they arose, and the actions that could be performed in their presence. We further recognised that such actions did not need to remove inconsistencies immediately, but rather allowed continued reasoning and development in their presence. Keeping track of deductions made during reasoning, and deciding on what actions to perform in the presence of inconsistencies, identified the need to analyse inconsistencies in this context. This paper addressed the formal analysis of such inconsistencies..

We summarised the use of a quasi-classical logic to reasoning in the presence of inconsistency. We then examined the use of labelled QC logic to "audit" reasoning results and to "diagnose" inconsistencies. The labels facilitated the identification of likely sources of inconsistencies. We further proposed some tools for qualifying the different kinds of deductions we made during reasoning. These tools provided us with a measure of confidence in our specification information.

We believe that such logical analysis provides developers with heuristics and guidance about what actions we can perform in the presence of particular inconsistencies (for example, actions to resolve a conflict, delay resolution, ameliorate an inconsistent specification, etc.). Our immediate research agenda is to examine these inconsistency handling actions further within our framework. In particular, we would like to examine the correlation, if any, between certain kinds of analysis results and the consequent inconsistency handling actions that could be taken.

We believe that our work provides the *foundations* for supporting a software specification process in which inconsistencies are analysed to determine the course of action needed for further development. This recognises the evolutionary nature of software development and provides a formal, yet flexible, mechanism for managing inconsistencies.

Acknowledgements

We would like to acknowledge the contributions and feedback of Alex Borgida, Gianpaolo Cugola, Elisabetta Di Nitto, Martin Feather, Anthony Finkelstein, Dov Gabbay, Carlo Ghezzi, Michael Goedicke, Jeff Kramer, Jonathan Moffett and Axel van Lamsweerde. This work was partially funded by the UK EPSRC as part of the VOILA project (GR/J15483), the British Council, and the European Union as part of the Basic Research Actions DRUMS II , PROMOTER II and the ISI project (ECAUS003).

7. References

[1] W. Atkinson and R. J. Cunningham (1991); "Proving Properties of Safety-Critical Systems"; *Software Engineering Journal*, 6(2): 41-50, March 1991; IEE/BCS.

[2] R. Balzer (1991); "Tolerating Inconsistency"; *Proceedings of 13th International Conference on Software Engineering (ICSE-13)*, Austin, Texas, USA, 13-17th May 1991, 158-165; IEEE CS Press.

[3] S. Benferhat, D. Dubois and H. Prade (1993); "Argumentative Inference in Uncertain and Inconsistent Knowledge Bases"; *Proceedings of Uncertainty in Artificial Intelligence*, Morgan Kaufmann.

[4] P. Besnard and A. Hunter (1995); "Quasi-classical Logic: Non-trivializable classical reasoning from inconsistent information"; *(In) Symbolic and Quantitative Approaches to Uncertainty (ECSQARU '95);* C. Froidevaux and J. Kohlas (Ed.); 44-51; LNCS, 946, Springer-Verlag.

[5] A. Borgida (1985); "Language Features for Flexible Handling of Exceptions in Information Systems"; *Transactions on Database Systems*, 10(4): 565-603, December 1985; ACM Press.

[6] M. Costa, R. J. Cunningham and J. Booth (1990); "Logical Animation"; *Proceedings of 12th International Conference of Software Engineering*, Nice, France, 144-149; IEEE CS Press.

[7] G. Cugola, E. Di Nitto, C. Ghezzi and M. Mantione (1995); "How To Deal With Deviations During Process Model Enactment"; *Proceedings of 17th International Conference on Software Engineering (ICSE-17)*, Seattle, USA, 23-30th April 1995, 265-273; ACM Press.

[8] J. De Kleer (1986); "An Assumption-based TMS"; *Artificial Intelligence*, 28: 127-162.

[9] M. Dowson (1993); "Consistency Maintenance in Process Sensitive Environments"; *Proceedings of Workshop on Process Sensitive Environments Architectures*, Boulder, Colorado, USA, Rocky Mountain Institute of Software Engineering (RMISE).

[10] J. Doyle (1979); "A Truth Maintenance System"; *Artificial Intelligence*, 12: 231-272.

[11] S. Easterbrook and B. Nuseibeh (1995); "Inconsistency Management in an Evolving Specification"; *Proceedings of 2nd International Symposium on Requirements Engineering (RE 95)*, York, UK, 48-55; IEEE CS Press.

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

[13] M. Elvang-Goransson and A. Hunter (1995); "Argumentative Logics"; *Data and Knowledge Engineering*, 16: 125-145.

[14] M. Feather (1995); "Modularized Exception Handling"; *Draft technical report,* 9th February 1995; USC/Information Sciences Institute, Marina del Rey, California, USA.

[15] A. Finkelstein and J. Dowell (1996); "A Comedy of Errors: the London Ambulance Service case study"; *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, Schloss Velen, Germany, 22-23rd March 1996, 2-4; IEEE CS Press.

[16] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh (1994); "Inconsistency Handling in Multi-Perspective Specifications"; *Transactions on Software Engineering*, 20(8): 569-578, August 1994; IEEE CS Press.

[17] D. Gabbay and A. Hunter (1995); "Negation and Contradiction"; *(In) What is Negation;* Oxford University Press.

[18] T. M. Hagensen and B. B. Kristensen (1992); "Consistency in Software System Development: Framework, Model, Techniques & Tools"; *Software Engineering Notes (Proceedings of ACM SIGSOFT Symposium on Software Development Environments)*, 17(5): 58-67, 9-11th December 1992; SIGSOFT & ACM Press.

[19] A. Hunter (1996); *Uncertainty in Information Systems*; McGraw-Hill.

[20] A. Hunter and B. Nuseibeh (1995); "Managing Inconsistent Specifications: Reasoning, Analysis and Action"; *Technical report,* June 1995; Department of Computing, Imperial College, London, UK.

[21] P. Krause and D. Clark (1993); "Representing Uncertain Knowledge"; *Intellect.*

[22] K. Narayanaswamy and N. Goldman (1992); ""Lazy" Consistency: A Basis for Cooperative Software Development"; *Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW '92)*, Toronto, Ontario, Canada, 31st October - 4th November, 257-264; ACM SIGCHI & SIGOIS.

[23] B. Nuseibeh (1996); "To Be And Not To Be: On Managing Inconsistency in Software Development"; *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, Schloss Velen, Germany, 22-23rd March 1996, 164-169; IEEE CS Press.

[24] B. Nuseibeh and A. Finkelstein (1992); "ViewPoints: A Vehicle for Method and Tool Integration"; *Proceedings of 5th International Workshop on Computer-Aided Software Engineering (CASE '92)*, Montreal, Canada, 6-10th July 1992, 50-60; IEEE CS Press.

[25] B. Nuseibeh, J. Kramer and A. Finkelstein (1994); "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification"; *Transactions on Software Engineering*, 20(10): 760-773, October 1994; IEEE CS Press.

[26] D. Poole (1985); "An Assumption-based TMS"; *Artificial Intelligence*, 36: 24-47.

[27] M. Ryan (1992); "Representing Defaults as Sentences with Reduced Priority"; *Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann.

[28] R. W. Schwanke and G. E. Kaiser (1988); "Living With Inconsistency in Large Systems"; *Proceedings of the International Workshop on Software Version and Configuration Control*, Grassau, Germany, 27-29 January 1988, 98-118; B. G. Teubner, Stuttgart.

[29] P. Zave and M. Jackson (1993); "Conjunction as Composition"; *Transactions on Software Engineering and Methodology*, 2(4): 379-411, October 1993; ACM Press.

========================

APPENDIX A: Formal definition and proof theory of labelled QC logic

For a more complete description of QC and labelled QC logic see [4] and [20] respectively.

Language of labelled QC logic

At this point in our work we assume a first order language without function symbols and existential quantifiers. This gives us certain computational advantages such as rendering consistency checking decidable (because, effectively, we are working with a propositional language).

Definition A1. Let P be a set of predicate symbols, V be a set of variable symbols, and C a set of constant symbols. Let A be a set of atoms, where $A = \{p(q_1,...,q_n) \mid p \in P \text{ and } q_1,...,q_n \in V \cup C\}$. We call $p(q_1,...,q_n)$ a ground atom iff $q_1,...,q_n$ are all constant symbols, otherwise we call it unground.

Example. Let has-exactly-one be a predicate symbol; X,Y be variable symbols; and Cashier, Terminal be constant symbols. Then has-exactly-one(X,Y) is an unground atom, has-exactly-one(X,Terminal) is an unground atom, and has-exactly-one(Cashier,Terminal) is a ground atom.

Definition A2. Let F be the set of classical propositional formulae formed from a set of atoms A, and the $\wedge$, $\vee$, $\rightarrow$ and $\neg$ connectives. We abbreviate the formula $\alpha \wedge \neg\alpha$ by the formula $\perp$, which we read as "inconsistency". We call a formula grounded iff it is made from only ground atoms, otherwise we call it ungrounded.

Definition A3. Let L be the set of formulae formed from F, where if $\alpha \in F$, and $x_1,..,x_n$ are the free variables of α, then $\forall x_1,...,\forall x_n \alpha \in L$.

Hence the set L contains only universally quantified formulae, where the quantifiers are outermost, and ground formulae.

Definition A4. Let $\vdash_X$ be some consequence relation for some logic X, defined by some proof rules. Then, the logic X is trivialisable if and only if for all α, β in the language of X, $\{\alpha, \neg\alpha\} \vdash_X \beta$.

Note that classical logic is trivialisable according to this definition. The following two definitions are used to explain the proof theory concisely.

Definition A5. For each atom $\alpha \in L$, α is a literal and $\neg\alpha$ is a literal. For $\alpha_1 \vee .. \vee \alpha_n \in L$, $\alpha_1 \vee .. \vee \alpha_n$ is a clause iff each of $\alpha_1, .., \alpha_n$ is a literal. For $\alpha_1 \wedge .. \wedge \alpha_n \in L$, $\alpha_1 \vee .. \vee \alpha_n$ is in conjunctive normal form (CNF) iff each of $\alpha_1, .., \alpha_n$ is a clause.

Definition A6. For $\alpha_1 \wedge .. \wedge \alpha_n \in L$, $\beta \in L$, $\alpha_1 \wedge .. \wedge \alpha_n$ is a CNF of β iff $\alpha_1 \wedge .. \wedge \alpha_n \vdash \beta$ and $\beta \vdash \alpha_1 \wedge .. \wedge \alpha_n$ and $\alpha_1 \wedge .. \wedge \alpha_n$ is in CNF.

Definition A7. Let S be some set of atomic symbols such as an alphabet. If $i \subseteq S$ and $\alpha \in L$, then i: α is a labelled formula. Let M be the set of formulae.

Proof theory for labelled QC logic

The proof theory of QC logic provides the power to derive a CNF of any formula, together with the power of resolution. As a "last step" in any derivation, disjunction introduction is also allowed. This means that any resolvant of a set of formulae can be derived, but no trivial formulae can be derived. This proof theory is presented as a set of natural deduction rules. All the QC natural deduction rules hold in classical logic, but some classical deduction rules, such as *(ex falso quodlibet)* do not hold in QC logic. We obtain *labelled QC logic* by using only labelled formulae as assumptions, and by amending the natural deduction rules to propagate the labels. The label of the consequent of a rule is the union of the labels of the premises of the rule.

Definition A8. Assume that $\wedge$ is a commutative and associative operator, and $\vee$ is a commutative and associative operator.

$$\frac{i: \alpha \wedge \beta}{i: \alpha}$$ [Conjunct elimination]

$$\frac{i: \alpha \vee \alpha \vee \beta}{i: \alpha \vee \beta}$$ [Disjunct contraction]

$$\frac{i: \alpha \vee \beta}{\neg\neg\alpha \vee \beta}$$ [Negation introduction]

$$\frac{\neg\neg\alpha \vee \beta}{\alpha \vee \beta}$$ [Negation elimination]

$$\frac{i: \forall x \alpha}{i: \beta}$$ [Universal instantiation, where β is obtained from α by replacing every occurrence of x by the same constant]

$$\frac{i: \alpha \vee \beta \quad j: \neg\alpha \vee \gamma}{i \cup j: \beta \vee \gamma}$$ [Resolution]

$$\frac{i: \alpha \quad j: \neg\alpha \vee \beta}{i \cup j: \beta} \qquad \frac{i: \alpha \vee \beta \quad j: \neg\alpha}{i \cup j: \beta}$$

[Arrow Elimination]

$$\frac{i: \alpha \vee (\beta \rightarrow \gamma)}{i: \alpha \vee \neg\beta \vee \gamma} \qquad \frac{i: \alpha \vee \neg(\beta \rightarrow \gamma)}{\alpha \vee (\beta \wedge \neg\gamma)}$$

$$\frac{i: \beta \rightarrow \gamma}{i: \neg\beta \vee \gamma} \qquad \frac{i: \neg(\beta \rightarrow \gamma)}{i: \beta \wedge \neg\gamma)}$$

[Distribution]

$$\frac{i: \alpha \vee (\beta \wedge \gamma)}{i: (\alpha \vee \beta) \wedge (\alpha \vee \gamma)} \qquad \frac{i: (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)}{i: \alpha \wedge (\beta \vee \gamma)}$$

[de Morgan's laws]

$$\frac{i: \neg(\alpha \wedge \beta) \vee \gamma}{i: \neg\alpha \vee \neg\beta \vee \gamma} \qquad \frac{i: \neg(\alpha \wedge \beta)}{i: \neg\alpha \vee \neg\beta}$$

$$\frac{i: \neg(\alpha \vee \beta) \vee \gamma}{i: \neg\alpha \wedge \neg\beta \vee \gamma} \qquad \frac{i: \neg(\alpha \vee \beta)}{i: \neg\alpha \wedge \neg\beta}$$

$$\frac{i: \alpha}{i: \alpha \vee \beta}$$ [Disjunct introduction]

Labelled QC logic is used by providing any set of labelled formulae (i.e., any $\Delta \subseteq M$ as assumptions, and any classical formula (i.e., any $\alpha \in L$) as a query. For a query that is a ground formula, it follows from the assumptions with some label i (denoted $\Delta \vdash_Q i: \alpha$) if and only if there is a derivation of a CNF of the query, labelled i, from the assumptions using the labelled QC natural deduction rules, remembering that disjunction introduction is only allowed as a last step in a derivation. For a query that is universally quantified, form a ground formula by instantiating it with constants that do not appear in the assumptions, and then treat the query as above.