Restructuring Requirements Specifications for Managing Inconsistency and Change: A Case Study

Alessandra Russo, Bashar Nuseibeh and Jeff Kramer
Department of Computing,
Imperial College, London SW7 2BZ, U.K.
{ar3, ban, jk}@doc.ic.ac.uk

Abstract

This paper describes our experiences in restructuring multi-perspective requirements specifications in order to identify and analyse inconsistencies and manage change. A partial, heterogeneous and reasonably large requirements specification from a NASA project was analysed and decomposed into a structure of "viewpoints", where each viewpoint encapsulates partial requirements of some system components described in the specification. Relationships between viewpoints were identified which included not only the interactions explicitly stated in the requirements but also some implicit and potentially problematic inter-dependencies. The restructuring process and a first informal analysis of the resulting relationships enabled the detection of inconsistencies and the definition of some interesting domain-dependent *consistency rules. We believe that this restructuring into viewpoints also facilitated requirements understanding through partitioning, and requirements maintenance and evolution through explicit identification of the inter-viewpoint relationships.*

1 Introduction

The requirements engineering process of large and complex systems often involves the participation of many developers who operate according to their specific skills, experience and knowledge. This is particularly the case for the elicitation and development of requirements specifications. Multiple participants are normally involved in specifying different partial requirements of the same underlying system. Inevitably, these requirements are more appropriately described using various development methods and diverse representation schemes, reflecting the different knowledge and *perspectives* that the developers have about the underlying domain. Forced integration into a single uniform specification is difficult and can obscure individual views and inconsistencies. On the other hand, relationships between the various partial specification fragments, such as overlaps and inter-dependencies, are difficult to identify making consistency checking and inconsistency handling difficult as well.

In earlier work we advocated an analysis of the relationships between the methods and representation schemes used to develop specification fragments [19], as well as relationships between domain-specific terms and concepts (e.g., identifying different terms which denote the same object) [7]. Our aim was to preserve the individual views, to support the development method and representation scheme appropriate for each view, and to make explicit the relationships between these views.

However, given that it is not always feasible to develop, *a priori*, multi-perspective specifications with a structure amenable to analysis and management, an approach that facilitates *restructuring* of *existing* specifications seems appropriate. The approach described in this paper restructures existing informal requirement specifications, *enriching* them by identifying and explicitly representing interactions and relationships between different parts of the requirements, so facilitating consistency checking, validation, and subsequent evolutionary change.

Specifically, this paper describes our practical experiences in restructuring a partial, reasonably large, multi-perspective requirements specification taken from NASA's *International Space Station* (ISS) project. We deploy the Viewpoints framework [9, 19] as a restructuring tool. This framework facilitates an explicit separation of the perspectives of different developers and their representation by "ViewPoints"[1], allowing the use of different development methods

[1] In this paper, we write the term "viewpoint" in lower case throughout, even though we are still referring to our particular form of "ViewPoints" described in earlier work [9, 19].

and representation schemes. Inter-viewpoint rules are defined to provide a means for structural integration of different specification fragments, a means for consistency checking and inconsistency analysis, and a means for facilitating change management.

Our case study is an informal, but structured, partial requirements specification of an integrated hardware and software system (C&DH), which is part of a wider system for handling the operation of an earth-orbiting space station. We focus our attention on a particular function of the C&DH, namely, the "Fault, Detection, Isolation and Recovery" (FDIR) function. We illustrate how the informal specification was restructured into related viewpoints, and identify some inconsistencies revealed by this process. We also examine three other different chronological versions of the same specification to illustrate the suitability of our approach for tracking evolutionary changes and assessing their impact on the consistency of the specification.

The approach which we describe in this paper is a departure from the "traditional" way in which viewpoints have been deployed in the past. Thus, while the case study serves to validate the useability of viewpoints in a practical setting, it also illustrates the benefits of using them to "reverse engineer" a multi-perspective structure, given a monolithic specification. More generally, we believe that our restructuring of the original specification made it more accessible, aiding our own understanding of the underlying domain. The identification and representation of *implicit relationships* between various parts of the specifications also revealed inconsistencies that would have been difficult to detect by other means.

The paper is structured as follows. Section 2 describes the background to our approach including a summary of multi-perspective software development using viewpoints [9, 19] and a brief overview of our viewpoint-based restructuring approach. In Section 3, we describe the application of our approach to the case study, describe some inconsistencies that were revealed by our approach, and discuss the way in which our restructuring facilitates change management. In Section 4 we present and discuss the lessons learned from this experience. Section 5 compares our approach to related work, while Section 6 summarises our conclusions and future plans to support improved inconsistency analysis, requirements traceability, and inconsistency handling.

2 Background

The basic principle underlying the viewpoints framework is separation of concerns. Viewpoints are loosely coupled, locally managed, distributable objects that encapsulate partial representation knowledge (a notation), development knowledge (a process) and specification knowledge (a view) [9]. From a methodological standpoint, a viewpoint's notation and process are reusable attributes, and are called a viewpoint *template* (a viewpoint "type" or development "technique"). Thus, viewpoints are *instantiations* of templates, which capture some partial specification, represented in a particular notation and developed by following a particular process. *In-viewpoint* rules for each viewpoint define its semantics, requirements for well-formedness, and internal consistency.

The key to successful (and meaningful) deployment of multiple viewpoints in any software development setting, is an understanding and subsequent expression of *inter-viewpoint* relationships that define interdependencies and overlaps between viewpoints. Such relationships may be between the templates from which viewpoints are instantiated (e.g., syntactic relationships between constructs of different viewpoint notations) [5, 19], or specification domain-dependent relationships that (usually) emerge as development proceeds [7]. These relationships are fundamental elements of any viewpoints' structure that results from a multi-perspective development process. They provide the structural integration "glue" that holds a multi-perspective specification together, and are the only means by which consistency and completeness are meaningful in the viewpoints framework.

Some inter-viewpoint relationships are definable before development begins. This, of course, is not always the case. In a requirements engineering context, relationships between different requirements emerge as further requirements are elicited and analysed. With this in mind, we adapted the viewpoint-oriented development approach in order to examine its suitability for handling requirements specifications which are (1) not originally structured as a multi-perspective specification, and (2) still evolving.

The basic approach Our restructuring approach is situated in the middle ground between informal (and error prone) specification inspection [16] and (expensive) formal analysis [6]. Our objective is to describe existing specifications as a structure of related viewpoints, and then use this as the basis for conducting a series of consistency checks and inconsistency analyses.

The approach is comprised of five activities.

Existing informal specifications are *decomposed* into parts, with each part *represented* using some notation within a viewpoint. The viewpoints' structure is then *enriched* by identifying and explicitly defining in-viewpoint and inter-viewpoint rules that express the consistency relationships within and between viewpoints. An *analysis* of this new structure is then used to validate the specification and check its consistency. The results of the analysis are then used to determine the inconsistency *handling* process and to support change management.

3 Case Study

Our case study was based on a partial, multi-perspective requirements specification of the fault protection software of NASA's International Space Station (ISS) project. This is an international co-operative programme for the construction and use of a space station orbiting earth. One of the main functions of the space station software is the *Command and Control* (C&C) function. This is responsible for the station operations by monitoring its functions, overriding them when necessary, performing fault detection, and issuing commands to control the main station modes (e.g., docking with a Shuttle, re-boosting to a higher orbit, maintaining microgravity for scientific experiments). The C&C function is carried out as part of the ISS Command and Data Handling (C&DH) system. The C&DH system is comprised of a set of Computer Software Configuration Items (CSCIs) installed in a hierarchically organised set of processors (Multiplexer/Demultiplexers - MDMs) which communicate among themselves via MIL-STD-1553 buses. The main flight processor responsible for the C&C function is denoted by "C&C MDM". The CSCI within the C&C MDM is the Command and Control Software (CCS) CSCI. The C&C MDM is located at the top of the hierarchy and is the controller of the 1553 buses to the MDMs in the hierarchy tier directly below (referred to as the Bus Controller - BC). The MDMs (along with other hardware) at this next tier are designated as Remote Terminals (RTs) on the buses.

The informal requirements specifications we analysed consisted of two partial requirements documents of the C&C MDM CSCI, called *Control Bus MDM Management* (4 pages) and *1553 Bus Failure, Detection, Isolation and Recovery* (20 pages). The former describes the management of the MDMs connected to the BC (one of these MDMs is also the C&C MDM) which includes initialisation, shutdown and *failure recovery*. The last of these specifies the management of the control buses – e.g., detecting which of the 1553 bus channels have communication failures. Both these specifications are mainly written in English, and they include reference to some large tables containing information such as "Data Item Names", "Identification Numbers" and occasionally inputs and outputs of some of the functions. The document structure is composed of sections and subsections. These are mostly organised in paragraphs, and in each paragraph alternative requirements are often listed. The 1553 FDIR document also includes a flowchart describing the general C&C FDIR behaviour model.

3.1 Restructuring into viewpoints

The two requirements documents were decomposed into "high-level" viewpoints according to their section and subsection structures, and then refined into additional "lower-level" viewpoints according to their specification contents. The result of the decomposition was a functional hierarchy of viewpoints, with an increasing level of specification detail. Parent viewpoints tended to denote the main inputs and outputs of different functions, whereas leaf viewpoints tended to describe (parts of) functions in more detail.

Four main viewpoint templates were defined, called *hierarchic tree* (HT), *input output flow* (IOF), *data flow diagram* (DFD) and *state transition diagram* (STD). The HT template was used to instantiate a viewpoint describing the hierarchic functional decomposition of the specifications. The STD template was used to instantiate a viewpoint representing the state-based behaviour of the FDIR function described in the requirements specifications by a flowchart[2]. The IOF template represented functions or processes in terms of inputs and outputs, while the DFD template used a representation style similar to Tabular Collection Forms used in CORE [18] which represents functions in terms of actions, their inputs and outputs, and the inputs' sources and outputs' destinations. Part of the HT-based viewpoint is shown in Figure 1. In developing the HT template, a number of domain-independent in-viewpoint rules were defined. These expressed syntactic properties of the tree-style such as *"there is no link between nodes that are at the same level in the hierarchy"* and consistency properties within the viewpoint such as *"all nodes must have different labels"*. Structural domain-independent inter-viewpoint rules were also defined. These described the related viewpoints that

[2]This was the only semi-formal description given in the 1553 FDIR requirements document.

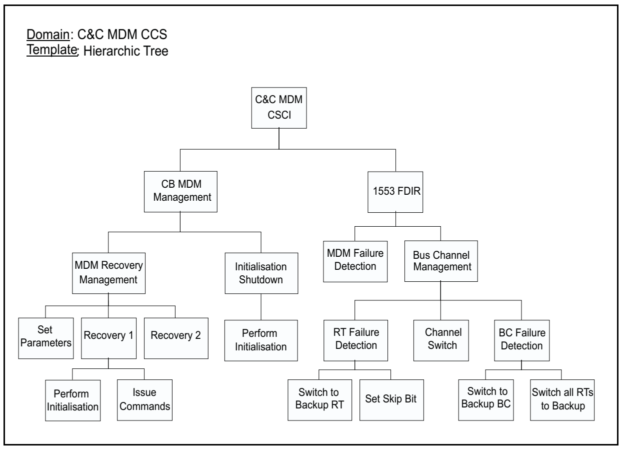

Figure 1: *Part of the hierarchical decomposition into viewpoints.*

should be instantiated for each node in the HT viewpoint. For example, the rule:

> (HT1) *For each leaf node there exists a VP instantiated from the DFD template.*

partly defined the meaning of a "complete" restructuring of the specification – if some part identified in the hierarchical decomposition is not elaborated by some other instantiated viewpoint, then this rule is violated. Other rules expressed properties related to hierarchic (parent-child) relationships (for detailed examples see [22]).

In the IOF template, functions were represented as alternative processes and inputs and outputs. Inputs can be either events (e.g., "loss of 1553 communication"), particular values of some special control parameters (e.g., "failure status = failed for RT"), or some specific commands (e.g., "Reconfigure MDM"). Similarly for the outputs. Processes can be either "non-primitive" or "primitive". Domain-dependent in-viewpoint rules specify the processes which are non-primitive. A domain-independent in-viewpoint rule (IOF1) was defined to express the consistency constraint that *"each data input and output is uniquely associated with a single process"*. Non-primitive processes can be decomposed further. For each of these processes additional viewpoints were instantiated. This was captured by the domain-independent inter-viewpoint rule:

> (IOF2) *For each non-primitive process there exists a viewpoint instantiated from either the IOF template or the DFD template.*

Two viewpoints describing the "MDM Recovery Management" and the "RT Failure Detection" functions were instantiated from the IOF template. The MDM Recovery Management viewpoint included two processes, called "Recovery1" and "Recovery2" which are responsible for MDM failure recovery. Each of these two processes are triggered by different inputs. For examples, the input "Reconfigure MDM" triggers the process Recovery1, and the input "Reconfigure C&CMDM" triggers the process Recovery2. The viewpoint describing the RT Failure Detection function, on the other hand, included two non-primitive processes, called "Set Skip Bit" and "Switch to Backup RT" with their respective inputs and outputs. A detailed description of these and other viewpoints is given in [22].

The DFD template facilitated a more detailed representation of (parts of) functions. These were given in terms of actions, inputs used by the actions and outputs produced by the action, the sources which generate the inputs and the destinations which use the outputs. Sources and destinations can themselves be other functions or processes described in the specification. An example of DFD viewpoint was a viewpoint with domain "Set Skip Bit" instantiated to further specify the non-primitive process "Set Skip Bit". It described an action, called "Verify Conditions", that checks the occurrence of alternative sets of conditions in addition to those described in the RT Failure Detection viewpoint, for the process Set Skip Bit. The result of this checking is used by the RT Failure Detection function. These alternative sets of conditions are described as inputs and outputs of the action Verify Conditions, with RT Failure Detection as output destination.

The last template used in the restructuring was the STD template. This was used to represent the general behaviour of the 1553 FDIR function described in a flowchart in the original specification. It described a series of different boolean conditions (or events) and their associated intermediate internal states which are covered by the 1553 FDIR function in order to detect what type of failure has occurred and what kind of recovery action to take. Recovery actions are processes external to the 1553 FDIR. The STD template represented this kind of flowchart specification by associating each internal and external state with a box, each boolean value of each condition with an arrow going from one box (state) to another, and declaring by means of an domain-dependent in-viewpoint rule which of the states was an external recovery state. Part of this STD-based viewpoint is shown in Figure 2.

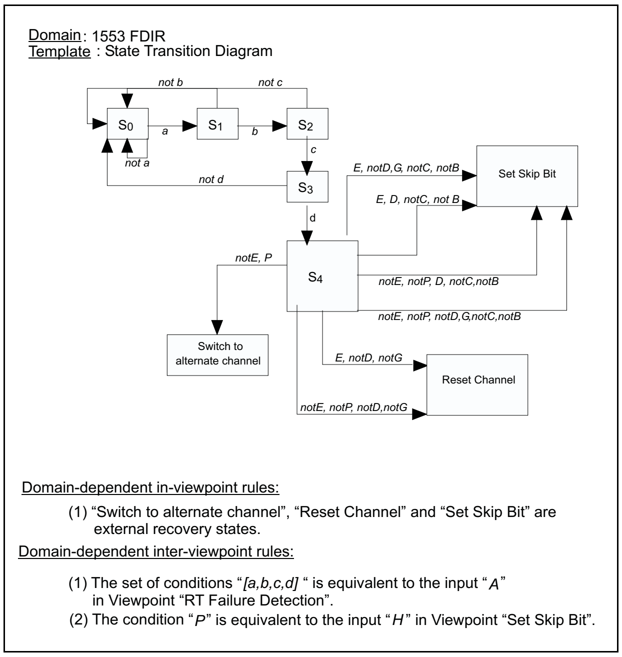

Figure 2: *Part of 1553 FDIR.*

Enriching the structure with inter-viewpoint rules Having defined a collection of viewpoint templates and some basic inter-viewpoint relationships between them, we then elicited further implicit relationships between viewpoints to enrich the existing structure. These were dictated partly by the syntactic properties of the templates – input/output arrows in the IOF and inputs/outputs in the DFD template denote the same class of object, and partly by the definition of domain-dependent inter-viewpoint rules which establish "ontological overlaps" [23] between input/output data of different viewpoints. Links between viewpoints expressed by these rules were not expressed in the original specification document, and the document structure itself would have not facilitated their identification since the constituent specifications were parts of two different document sections.

Similar rules were defined for the DFD template, also taking into account information about the source and destination of inputs and outputs respectively. An example rule was:

> (DFD1) *The outputs of a DFD viewpoint are inputs to any viewpoint A instantiated from the IOF template, whenever the destination of these outputs is equal to the domain of the viewpoint A.*

Finally, domain-independent inter-viewpoint rules were defined for the STD template. This facilitated the identification of various inconsistencies within the 1553 FDIR function, by explicitly stating relationships between the general 1553 FDIR behavioural model and the individual specifications of the different types of failure detections covered by this function. An example was the rule:

> (STD1) *For each external process X, if there exists a viewpoint A of type IOF which includes the process X, then for each path leading to X, there exists an identical set of inputs to X in the VP A.*

Domain-dependent in-viewpoint and inter-viewpoint rules were also defined. These represented either some explicit glossaries about domain-specific terms which were already given in the specifications, some domain-specific relationships between different terms which were elicitated by our analysis, or by some common sense interpretation of the English text descriptions (e.g., inter-viewpoint rules in Figure 2). A detailed account of these rules is given in [22].

The next section describes some of the inconsistencies identified by these rules for the process "Set Skip Bit" in the IOF-based viewpoint RT Failure Detection.

3.2 Identifying inconsistencies

The restructured representation of the two informal requirements documents, facilitated the detection and analysis of existing inconsistencies, expressed as violations of in-viewpoint and inter-viewpoint rules. The rules affected were among those added during the enriching phase of the approach, for which no explicit corresponding definition existed in the original specification. Manual inspections of these documents would therefore have made the identification of such inconsistencies more difficult.

One inconsistency identified related to the violation of the domain-independent in-viewpoint rule (IOF1) in the MDM Recovery Management viewpoint. Using the domain-dependent rule "C&C MDM is an MDM", the input "Reconfigure MDM" to the process Recovery 1, can be rewritten as "Reconfigure C&C MDM" whenever the MDM being considered is the C&C MDM. For this particular case, this input label to the process Recovery 1 becomes equal to the input label "Reconfigure C&C MDM" of the process Recovery 2, thereby violating the in-viewpoint rule (IOF1). Intuitively, the domain-dependent rule expresses a domain-specific overlap between the two terms MDMs and C&C MDM which implies that the function MDM Recovery Management is not able to choose which alternative process (Recovery 1 or

Recovery 2) to activate whenever there is a request to reconfigure the C&C MDM.

Various other inconsistencies were identified within the 1553 FDIR function – at least one for each type of failure detection. Only one example is described here, as the others were detected in a similar way by means of the same domain-independent inter-viewpoint rule of the 1553 FDIR viewpoint. The example involves the part of 1553 FDIR viewpoint described in Figure 2 and the RT Failure Detection viewpoint briefly described in section 3.1. In this case, it is the domain-independent rule (STD1) which was violated, where the process X is equal to "Set Skip Bit". It resulted in the paths leading to the process Set Skip Bit in the viewpoint 1553 FDIR (Figure 2) not corresponding to the sets of alternative inputs to the process Set Skip Bit given by the IOF viewpoints RT Failure Detection and the DFD viewpoint Set Skip Bit. A more detailed description of the reasoning process used to detect such inconsistency is given in [22]. The violation of the (STD1) domain-independent inter-viewpoint rule indicated an inconsistency between the 1553 FDIR and the RT Failure Detection requirements specifications.

3.3 Analysing evolutionary changes

We now discuss how we used the approach to manage evolutionary changes in the same informal specification documents. Our original specification document was the "June" version - we also examined two earlier versions (February and April) and one later version (August). For brevity, we only summarise our results below.

The first step was to examine if the requirements document structure was loosely preserved in each of these versions. The major changes we found were (i) the existence of an additional section on "MDM Recovery Management" in the April version, (ii) some paragraphs of the subsection "Bus Channel" in April's version becoming themselves subsections of the same level as the Bus Channel in June's version, and (iii) again in June's version, the fusion of different subsections specifying the MDM Initialisation function for each type of MDM, into only one general subsection referring to a table summarising the different type of MDMs. For these major changes, the hierarchic structure of viewpoints and the basic inter-viewpoint rules was, in general, preserved. In case (i) the addition of more specifications facilitated the identification of more inter-viewpoint rules for the IOF template. In case (ii), the hierarchic tree structure defined for April's version was changed by moving some viewpoints from being subnodes of the node "Bus Channel" to becoming, together with the Bus Channel, subnodes of the 1553 FDIR node. Of course wherever necessary, some of these viewpoints were reformulated with the correct template in order to preserve the inter-viewpoint rules. In case (iii), the hierarchic tree structure defined for April's version was further changed by joining together the viewpoints describing the MDM Initialisation function for each type of MDM into one viewpoint only. This determined the addition of a domain-dependent in-viewpoint rule which defined the domain of the possible MDMs.

Various other minor changes were made in these four versions of the specification. These were mainly changes within the specifications of some functions. Interesting ones included the changes made to the the 1553 FDIR flowchart and its associated text descriptions. In each version, the flowchart was gradually extended by introducing either more intermediate conditions or more external failure detections. Text descriptions of these failure detections were consequently updated or added. For each of these changes, using the inter-viewpoint rules, we were able to (1) identify the related parts of the specifications which had been affected by the changes, (2) define the types of changes that needed to be made in order to preserve consistency, and (3) map back into the specifications to check whether these changes were actually made to the specification.

Our investigation showed that, in many cases, additional inconsistencies were introduced by the evolutionary changes. The changes were not consistently traced within the specifications – variations were made in some parts of the document only, leaving other related specification fragments unchanged. We believe that many of these inconsistencies arose because the relationships between many specification fragments were left implicit. We believe that our restructuring approach helped overcome this problem by providing a way for explicitly representing many of these relationships as inter-viewpoint rules.

4 Lessons Learned

We now discuss the lessons we learned from this experience, indicating how the approach facilitated our analysis, inconsistency handling and change management.

4.1 Decomposition.

In general, different criteria could be adopted for decomposing the problem (requirements specification) into parts (viewpoints). For example, viewpoints could be defined for each individual component (or function) of the system described in the original specification, for each representation scheme used in the specification, or for each section and subsection of the specification (that is, decomposing using the existing *document structure*). Since we were not experts in the domain of the case study, the first option was clearly not easy to adopt. Moreover, the second option of decomposing on the basis of the representation schemes used in the specification would have lead to an insufficiently fine granularity for the structure to be useful[3]. On the other hand, our experience suggests that the document structure of such specifications reflects a first level of decomposition into major subsystems, and it was also generally preserved as the specifications evolved.

> **Lesson 1:** *Decomposition based on an existing document structure did not require any expertise in the domain of the specification, and provided a path back to the original specification after analysis and processing was performed on the new structure. Moreover, the resulting structure was generally stable as the original specification documents evolved.*

4.2 Representation.

While translating (parts of) a specification does not, strictly speaking, fall within the realms of restructuring, there are advantages that may be accrued from such a translation process (some even argue that the most significant benefits gained from using formal methods are the result of the formalisation process itself! [21]). The second stage of our approach therefore allows the representation of the different viewpoint specifications in different notations which may include the original notation in which they were expressed. The choice of representation scheme for different viewpoints also involves the expression of some basic (domain-independent) rules to define the conditions for well-formedness and internal viewpoint consistency. Of course, new domain-independent rules were also added as the restructuring process proceeded and as a better understanding of the specifications was gained.

> **Lesson 2:** *A mixture of formal and informal representation schemes may be used to represent requirements as long as the relationships between them are explicitly defined.*

[3]It is often the case, as in our case study, that informal specifications are written, almost entirely, in natural language.

4.3 Enriching the viewpoint structure

As described in Section 3, the structure of a viewpoint-oriented specification is more than just a collection of different viewpoints. It is also the collection of inter-viewpoint rules that express the relationships between these viewpoints. Our approach distinguished between domain-independent and domain-dependent rules. Domain-independent rules expressed the relationships between viewpoint templates, such as the relationship between a parent "input-output flow" diagram and the child diagram that decomposes one of the parent's constituent processes. Many of these rules were defined when choosing or developing a viewpoint template representation scheme, but others were also elicited after some analysis of the actual specifications. For example, the inter-viewpoint rules (DFD1) and (STD1) were all added to the templates after a better understanding of the documents was gained through restructuring.

Domain-dependent rules, expressed the relationships between actual specification fragments, such as "ontological overlaps" [23] denoting synonyms in the problem domain. These were of different types. For instance, the domain-dependent rule in the MDM Recovery Management viewpoint stating that the C&CMDM is an MDM, expressed an in-viewpoint ontological overlap between the terms MDMs and C&C MDM, whereas the domain-dependent rule in the RT Failure Detection viewpoint stating one of the outputs of its processes to be equivalent to some inputs in the MDM Recovery Management viewpoint, expressed an "intersection" between the two different functions RT Failure Detection and MDM Recovery Management. This itself was based on the ontological overlap expressed by other domain-dependent inter-viewpoint rules defined in the RT Failure Detection viewpoint (see [22] for further details). Finally, the domain-independent rule (STD1) expressed an "inclusive" overlap between the general 1553 FDIR function and the function RT Failure Detection.

> **Lesson 3:** *Inter-viewpoint rules were the basis upon which consistency was checked and change propagation was managed. They were also the measure of 'completeness' in this context.*

4.4 Analysis.

Our case study suggested that many inconsistencies in a large monolithic informal specification include much information that is implicit. For example, the first inconsistency described in Section 3.2

was related to the implicit assumption that each process described in the MDM Recovery Management specification is assigned to different failure conditions. Similarly, the inconsistency between 1553 FDIR and RT Failure Detection viewpoints was related to the implicit common sense assumption that being the general failure detection behaviour of the 1553 FDIR specification ought to be reflected in the text description of each individual type of failure detection. The third type of inconsistency described in Section 3.2 was based on implicit assumptions about the existence of an interface between the RT Failure Detection and the MDM Recovery Management functions. The restructuring approach facilitated the identification of such implicit assumptions, defining them explicitly as rules. Inconsistencies were then identified by either checking these rules, or by using them to integrate parts of the specification in order to prove their consistency.

What added confidence to our analysis was that some of the inconsistencies identified by our study were also identified by NASA's Independent Validation & Verification (IV&V) group, who used a different approach based on modelling informal specifications using formal methods such as SCR and PVS [6]. While the results of NASA's and our approaches are comparable – similar inconsistencies were identified in the 1553 FDIR function – we believe that our approach provided a much "cheaper" alternative to wholesale formalisation of the original specification documents.

Nevertheless, given a structure of related viewpoints that represents a requirements specification, different analysis techniques may be deployed to validate the requirements and to check their consistency. In previous work for example [13], we used an adaptation of classical logic to perform some kinds of analysis, while in other work [5] informal inspection was sufficient. The approach we used in this case study combined informal inspections with some automated support using Prolog.

> **Lesson 4:** *The viewpoints structure made the specifications more amenable to analysis (both manual and automated).*

4.5 Inconsistency handling and change management

The expectation is that analysis will necessitate taking action to handle inconsistencies [8]. Acting in the presence of inconsistency is still an open research issue, so we adopted a simple approach of consulting the domain experts when faced with such inconsistencies.

Nevertheless, we found that the identification and explicit formalisation of relationships within and between parts of the specification facilitate change management. The analysis of different versions of the same documents illustrated that additional inconsistencies were often caused whenever a change was made to the specifications. The lack of explicit relationships within the original specifications meant that any addition of new specification information was often not propagated to related parts of that specification. For example, in each version of the original specifications, extensions of the flowchart of the 1553 FDIR function were not propagated in the related failure detection text descriptions, and vice-versa. Moreover, changes made in order to resolve existing inconsistencies also caused additional inconsistencies [7]. An example was in June's version of the 1553 FDIR function, where the flowchart was changed to solve some inconsistencies with the text descriptions present in the April version, but the related text description was not changed consistently, thus causing the additional inconsistencies described in Section 3.2. The explicit relationships between the 1553 FDIR viewpoint and the viewpoints of the text descriptions (e.g., RT Failure Detection) would have helped avoid such additional inconsistencies, so guaranteeing a consistent evolution of the documents.

> **Lesson 5:** *In-viewpoint and inter-viewpoint rules (whether domain-independent or domain-dependent) provided explicit information which was often missing from the original specifications and which was needed for consistent change management. Indeed, the impact of evolutionary changes could be traced, using these rules to check consistent change propagation to related parts.*

As with the analysis, the advantage of this approach is that it is not expensive in either time or effort. In our case study, the document structure was generally preserved. Thus, evolutionary changes like addition, fusion or elimination of chunks of specification did not imply a major reorganisation of the document structure. Consequently, once a restructured representation of a version was developed, the modelling of subsequent versions was built upon this representation. Other methods, such as the formal modelling approaches used in [6], are much more costly in that the modelling of new versions requires a re-modelling of the specifications.

We also learned from our experience of automating parts of our approach. The natural language representation style of the informal specifications did not allow a fully automated process for restructuring and representing these documents into viewpoints. A manual process was needed to provide the viewpoints structure and the related rules. However, the resulting viewpoint structure was far more amenable to translation into formal languages (such as Prolog) to provide automated support for consistency checking. This was easier to achieve than translating the original natural language specification into a formal language. Moreover, given that changes to the specifications were, in most cases, not drastic, the restructured representation was preserved during the evolution of the documents.

> **Lesson 6:** *The manual effort of representing the specifications as viewpoints needed to be invested only once and then the specifications could be analysed semi-automatically in the subsequent versions.*

5 Related Work

A number of related approaches have used (semi) formal notations and partial formal modelling to analyse and validate existing informal requirements specifications [1]. For example, the work of Easterbrook *et al.* [6] describes the use of formal methods such as SCR [12] and PVS [20] to model the same informal requirements examined in this paper. This work describes selective modelling of the most critical parts of a requirements specification, and the testing of some critical properties. As in our work, the approach does not aim to guarantee completeness and correctness of the existing specifications, but to increase confidence in such specifications by identifying inconsistencies and feeding the results back into the requirements development process. The inconsistencies identified and discussed in [6] include some of the same inconsistencies identified by our own restructuring approach, as well as some additional inconsistencies relating to the dynamic properties of the underlying system.

Our restructuring approach, however, could be used in conjunction with other formal modelling approaches (such as SCR), by mapping particular viewpoints into the chosen formal model (such as an SCR model). The Safety Checklists proposed by Lutz [17], for example, could also complement our approach. Checklists of properties about interface requirements in safety-critical systems could be used to enrich our restructuring process by modelling them as in-viewpoint and inter-viewpoint domain-dependent rules.

Of course, restructuring specifications has been a topic of considerable interest and research in the past. Most notably, Statecharts [10] attempt to reduce the complexity of specifications modelled as state-transition diagrams by using clustering of states into superstates. Statecharts have been used successfully by Heimdahl, Leveson and others [11, 15] to restructure formal, state-based specifications. While such approaches provide powerful (and automated) tools to analyse formal specifications, they are not suitable for analysing informal specifications that may deploy multiple representation schemes, such as the ones we have described in this paper.

Finally, recent work on formal requirements analysis [3] recognises the need to devise practical ways of structuring the V&V process in order to make it methodologically suitable for large scale analysis. The work described in this paper is a first step in this direction. The use of "lightweight" formal modelling [14] has also been proposed as a complementary approach to the kind of analysis we have described.

6 Conclusions and Future Work

This paper has described our experiences in *restructuring* multi-perspective requirements specifications in order to identify and analyse inconsistencies and manage change. Our approach comprises of three main activities: (1) decomposing specifications into parts, (2) representing these parts within viewpoints using of different representation styles, and (3) enriching the resulting structure of viewpoints by identifying and explicitly defining in- and inter-viewpoint rules that express specific relationships between different templates, and domain-specific properties within and between the different specification fragments. Using these we were able to check consistency and completeness, and to track and analyse evolutionary changes in the original requirements specification.

While a large part of our approach can be (and was) performed manually, it is nevertheless amenable to automated support. We developed a toolkit to support viewpoint development and inconsistency analysis. The tool provides a web-based front-end (written in Java) which facilitates the specification of viewpoints by using a set of predefined templates. For each of these templates, a class of domain-independent in-viewpoint and inter-viewpoint rules were predefined and subsequently checked by a Prolog engine. Different graphical viewpoint instantiations are converted at run-time into Prolog and checked for consistency. Changes to viewpoints are made using the web-based tool and also checked for consistency using the Prolog reasoning engine.

At the moment, the toolkit is limited in that (i) the Prolog engine presents the results of its consistency checking in Prolog format, and (ii) the domain-dependent rules are hard-coded into the system. We are currently extending this to allow users to get the results of consistency checking via the same web-based front-end and to define domain-dependent rules themselves. Thus, the user should be able to specify the related domain-dependent in-viewpoint and inter-viewpoint rules during the instantiation process of viewpoints, to check consistency, and to get information about existing inconsistencies in the same web-based view.

We believe that the framework we have presented in this paper provides us with the infrastructure for organising specifications, detecting and analysing inconsistencies and managing change. Clearly, further work is needed to determine how to act in the presence of inconsistencies. However, we believe that incremental contributions towards this long term goal can be achieved by complementing our approach with existing work on inconsistency handling that combines both formal (e.g., logic-based) [13] and human-centered [5, 2] approaches.

Acknowledgement

We would like to thank Steve Easterbrook (NASA/WVU) for providing us with the case study, and for his patient and helpful answers to our clarification questions. Thanks also to John Hinkle (NASA) for his feedback on earlier versions of the paper, and to Alex Medina for implementing the Java front-end of the Prolog toolkit. The research described in the paper was funded by the UK EPSRC projects VOILA (GR J15483) and MISE (GR/L 55964). We are also grateful for financial support from NASA's IV&V Facility, West Virginia, USA, and from Texas Instruments Inc. for the "Requirements Architectures" project.

References

[1] D. Craigen, S. Gerhart and T. Ralston, "Formal Methods Reality Check: Industrial Usage", *IEEE Trans. on Soft. Eng.*, 21(2):90-98, Feb. 1995.

[2] G.P. Cugola, E. Di Nitto, A. Fuggetta, C. Ghezzi, "A Framework for Formalizing Inconsistencies in Human-Centered Systems", *ACM Trans. on Soft. Eng. and Meth.*, Sept. 1996.

[3] B. Dutertre and V Stavridou, "Formal Requirements Analysis of an Avionics Control System", *IEEE Trans. on Soft. Eng.*, 23(5):267-278, May 1997.

[4] S. Easterbrook and J. Callahan, "Formal Methods for V&V of Partial Specifications: An Experience Report", *IEEE Proc. of 3rd Int. Symp. on Req. Eng. (RE'97)*, 160-168, 1997.

[5] S. Easterbrook, A. Finkelstein, J Kramer and B. Nuseibeh, "Co-ordinating Distributed ViewPoints: The anatomy of a consistency check", *Concurrent Engineering: Research and Applications*, 2(3): 209-222, CERA Institute West Bloomfield, USA, 1994.

[6] S. Easterbrook, R. Lutz, J.K. Covington, J.Kelly, M. Ampo and D. Hamilton, "Experiences using Formal Methods for Requirement Modeling", NASA/WVU Software Research Lab, Fairmont, WV, Techn. Report # NASA-IVV-96-018, 1996.

[7] S. Easterbrook and B. Nuseibeh, "Using ViewPoints for Inconsistency Management", *BCS/IEE Soft. Eng. Journal*, 11(1): 31-43, Jan. 1996.

[8] A. Finkelstein and D. Gabbay and A. Hunter and J. Kramer and B. Nuseibeh, "Inconsistency Handling in Multiperspective Specifications", *IEEE Trans. on Soft. Eng.*, 20(8): 569-578, August 1994.

[9] A. Finkelstein and J. Kramer and B. Nuseibeh and L. Finkelstein and M. Goedicke, "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development", *Int. Journal of Soft. Eng. and Know. Eng.*, 2(1):31-58, World Scientific Publishing Co., March 1992.

[10] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming.*, 8: 231-274, 1987.

[11] M.P.E. Heimdahl and N.G. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements", *IEEE Trans. on Soft. Eng.*, 22(6):363-377, June 1996.

[12] C.L. Heitmeyer, B. Labaw and D. Kiskis, "Consistency Checking of SCR-Style Requirements Specifications", *2nd IEEE Symp. on Req. Eng.*, York, UK, 27-29, March 1995.

[13] A. Hunter and B. Nuseibeh, "Analysing Inconsistent Specifications", *IEEE Proc. of 3rd Int. Symp. on Req. Eng. (RE'97)*, 78-86, 1997.

[14] D. Jackson and C. Damon, "Elements of Style: Analyzing a Software Design Feature with a Counterexample Detector", *IEEE Trans. on Soft. Eng.*, 22(7), July 1996.

[15] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl and B.E. Melhart, "Software Requirements Analysis for Real-time Process-Control Systems", *IEEE Trans. on Soft. Eng.*, 17(3):241-258, March 1991.

[16] J.C. Kelly, J.S. Sherif and J.Hops, "An Analysis of Defect Densities Found During Software Inspections", *Journal of Systems and Software*, 17: 111-117, 1992.

[17] R.R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical Embedded Systems", *IEEE Proc. of the 1st Int. Symp. on Req. Eng.*, San Diego, CA, Jan. 1993.

[18] G. Mullery, "CORE - a method for controlled requirements expression", *IEEE Proc. of 4th Int. Conf. on Soft. Eng. (ICSE-4)*, 126-135, 1979.

[19] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *IEEE Trans. on Soft. Eng.*, 20(10): 760-773, Oct. 1994.

[20] S. Owre, J. Rushby, N. Shankar and F. von Henke,' "Formal Verification for Fault Tolerant Architectures: Prolegomena to the Design of PVS", *IEEE Trans. on Soft. Eng.*, 21:107-125, 1995.

[21] J. Rushby, "Formal Methods and their role in certification of critical systems", Computer Science Laboratory, SRI International, USA, Techn. Report CSL-95, 1995.

[22] A. Russo, B. Nuseibeh and J. Kramer, "Restructuring Requirements Specifications for Managing Inconsistency and Change: A Case Study", Techn. Report 98/1, Dept. of Computing, Imperial College, London, U.K.

[23] G. Spanoudakis and A. Finkelstein, "Reconciling requirements: a method for managing interference, inconsistency and conflict", *Annals of Soft. Eng.*, Special Issue on Soft. Req. Eng., 1997. Available from http://www.cs.city.ac.uk/homes/-gespan/publications.html.